A parameter-less algorithm for tensor co-clustering

Elena Battaglia[1] · Ruggero G. Pensa[1]

Abstract

The majority of the data produced by human activities and modern cyber-physical systems involve complex relations among their features. Such relations can be often represented by means of tensors, which can be viewed as generalization of matrices and, as such, can be analyzed by using higher-order extensions of existing machine learning methods, such as clustering and co-clustering. Tensor co-clustering, in particular, has been proven useful in many applications, due to its ability of coping with n-modal data and sparsity. However, setting up a co-clustering algorithm properly requires the specification of the desired number of clusters for each mode as input parameters. This choice is already difficult in relatively easy settings, like flat clustering on data matrices, but on tensors it could be even more frustrating. To face this issue, we propose a new tensor co-clustering algorithm that does not require the number of desired co-clusters as input, as it optimizes an objective function based on a measure of association across discrete random variables (called Goodman and Kruskal's τ) that is not affected by their cardinality. We introduce different optimization schemes and show their theoretical and empirical convergence properties. Additionally, we show the effectiveness of our algorithm on both synthetic and real-world datasets, also in comparison with state-of-the-art co-clustering methods based on tensor factorization and latent block models.

Keywords Clustering · Higher-order data · Unsupervised learning

1 Introduction

The increasing complexity of the data produced by humans and cyber-physical systems requires more sophisticated machine learning algorithms able to handle it and take advantage of the manifold of the variable space. This phenomenon also affects data structures that, on the one hand, should adapt to large datasets and, on the other hand, should be able

✉ Ruggero G. Pensa
ruggero.pensa@unito.it

Elena Battaglia
elena.battaglia@unito.it

[1] Department of Computer Science, University of Turin, Turin, Italy

to represent complex relations among data instances. A clear example of such evolution is certainly constituted by tensors, which have gained much attention in the last twenty years.

Tensors are widely used mathematical objects that well represent complex information such as gene expression data (Zhao and Zaki 2005), social networks (Hong and Jung 2018), heterogenous information networks (Ermis et al. 2015; Yu et al. 2019), time-evolving data (Araujo et al. 2018), behavioral patterns (He et al. 2018), and multi-lingual text corpora (Papalexakis and Dogruöz 2015). In general, every n-ary relation can be easily represented as a tensor. From the algebraic point of view, in fact, they can be seen as multimodal generalizations of matrices and, as such, can be processed with mathematical and computational methods that generalize those usually employed to analyze data matrices, e.g., non-negative factorization (Shashua and Hazan 2005), singular value decomposition (Zhang and Golub 2001), itemset and association rule mining (Cerf et al. 2009; Nguyen et al. 2011; Cerf et al. 2013), clustering and co-clustering (Banerjee et al. 2007; Wu et al. 2016).

Clustering, in particular, is by far one of the most popular unsupervised machine learning techniques since it allows analysts to obtain an overview of the intrinsic similarity structures of the data with relatively little background knowledge about them. However, with the availability of high-dimensional heterogenous data, co-clustering has gained popularity, since it provides a simultaneous partitioning of each mode (rows and columns of the matrix, in the two-dimensional case). In practice, it copes with the curse of dimensionality problem by performing clustering on the main dimension (data objects or instances) while applying dimensionality reduction on the other dimension (features). Despite its proven usefulness, the correct application of tensor co-clustering is limited by the fact that it requires the specification of a congruent number of clusters for each mode, while, in realistic analysis scenarios, the actual number of clusters is unknown. Furthermore, matrix/tensor (co-)clustering is often based on a preliminary tensor factorization step that, in its turn, requires further input parameters (e.g., the number of latent factors within each mode). As a consequence, it is merely impossible to explore all combinations of parameter values in order to identify the best clustering results.

The main reason for this problem is that most clustering algorithms (and tensor factorization approaches) optimize objective functions that strongly depend on the number of clusters (or factors). Hence, two solutions with two different numbers of clusters can not be compared directly. Although this considerably reduces the size of the search space, it prevents the discovery of a better partitioning once a wrong number of clusters is selected. In this paper, which extends our previous work (Battaglia and Pensa 2019), we address this limitation by proposing a new tensor co-clustering algorithm that optimizes a new class of objective functions that can be viewed as n-modal extensions of an association measure called Goodman-Kruskal's τ (Goodman and Kruskal 1954), whose local optima do not depend on the number of clusters. We model our tensor co-clustering approach as a multi-objective optimization problem and discuss, both theoretically and experimentally, the convergence properties of our extensions and of the related optimization schemes. Additionally, we conduct a thorough experimental validation showing that our algorithms provide accurate clustering results in each mode of the tensor. Compared with state-of-the-art techniques that require the desired number of clusters in each mode as input parameters, it achieves similar or better results at the price of a reasonable increase of the running time. Additionally, it is also effective in clustering real-world datasets.

In summary, the main contributions of this paper are as follows: (1) we define a new class of objective functions for n-mode tensor co-clustering, based on Goodman-Kruskal's τ association measure, which do not require the number of clusters as input parameter;

(2) we propose several variants of a multi-objective optimization algorithm, based on stochastic local search, and study their convergence properties showing that they support the rapid convergence towards a local optimum; (3) we show the effectiveness of our method experimentally on both synthetic and real-world data, also in comparison with state-of-the-art competitors.

The remainder of the paper is organized as follows: the related works are analyzed in Sect. 2; the generalization of the Goodman-Kruskal's τ association measure is presented in Sect. 3 while the variants of the optimization algorithms are described in Sect. 4; Sect. 5 provides the report of our experiments; finally, we draw some conclusions in Sect. 6.

2 Related work

Analyzing multi-way data (or n-way tensors) has attracted a lot of attention due to their intrinsic complexity and richness. Hence, to deal with this complexity, in the last two decades, many *ad hoc* methods and extensions of 2-way matrix methods have been proposed, many of which are tensor decomposition models and algorithms (Kolda and Bader 2009). As an example, both singular value decomposition (Zhang and Golub 2001) and non-negative matrix factorization (Shashua and Hazan 2005) have been extended to work with high-order tensor data. Furthermore, knowledge discovery and exploratory data mining techniques, including closed itemset mining (Cerf et al. 2009, 2013) and association rule discovery (Nguyen et al. 2011), have been successfully applied to n-way data as well.

The problem of clustering and co-clustering of higher-order data has also been extensively addressed. Co-clustering has been developed as a matrix method and studied in many different application contexts including text mining (Dhillon et al. 2003; Pensa et al. 2014), gene expression analysis (Cho et al. 2004) and graph mining (Chakrabarti et al. 2004) and has been naturally extended to tensors for its ability of handling n-modal high-dimensional data well. Banerjee et al. (2007) perform clustering using a relation graph model that describes all the known relations between the modes of a tensor. Their tensor clustering formulation captures the maximal information in the relation graph by exploiting a family of loss functions known as Bregman divergences. They also present several structurally different multi-way clustering schemes involving a scalable algorithm based on alternate minimization. Instead, Zhou et al. (2009) use tensor-based latent factor analysis to address co-clustering in the context of web usage mining. Their algorithm is executed via the well-known multi-way decomposition algorithm called CANDECOMP/PARAFAC (Harshman 1970).

Papalexakis et al. (2013) formulate co-clustering as a constrained multi-linear decomposition with sparse latent factors. They propose a basic multi-way co-clustering algorithm exploiting multi-linearity using Lasso-type coordinate updates. Additionally, they propose a line search optimization approach based on iterative majorization and polynomial fitting. Zhang et al. (2013) propose an extension of the tri-factor non-negative matrix factorization model (Ding et al. 2006) to a tensor decomposition model performing adaptive dimensionality reduction by integrating the subspace identification and the (hard or soft) clustering process into a single process. Their algorithm computes two basis matrices representing the common characteristics of the samples and one 3-D tensor denoting the peculiarities of the samples. The model can be used to perform dimensional reduction as well. Instead, Wu

et al. (2016) introduce a spectral co-clustering method based on a new random walk model for nonnegative square tensors.

Other more recent approaches (Boutalbi et al. 2019a, b) rely on an extension of the latent block model. In these works, co-clustering for sparse tensor data is viewed as a multi-way clustering model where each slice of the third mode of the tensor represents a relation between two sets. Finally, Wang and Zeng (2019) present a co-clustering approach for tensors by using a least-square estimation procedure for identifying n-way block structure that applies to binary, continuous, and hybrid data instances.

Differently from all these approaches, our tensor co-clustering algorithm is not based on any factorization method or block model hypothesis. Instead, it optimizes an extension of a measure of association whose effectiveness has been proven in matrix (2-way) clustering (Huang et al. 2012) and co-clustering (Ienco et al. 2013), and that naturally helps discover the correct number of clusters in tensors with arbitrary shape and density. It is worth noting, in fact, that the co-clustering performances of all the methods mentioned in this section strongly rely on the correct choice of the number of clusters/factors/blocks, which limits their application in realistic data analysis scenarios.

3 An association measure for tensor co-clustering

In this section, we introduce the objective function optimized by our tensor co-clustering algorithm (presented in the next section). It consists in an association measure, called Goodman and Kruskal's τ (Goodman and Kruskal 1954), that evaluates the dependence between two discrete variables and has been used to assess the quality of 2-way co-clustering (Robardet and Feschet 2001) with good partitioning results. We generalize its definition to a n-mode tensor setting.

3.1 Goodman and Kruskal τ and its generalization

Goodman and Kruskal's τ (Goodman and Kruskal 1954) is an association measure that estimates the strength of the link between two discrete variables X and Y according to the proportional reduction of the error in predicting one of them knowing the other. In more details, let $x_1, \ldots, x_m$ be the values that variable X can assume, with probability $p_X(1), \ldots, p_X(m)$ and let $y_1, \ldots, y_n$ be the possible values Y can assume, with probability $p_Y(1), \ldots, p_Y(n)$. The error in predicting X can be evaluated as the probability that two different observations from the marginal distribution of X fall in different categories:

$$e_X = \sum_{i=1}^{m} p_X(i)(1 - p_X(i)) = 1 - \sum_{i=1}^{m} p_X(i)^2.$$

Similarly, the error in predicting X knowing that Y has value y_j is

$$e_{X|Y=y_j} = \sum_{i=1}^{m} p_{X|Y=y_j}(i|j)(1 - p_{X|Y=y_j}(i|j)) = 1 - \sum_{i=1}^{m} p_{X|Y=y_j}(i|j)^2$$

and the expected value of the error in predicting X knowing Y is

$$\mathbb{E}[e_{X|Y}] = \sum_{j=1}^{n} e_{X|Y=y_j} p_Y(j)$$

$$= \sum_{j=1}^{n} (1 - \sum_{i=1}^{m} p_{X|Y=y_j}(i|j)^2) p_Y(j) = 1 - \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{p_{X,Y}(i,j)^2}{p_Y(j)}.$$

Then the Goodman and Kruskall $\tau_{X|Y}$ measure of association is defined as

$$\tau_{X|Y} = \frac{e_X - \mathbb{E}[e_{X|Y}]}{e_X} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \frac{p_{X,Y}(i,j)^2}{p_Y(j)} - \sum_{i=1}^{m} p_X(i)^2}{1 - \sum_{i=1}^{m} p_X(i)^2}.$$

Conversely, the proportional reduction of the error in predicting Y while X is known is

$$\tau_{Y|X} = \frac{e_Y - \mathbb{E}[e_{Y|X}]}{e_Y} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} \frac{p_{X,Y}(i,j)^2}{p_X(i)} - \sum_{j=1}^{n} p_Y(j)^2}{1 - \sum_{j=1}^{n} p_Y(j)^2}.$$

In order to use this measure for the evaluation of a tensor co-clustering, we need to extend it so that τ can evaluate the association of n distinct discrete variables. Let $X_1, \ldots, X_n$ be discrete variables such that X_i can assume m_i distinct values (for simplicity, we will denote the possible values as $1, \ldots, m_i$), for $i = 1, \ldots, n$. Let $p_{X_i}(k)$ be the probability that $X_i = k$, for $k = 1, \ldots, m_i$, for $i = 1, \ldots, n$. Reasoning as in the two-dimensional case, we can define the reduction in the error in predicting X_i while $(X_j)_{j \neq i}$ are all known as

$$
\begin{aligned}
\tau_{X_i} = \tau_{X_i|(X_j)_{j \neq i}} &= \frac{e_{X_i} - \mathbb{E}[e_{X_i|(X_j)_{j \neq i}}]}{e_{X_i}} \\
&= \frac{\sum_{k_1=1}^{m_1} \cdots \sum_{k_n=1}^{m_n} \frac{p_{X_1,\ldots,X_n}(k_1 \ldots k_n)^2}{p_{(X_j)_{j \neq i}}((k_j)_{j \neq i})} - \sum_{k_i=1}^{m_i} p_{X_i}(k_i)^2}{1 - \sum_{k_i=1}^{m_i} p_{X_i}(k_i)^2},
\end{aligned}
\tag{1}
$$

for all $i \leq n$. When $n = 2$, the measure coincides with Goodman-Kruskal's τ.

Notice that, in the n-dimensional case as well as in the 2-dimensional case, the error in predicting X_i knowing the value of the other variables is always positive and smaller or equal to the error in predicting X_i without any knowledge about the other variables. It follows that τ_{X_i} takes values between [0, 1]. It will be 0 if knowledge of prediction of the other variables is of no help in predicting X_i, while it will be 1 if knowledge of the values assumed by variables $(X_j)_{j \neq i}$ completely specifies X_i.

3.2 Tensor co-clustering with Goodman-Kruskal's τ

Let $\mathcal{X} \in \mathbb{R}_+^{m_1 \times \cdots \times m_n}$ be a tensor with n modes and non-negative values. Let us denote with $x_{k_1 \ldots k_n}$ the generic element of $\mathcal{X}$, where $k_i = 1, \ldots, m_i$ for each mode $i = 1, \ldots, n$. A co-clustering $\mathcal{P}$ of $\mathcal{X}$ is a collection of n partitions $\{\mathcal{P}_i\}_{i=1,\ldots,n}$, where $\mathcal{P}_i = \cup_{j=1}^{c_i} C_j^i$ is a partition of the elements on the i-th mode of $\mathcal{X}$ in c_i groups, with $c_i \leq m_i$ for each $i = 1, \ldots, n$. Each co-clustering $\mathcal{P}$ can be associated to a tensor $\mathcal{T}^{\mathcal{P}} \in \mathbb{R}_+^{c_1 \times \cdots \times c_n}$, whose generic element is

$$t_{i_1 \ldots i_n} = \sum_{k_1 \in C^1_{i_1}} \sum_{k_2 \in C^2_{i_2}} \cdots \sum_{k_n \in C^n_{i_n}} x_{k_1 \ldots k_n}. \tag{2}$$

Consider now n discrete variables $X_1, \ldots, X_n$, where each X_i takes values in $\{C^i_1, \ldots C^i_{c_i}\}$. We can look at $\mathcal{T}^{\mathcal{P}}$ as the contingency n-modal table that empirically estimates the joint distribution of $X_1, \ldots, X_n$: the entry $t_{k_1 \ldots k_n}$ represents the absolute frequency of the event $(\{X_1 = C^1_{k_1}\} \cap \cdots \cap \{X_n = C^n_{k_n}\})$ and the frequency of $X_i = C^i_k$ is the marginal frequency obtained by summing all entries $t_{k_1 \ldots k_{i-1} k k_{i+1} \ldots k_n}$, with $k_1, \ldots, k_{i-1}, k_{i+1}, \ldots, k_n$ varying trough all possible values and the i-th index k_i fixed to k. In the same way, we can compute the frequency of the event $(\{X_i = C^i_k\} \cap \{X_j = C^j_h\})$ as the sum of all elements $t_{k_1 \ldots k_n}$ of $\mathcal{T}^{\mathcal{P}}$ having $k_i = k$ and $k_j = h$. More in general, we can compute the marginal joint distribution of $d < n$ variables as the sum of all the entries of $\mathcal{T}^{\mathcal{P}}$ having the indices corresponding to the d variables fixed to the values we are considering. For instance, given $\mathcal{T}^{\mathcal{P}} \in \mathbb{R}^{4 \times 3 \times 5 \times 2}_+$, the absolute frequency of the event $(\{X_1 = 3\} \cap \{X_3 = 4\})$ is

$$t^{(1,3)}_{(3,4)} = \sum_{k_2=1}^{3} \sum_{k_4=1}^{2} t_{3,k_2,4,k_4}.$$

From now on, we will use the newly introduced notation $t^{\mathbf{v}}_{\mathbf{w}}$ to denote the sum of all elements of a tensor having the modes in the upper vector $\mathbf{v}$ (in the example $(1, 3)$) fixed to the values of the lower vector $\mathbf{w}$ (in the example $(3,4)$). A formal definition of the scalar $t^{\mathbf{v}}_{\mathbf{w}}$ can result clunky: given a tensor $\mathcal{T} \in \mathbb{R}^{m_1 \times \cdots \times m_n}_+$ and two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}_+^{d}$, with dimension $d \leq n$, such that $v_j \leq n$, $v_i < v_j$ if $i < j$ and $w_i \leq m_{v_i}$ for each $i, j = 1, \ldots, d$, we will use the following notation

$$t^{\mathbf{v}}_{\mathbf{w}} = \sum_{k_{\bar{v}_1}=1}^{m_{\bar{v}_1}} \cdots \sum_{k_{\bar{v}_r}=1}^{m_{\bar{v}_r}} t_{e_1 \ldots e_n}$$

where $\bar{\mathbf{v}}$ is the vector of dimension $r = n - d$ containing all the integers $i \leq n$ that are not in $\mathbf{v}$ and $e_i = w_i$ if $i \in \mathbf{v}$ while $e_i = k_i$ otherwise.

Summarizing, given a tensor $\mathcal{X}$ with n modes and a co-clustering $\mathcal{P}$ over $\mathcal{X}$, we obtain a tensor $\mathcal{T}^{\mathcal{P}}$ that represents the empirical frequency of n discrete variables $X_1, \ldots, X_n$ each of them with c_i possible values (where c_i is the number of clusters in the partition on the i-th mode of $\mathcal{X}$). Therefore, we can derive from $\mathcal{T}^{\mathcal{P}}$ the probability distributions of variables $X_1, \ldots, X_n$ and substitute them in Eq. 1: in this way we associate to each co-clustering $\mathcal{P}$ over $\mathcal{X}$ a vector $\tau^{\mathcal{P}} = (\tau^{\mathcal{P}}_{X_1}, \ldots, \tau^{\mathcal{P}}_{X_n})$ that can be used to evaluate the quality of the co-clustering. In particular, for any $i, j \leq n$ and any $k_i = 1, \ldots, c_i$:

$$p_{X_1 \ldots X_n}(k_1, \ldots, k_n) = \frac{t_{k_1 \ldots k_n}}{T}, \quad p_{X_i}(k_1) = \frac{t^{(i)}_{(k_i)}}{T}, \quad p_{(X_j)_{j \neq i}}((k_j)_{j \neq i}) = \frac{t^{(j)_{j \neq i}}_{(k_j)_{j \neq i}}}{T},$$

where T is the sum of all entries of $\mathcal{T}^{\mathcal{P}}$. It follows that
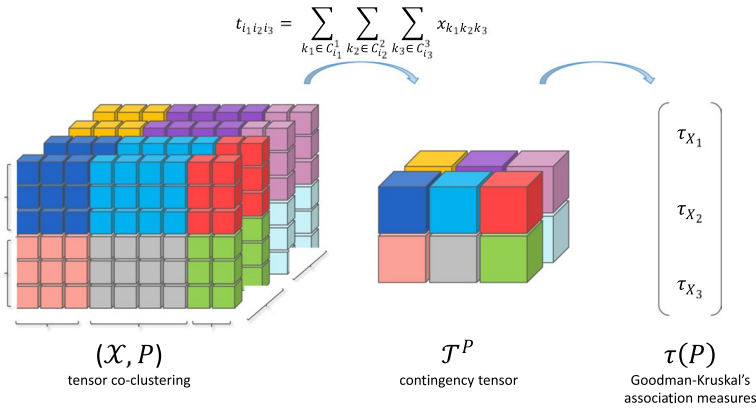
$$t_{i_1 i_2 i_3} = \sum_{k_1 \in C_{i_1}^1} \sum_{k_2 \in C_{i_2}^2} \sum_{k_3 \in C_{i_3}^3} x_{k_1 k_2 k_3}$$

$$\begin{pmatrix} \tau_{X_1} \\ \tau_{X_2} \\ \tau_{X_3} \end{pmatrix}$$

$(\mathcal{X}, P)$	$\mathcal{T}^P$	$\tau(P)$
tensor co-clustering	contingency tensor	Goodman-Kruskal's association measures

Fig. 1 An example of tensor co-clustering with the related contingency tensor and the associated Goodman-Kruskal's τ measures

$$\tau_{X_i}^{\mathcal{P}} = \frac{\sum_{k_1=1}^{c_1} \cdots \sum_{k_n=1}^{c_n} \frac{t_{k_1 \ldots k_n}^2}{t_{(k_j)_{j \neq i}}^{(i)} \cdot T} - \sum_{k_i=1}^{c_i} \frac{\left(t_{(k_i)}^{(i)}\right)^2}{T^2}}{1 - \sum_{k_i=1}^{c_i} \frac{\left(t_{(k_i)}^{(i)}\right)^2}{T^2}} \tag{3}$$

for each $i = 1, \ldots, n$. The overall co-clustering schema is depicted in Fig. 1.

It is worth pointing out that the procedure just described makes sense when the tensor $\mathcal{X}$ itself can be interpreted as a contingency tensor; the main assumption of our method is that the quantity $\frac{t_{k_1 \ldots k_n}}{T}$, where $t_{k_1 \ldots k_n}$ is the entry of $\mathcal{T}^P$ given by the sum of all the entries of $\mathcal{X}$ belonging to the same co-cluster, should be interpreted as a probability. This has to be true for each possible co-clustering $\mathcal{P}$ of $\mathcal{X}$, even for the discrete co-clustering (the co-clustering containing only singletons), whose contingency tensor $\mathcal{T}^P$ is $\mathcal{X}$. Typical tensors of this kind are those in which the n modes represent different variables, each element on a mode is a possible scenario (or value that the variable can assume), and each entry of the tensor is the count of the occurrences of the intersection of n scenarios. For instance, a words-documents matrix or an authors-words-conferences tensor are suitable choices. However, tensors of non-negative real numbers, in which all the entries represent homogeneous measurements of the same quantity under different scenarios, can also fit.

Suppose now we have two different partitions $\mathcal{P}$ and $\mathcal{Q}$ on the same tensor $\mathcal{X}$, corresponding to two different vectors $\tau^{\mathcal{P}}, \tau^{\mathcal{Q}} \in [0, 1]^n$. There is no obvious order relation in $[0, 1]^n$, so it is not immediately clear which one between $\boldsymbol{\tau}^{\mathcal{P}}$ and $\boldsymbol{\tau}^{\mathcal{Q}}$ is "better" than the other.

In Ienco et al. (2013), the authors, in order to compare partitions, adopt a dominance-based approach that induce a partial-order over $\mathbb{R}^n$. They introduce the notion of

Pareto-dominance for partitions and state that an optimal solution for the co-clustering problem is one that is not-dominated by any other solution. We formally define these concepts, in our tensor co-clustering framework, below.

Definition 1 (Pareto dominance) Let $\mathcal{X}$ be a n-modal tensor and let $\mathcal{P}$ and $\mathcal{Q}$ be two partitions on $\mathcal{X}$. We say that partition $\mathcal{P}$ dominates partition $\mathcal{Q}$, in symbols $\mathcal{P} \succ \mathcal{Q}$, if $\tau_{X_i}^{\mathcal{P}} \geq \tau_{X_i}^{\mathcal{Q}}$ for each $i = 1, \ldots n$ and there exists j such that $\tau_{X_j}^{\mathcal{P}} > \tau_{X_j}^{\mathcal{Q}}$.

Pareto dominance relation induces a partial order relation over the set $\mathbb{P}(\mathcal{X})$ of all partitions on $\mathcal{X}$. It means that, given two partitions $\mathcal{P}$ and $\mathcal{Q}$, we can always say whether $\mathcal{P}$ dominates $\mathcal{Q}$ or not, but it is possible that $\mathcal{P} \not\succeq \mathcal{Q}$ and $\mathcal{Q} \not\succeq \mathcal{P}$. As a consequence, it is not guaranteed that a unique maximum (with respect to relation $\succeq$) does exist in $\mathbb{P}(\mathcal{X})$.

Definition 2 (Pareto optimal partition) We say that a partition $\mathcal{P}$ on tensor $\mathcal{X}$ is a *Pareto-optimal partition* if $\mathcal{P}$ is not dominated by any other partition. In symbols, $\mathcal{P}$ is an optimal partition if $\mathcal{P} \not\prec \mathcal{Q}$ for any $\mathcal{Q} \in \mathbb{P}(\mathcal{T})$.

4 A stochastic local search approach to tensor co-clustering

Our co-clustering approach can be formulated as a multi-objective optimization problem: given a tensor $\mathcal{X}$ with n modes and dimension m_i on mode i, an optimal co-clustering $\mathcal{P}$ for $\mathcal{X}$ is one that is not dominated by any other co-clustering $\mathcal{Q}$ for $\mathcal{X}$. Since we do not fix the number of clusters, the space of possible solutions is huge (for example, given a very small tensor of dimension $10 \times 10 \times 10$, the number of possible partitions is 1.56×10^{15}): it is clear that a systematic exploration of all possible solutions is not feasible for a generic tensor $\mathcal{X}$. For this reason we need to find a heuristic that allows us to reach a "good" partition of $\mathcal{X}$, i.e. a partition $\mathcal{P}$ with high values of $\tau_{X_k}^{\mathcal{P}}$ for all modes k. With this aim, we propose a stochastic local search approach to solve the maximization problem.

4.1 Tensor co-clustering algorithm

Algorithm 1: $\tau TCC(\mathcal{X}, N_{iter})$

Input: A tensor $\mathcal{X}$ with n modes and shape $m_1 \times \cdots \times m_n$, the maximum number of iterations N_{iter}

Result: $\mathcal{P}_1, \ldots, \mathcal{P}_n$

1   Initialize $\mathcal{P}_1, \ldots, \mathcal{P}_n$ with discrete partitions;

2   $i \leftarrow 0$;

3   $m \leftarrow \max_{j=1,\ldots,n}(m_j)$;

4   $iter\_whithout\_moves \leftarrow 0$;

5   **while** $i \leq N_{iter}$ & $iter\_whithout\_moves < m$ **do**

6      $moves\_in\_iteration \leftarrow 0$;

7   **for** $k = 1$ *to* n **do**

8   **if** $iter\_whithout\_moves < t$ **then**

9   Randomly choose C_b^k in $\mathcal{P}_k$;

10   Randomly choose x in C_b^k;

11   **else**

12           $x \leftarrow next(x, k)$ //Select the element following the one selected at iteration $i - 1$ on mode k;

13           $C_b^k \leftarrow$ Cluster of x;

14   **end**

15   **for** C_k^j in $\mathcal{P}_k \cup \emptyset$ **do**

16           $\mathcal{Q}_k^j \leftarrow (\mathcal{P}_k \setminus \{C_b^k, C_j^k\}) \bigcup \{C_b^k \setminus \{x\}, C_j^k \cup \{x\}\}$;

17           $\mathcal{Q}^j \leftarrow (\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{k-1}, \mathcal{Q}_k, \mathcal{P}_{k+1}, \ldots, \mathcal{P}_n)$;

18   Compute contingency tensor T^j associated to partition $\mathcal{Q}^j$;

19   Compute $\tau^{\mathcal{Q}_j}$ using Equation (3) or Equation (4) //see section 4.2;

20   **end**

21         $e \leftarrow SelectBestPartition(k, b, (\tau^{\mathcal{Q}_j})_{j=1,\ldots,|\mathcal{P}_k \cup \emptyset|})$;

22         $\mathcal{P}_k \leftarrow Q_k^e$;

23   **if** $e! = b$ **then**

24           $moves\_in\_iteration \leftarrow moves\_in\_iteration + 1$;

25   **end**

26   **if** $moves\_in\_iteration > 0$ **then**

27         $iter\_whithout\_moves \leftarrow iter\_whithout\_moves + 1$;

28      $i \leftarrow i + 1$;

29   **end**

Algorithm 1 provides the general sketch of our tensor co-clustering algorithm, called τTCC . It repeatedly considers one mode by one, sequentially, and tries to improve the quality of the co-clustering by moving one single element from its original cluster to another cluster on the same mode. We will present in the following paragraphs different ways to measure the improvement in the quality of the partition at each iteration (function *SelectBestPartition* in Algorithm 1), but all the different approaches we will consider can be plugged in the general framework described in Algorithm 1 and explained below.

The partitions on each mode are initialized with the discrete partitions (each element stays in a cluster on its own). At each iteration *i*, fixed the *k*-th mode, the algorithm randomly selects one cluster C_b^k and one element $x \in C_b^k$. Then it tries to move x in every other cluster C_e^k and in the empty cluster $C_e^k = \emptyset$: among them, it selects the one that most improves the quality of the partition, according to the criterion chosen to measure it (see

Sect. 4.2). Of course, if there is not any move that increases the quality of the partition, the selected object is left in the original cluster C_b^k. When all the n modes have been considered, the i-th iteration of the algorithm is concluded. These operations are repeated until a stopping condition is met; we decide to stop the algorithm when no further moves are possible. Because of the stochasticity in the choice of the element to move at each iteration, we cannot be sure that all moves have been tried even if the algorithm has been stuck in the same solution for several iterations. For this reason, when the number of iterations without moves exceeds a given threshold (we set this threshold equal to the dimensionality of the largest mode), we change the object selection strategy and we select, sequentially, all the objects on all the modes. If all objects have been tried but no move is possible, the algorithm ends. Nonetheless, we also include a parameter N_{iter} to control the maximum number of iterations.

At the end of each iteration, one of the following possible moves has been done on mode k:

– an object x has been moved from cluster C_b^k to a pre-existing cluster C_e^k: in this case the final number of clusters on mode k remains the same (let us call it c_k) if C_b^k is non-empty after the move. If C_b^k is empty after the move, it will be deleted and the final number of clusters will be $c_k - 1$;
– an object x has been moved from cluster C_b^k to a new cluster $C_e^k = \emptyset$: the final number of clusters on mode k will be $c_k + 1$ (the useless case when x is moved from $C_b^k = \{x\}$ to $C_e^k = \emptyset$ is not considered);
– no move has been performed and the number of clusters remains c_k.

Thus, during the iterative process, the updating procedure is able to increase or decrease the number of clusters at any time. This is due to the fact that, contrary to other measures, such as the loss in mutual information (Dhillon et al. 2003), τ measure has an upper limit which does not depend on the number of co-clusters and thus enables the comparison of co-clustering solutions of different cardinalities.

4.2 Neighboring partition selection criteria

As seen above, our co-clustering framework tries to move one element in one fixed mode from its original cluster to another cluster which maximizes the quality improvement of the tensor partition. Since we need to optimize the set $\{\tau_{X_k}^{\mathcal{P}}\}_{k=1}^n$ of n objective functions (one for each mode of the tensor), we can define different ways to measure this increase, corresponding to different ways to implement function *SelectBestPartition* in Algorithm 1. Suppose the algorithm is performing step i of the algorithm: during this step, it considers the k-th mode of the tensor and selects an object x in cluster C_b^k. Function *SelectBestPartition* takes a set of candidate co-clusterings and their respective values of τ as input, and has to decide which of them is the best one. In the following, we provide the details of different selection strategies.

4.2.1 Alternating optimization of τ_{X_k}

Since all the candidate co-clusterings differ only in the partition on the k-th mode, we can look at the k-th partitions only and select the one with highest value of τ_k. In case of ties, the partition with the highest average τ is selected. The move is made only if $\tau_{X_k}^{Q_e} \geq \tau_{X_k}^{Q_b}$,

where $\mathcal{Q}_e$ and $\mathcal{Q}_b$ are the co-clusterings having $x \in C_e^k$ and $x \in C_b^k$ respectively (in the k-th partition), while the partitions on all the other modes of the tensor are the same. We call this strategy $SelectBestPartition_{ALT}$ (see Algorithm 2).

Algorithm 2: $SelectBestPartition_{ALT}(k, b, (\tau^{\mathcal{Q}_j})_{j=1,\ldots,c})$

Input: The mode k of the tensor, the original cluster b of the selected object,
  $(\tau^{\mathcal{Q}_j})j = 1, \ldots, c$ where each $\tau^{\mathcal{Q}_j}$ is a n-dimensional vector $(\tau_1^{\mathcal{Q}_j}, \ldots, \tau_n^{\mathcal{Q}_j})$
Result: e index of the selected partition among the c proposed

1   $BestPartitions \leftarrow argmax_{j=1,\ldots,c}(\tau_k^{\mathcal{Q}_j})$;
2   **if** $|BestPartitions| > 1$ **then**
3   Randomly select an index m in set $BestPartitions$;
4      $e \leftarrow m$;
5      $max_\tau \leftarrow avg(\tau^{\mathcal{Q}_m})$;
6   **for** h in $BestPartitions \setminus \{m\}$ **do**
7   **if** $avg(\tau^{\mathcal{Q}_h}) > max_\tau$ **then**
8            $e \leftarrow h$;
9            $max_\tau \leftarrow avg(\tau^{\mathcal{Q}_h})$;
10   **end**
11   **end**
12 **else**
13      $e \leftarrow$ the only index in $BestPartitions$;
14 **end**

The idea behind this selection strategy is that the alternating optimization of the single components τ_{X_k} should lead to a final vector $(\tau_{X_1}, \ldots, \tau_{X_n})$ with high values in each component.

However, this optimization strategy has a drawback: since the choice of the best move on mode k is done by looking only at the partition on the k-th mode, it is possible that, after the move, the overall quality of the co-clustering decreases. In Fig. 2 we propose a toy example to better explain this concept. Suppose we are applying our algorithm to a 2-way tensor (a matrix), having on the X mode all the clients of a shop and on the Y mode all the products sold. Each entry of the matrix represents the quantity of each product bought by each customer.

There are three well separated co-clusters in X: the first co-cluster consists of costumers who buy product 1, 2 and 3, the second co-cluster represents costumers who buy products 4 and 5, and last co-cluster includes customers who buy product 6.

After some iterations, the algorithm finds five clusters on the X axis and three on the Y axis, with the contingency matrix T of Fig. 2a. Then it selects the last row and tries to move it. There are five possible moves, as shown in Fig. 2b. τ_X has the highest value for $e = 0$ and, according to Algorithm 2, the last row goes in the first cluster, even if it is clear that the row is 'more similar' to those in clusters 3 and 4. Furthermore, after this move the algorithm will necessarily end with the partition having contingency table T_{final} in Fig. 2c, while it is evident that a "more desirable" co-clustering of X is $T_{correct}$ in Fig. 2c. This intuitive assessment is also confirmed by the fact that the average τ measured on $T_{correct}$ (0.771) is higher than the one measured on T_{final} (0.713).

The reason of this behavior is that the algorithm decides where to move the selected row by looking only at the value of τ_X. A more suitable choice would have been to move the last row in cluster 2 or 3, but this means that the algorithm has to look at τ_Y as well. Furthermore, we need a way to decide which combination of τ_X and τ_Y is preferable. In the following

$$X = \begin{bmatrix} 2 & 1 & 2 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \longrightarrow \quad T = \begin{bmatrix} 16 & 0 & 1 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 2 \\ 1 & 0 & 2 \end{bmatrix}$$

(a)

C_e	τ_X	τ_Y	$avg(\tau)$
C_0	0.710	0.671	0.690
C_1	0.591	0.604	0.597
C_2	0.655	0.782	0.718
C_3	0.692	0.775	0.733
C_4	0.618	0.792	0.705

(b)

$$T_{final} = \begin{bmatrix} 17 & 0 & 6 \\ 0 & 5 & 0 \end{bmatrix}, T_{correct} = \begin{bmatrix} 16 & 0 & 1 \\ 0 & 5 & 0 \\ 1 & 0 & 5 \end{bmatrix}.$$

(c)

Fig. 2 A 2-way tensor to be partitioned and the related contingency matrix obtained by Algorithm 2 after some iterations **a**. Rows 1, 2, and 3 are in the first cluster; rows 4 and 5 in the second cluster; all the other rows form singleton clusters. Columns 1, 2, and 3 are in the first cluster; columns 4 and 5 are in the second cluster; column 6 forms a singleton cluster. In **b**, the table reports the values of τ when moving the last row of X in any row cluster C_e of T. The final contingency tables are shown in (c): T_{final} is the contingency matrix obtained with Algorithm 2, $T_{correct}$ is a more desirable final result

subsections we will present some alternative optimization methods, with the aim of mitigating the issue illustrated above.

4.2.2 Optimization of $avg(\tau)$

A way to compare real-valued vectors is to use a scalarization function $\mathbb{R}^n \longrightarrow \mathbb{R}$ and to exploit the natural order in $\mathbb{R}$. Here, we use a function that maps each vector $\tau = (\tau_{X_1}, \ldots, \tau_{X_n})$ into a weighted sum $avg(\tau) = \sum_{i=1}^n w_i \tau_{X_i}$, with fixed $w_i = \frac{1}{n}$. Thus we can map the set $\mathbb{P}(\mathcal{X})$ of all the partitions over tensor $\mathcal{X}$ in $\mathbb{R}$, with the function $avg \circ \tau : \mathbb{P}(\mathcal{X}) \xrightarrow{\tau} \mathbb{R}^n \xrightarrow{avg} \mathbb{R}$ (where $\circ$ is the composition operator). As a consequence, $\mathbb{P}(\mathcal{X})$ inherits the total-order structure of $(\mathbb{R}, \leq)$ and it is always possible to decide which partition, among a finite set, is the best one.

The above consideration gives us a criterion to select a partition among the set of candidates proposed at each step of the algorithm: the best co-clustering $\mathcal{Q}$ is the one with the highest $avg \circ \tau(\mathcal{Q})$. This means that the selected element x on mode k is moved from its original cluster C_b^k to the cluster C_e^k which maximizes $avg(\tau)$. If there are several clusters $C_{e_1}^k, \ldots, C_{e_r}^k$ which maximize $avg(\tau)$, the arrival cluster is randomly selected among them. The move will be executed only if $avg(\tau^e) > avg(\tau^b)$. We call this strategy *SelectBestPartition_{AVG}* (see Algorithm 3).

Algorithm 3: $SelectBestPartition_{AVG}(k, b, (\tau^{Q_j})_{j=1,...,c})$

Input: The mode k of the tensor, the original cluster b of the selected object,
$(\tau^{Q_j})j = 1, ..., c$ where each τ^{Q_j} is a n-dimensional vector $(\tau_1^{Q_j}, ..., \tau_n^{Q_j})$
Result: e index of the selected partition among the c proposed

1 $BestPartitions \leftarrow argmax_{j=1,...,c}(avg(\tau^{Q_j}))$;
2 **if** $|BestPartitions| > 1$ **then**
3   Randomly select an index m in set $BestPartitions$;
4      $e \leftarrow m$;
5 **end**

This strategy has many theoretical advantages over the previous one: it works with a unique objective function and each solution is necessarily better than the previous solutions. Nevertheless, there is a disadvantage with this approach: by looking only at the partitions that increase the objective function $avg(\tau)$ we are reducing the search space. Therefore, there is a greater risk to getting stuck in a poor-quality local optimal solution. In fact, if there is no move that improves $avg(\tau)$, the algorithm ends with a sub-optimal partition $\mathcal{P}$, while with the alternating optimization strategy we would have been able to move from $\mathcal{P}$ and continue with the optimization, potentially reaching a final result with greater $avg(\tau)$. Furthermore, as we will show experimentally in Sect. 5, when an object on mode k is moved, usually the increase of τ_{X_k} is compensated by a decrease of (some of) the other τ_{X_j}, for $j \neq k$: this could be a serious issue when the number of modes n is elevated, because the decrease of $\sum_{j \neq k} \tau_{X_j}$ is often greater than the increase of the single τ_{X_k}, and the algorithm remains stuck in the initial discrete solution. Finally, this method is computationally more expensive than the previous one, because it requires the computation of all τ_{X_j}, while the alternating optimization strategy requires the computation of τ_{X_k} only.

4.2.3 Aggregate optimization of $\tau_{X_k|(X_j)_{j \neq k}} + \tau_{(X_j)_{j \neq k}|X_k}$

Algorithm 2 maximizes only τ_{X_k} when moving an object on mode k, ignoring all other τ_{X_i} ($i \neq k$). Instead, Algorithm 3 looks at the whole vector $\tau^{\mathcal{P}}$ and choose the partition which maximizes $avg(\tau)$. Here we propose an alternative method that stays in the between: it is an alternating maximization of the single τ_{X_k}, according to the mode k considered at the moment, but it adds a term $\tau_{(X_j)_{j \neq k}|X_k}$ to the objective function. This addend takes into account the aggregate modification of the other components of τ induced by the move on k-th mode. In more details, in Sect. 3.1 we have generalized the Goodman and Kruskal's τ measure to n modes as the reduction of the error in predicting one variable when all the other variables are known; we can also define another generalization, i.e. the reduction of the error in predicting the joint value of all the other variables when X_k is known. Reasoning as in Sect. 3.1, we have that

$$
\tau_{(X_j)_{j \neq k}|X_k} = \frac{e_{(X_j)_{j \neq k}} - \mathbb{E}[e_{(X_j)_{j \neq k}|X_k}]}{e_{(X_j)_{j \neq k}}}
$$
$$
= \frac{\sum_{h_1=1}^{c_1} \cdots \sum_{h_n=1}^{c_n} \frac{t_{h_1 \ldots h_n}^2}{t_{(h_k)}^{(k)} \cdot T} - \sum_{h_1=1}^{c_1} \cdots \sum_{h_{k-1}=1}^{c_{k-1}} \sum_{h_{k+1}=1}^{h+1} \cdots \sum_{h_n=1}^{c_n} t_{(h_j)_{j \neq k}}^{(j)_{j \neq k}}}{\sum_{h_1=1}^{c_1} \cdots \sum_{h_{k-1}=1}^{c_{k-1}} \sum_{h_{k+1}=1}^{c_{k+1}} \cdots \sum_{h_n=1}^{c_n} t_{(h_j)_{j \neq k}}^{(j)_{j \neq k}}}. \tag{4}
$$

The best partition among those considered by this strategy is the one with highest value of $\tau_{X_k|(X_j)_{j\neq k}} + \tau_{(X_j)_{j\neq k}|X_k}$. Again, the move is performed only if $\tau^{Q^e}_{X_k|(X_j)_{j\neq k}} + \tau^{Q^e}_{(X_j)_{j\neq k}|X_k} \geq \tau^{Q^b}_{X_k|(X_j)_{j\neq k}} + \tau^{Q^b}_{(X_j)_{j\neq k}|X_k}$ (in case of ties we look at the best $\tau_{X_k|(X_j)_{j\neq k}}$). In this way, we require that the best partition among the neighboring ones is one that increases τ_{X_i} with a decay of the quality of the partitions on the other modes that, overall, is less important than the improvement on mode k. We call this aggregate-based strategy $SelectBestPartition_{AGG}$ (see Algorithm 4).

Algorithm 4: $SelectBestPartition_{AGG}(k, b, (\tau^{Q_j})_{j=1,\dots,c})$

Input: The mode k of the tensor, The original cluster b of the selected object, $(\tau^{Q_j})j = 1, \dots, c$ where each τ^{Q_j} is a two-dimensional vector $(\tau^{Q_j}_k, \tau^{Q_j}_{(X_i)_{i=k}|X_k})$

Result: e index of the selected partition among the c proposed

1   $BestPartitions \leftarrow argmax_{j=1,\dots,c}(\tau^{Q_j})$;
2   **if** $|BestPartitions| > 1$ **then**
3   Randomly select an index m in set $BestPartitions$;
4     $e \leftarrow m$;
5     $max_\tau \leftarrow \tau^{Q_m}_k$;
6   **for** h in $BestPartitions \setminus \{m\}$ **do**
7   **if** $\tau^{Q_h}_k > max_\tau$ **then**
8        $e \leftarrow h$;
9        $max_\tau \leftarrow \tau^{Q_h}_k$;
10   **end**
11   **end**
12 **else**
13    $e \leftarrow$ the only index in $BestPartitions$;
14 **end**

4.2.4 Alternative alternating optimization of τ_{X_k}

All the three methods proposed above perform a move only when the respective objective function (τ_{X_k} in Algorithm 2, $avg(\tau)$ Algorithm 3 or $\tau_{X_k|(X_j)_{j\neq k}} + \tau_{(X_j)_{j\neq k}|X_k}$ in Algorithm 4) increases its value. If there is no move able to increase the value of the objective function, no move is done. Here we propose a slightly different strategy for the alternating optimization of τ. Suppose we are considering partition $\mathcal{P}$ and we want to move an object on mode k: we consider only those moves that improve (or at least do not worsen) τ_{X_k} and, among them, we choose the one with the greatest value of $avg(\tau)$. Notice that we do not require to increase the value of $avg(\tau)$ with respect to partition $\mathcal{P}$: we perform the move if there is any improvement (even little) of τ_{X_k} (as in Algorithm 2) and we choose the cluster with the highest $avg(\tau)$. Ties are solved in favor of the partition with the highest τ_{X_k}. This method is called $SelectBestPartition_{ALT2}$, and is sketched in Algorithm 5. As we will show in Sect. 5, this method usually achieves better results than the others and still exhibits a good convergence behavior.

Algorithm 5: $SelectBestPartition_{ALT2}(k, b, (\tau^{Q_j})_{j=1,...,c})$

Input: The mode k of the tensor, the initial cluster b of the selected object,
$(\tau^{Q_j})j = 1, ..., c$ where each τ^{Q_j} is a n-dimensional vector $(\tau_1^{Q_j}, ..., \tau_n^{Q_j})$
Result: e index of the selected partition among the c proposed

1   $P \leftarrow \{j | \tau_k^{Q_j} \geq \tau_k^{Q_b})\}$;
2   $BestPartitions \leftarrow argmax_{j=1,...,c}(avg(\tau^{Q_j})_{j \in P})$;
3   **if** $|BestPartitions| > 1$ **then**
4   Randomly select an index m in set $BestPartitions$;
5      $e \leftarrow m$;
6      $max_\tau \leftarrow \tau_k^{Q_m}$;
7   **for** h in $BestPartitions \setminus \{m\}$ **do**
8   **if** $\tau_k^{Q_h} > max_\tau$ **then**
9              $e \leftarrow h$;
10             $max_\tau \leftarrow \tau_k^{Q_h}$;
11   **end**
12   **end**
13 **else**
14      $e \leftarrow$ the only index in $BestPartitions$;
15 **end**

4.2.5 Alternative aggregate optimization of $\tau_{X_k | (X_j)_{j \neq k}} + \tau_{(X_j)_{j \neq k} | X_k}$

Finally, we propose a criterion (named *SelectBestPartition$_{AGG2}$*) based on the same selection strategy as the previous one, but applied to function $\tau_{X_k | (X_j)_{j \neq k}} + \tau_{(X_j)_{j \neq k} | X_k}$. More in details, the algorithm considers only the moves which improve τ_{X_k} and, among them, chooses the one with highest value of $\tau_{X_k | (X_j)_{j \neq k}} + \tau_{(X_j)_{j \neq k} | X_k}$. The strategy is described in Algorithm 6.

Algorithm 6: $SelectBestPartition_{AGG2}(k, b, (\tau^{\mathcal{Q}_j})_{j=1,\ldots,c})$

Input: The mode k of the tensor, the initial cluster b of the selected object,
$(\tau^{\mathcal{Q}_j})j = 1, \ldots, c$ where each $\tau^{\mathcal{Q}_j}$ is a two-dimensional vector
$(\tau_k^{\mathcal{Q}_j}, \tau_{(X_i)_{i=k}|X_k}^{\mathcal{Q}_j})$

Result: e index of the selected partition among the c proposed

1   $P \leftarrow \{j | \tau_k^{\mathcal{Q}_j} \geq \tau_k^{\mathcal{Q}_b})\}$;
2   $BestPartitions \leftarrow argmax_{j=1,\ldots,c}(sum(\tau^{\mathcal{Q}_j})_{j \in P})$;
3   **if** $|BestPartitions| > 1$ **then**
4   Randomly select an index m in set $BestPartitions$;
5      $e \leftarrow m$;
6      $max_\tau \leftarrow \tau_k^{\mathcal{Q}_m}$;
7   **for** h in $BestPartitions \setminus \{m\}$ **do**
8   **if** $\tau_k^{\mathcal{Q}_h} > max_\tau$ **then**
9            $e \leftarrow h$;
10            $max_\tau \leftarrow \tau_k^{\mathcal{Q}_h}$;
11   **end**
12   **end**
13 **else**
14      $e \leftarrow$ the only index in $BestPartitions$;
15 **end**

In the remainder of the paper, we refer to the five selection strategies as **ALT** (for Algorithm 2), **AVG** (for Algorithm 3), **AGG** (for Algorithm 4), **ALT2** (for Algorithm 5), and **AGG2** (for Algorithm 6).

4.3 Local convergence of τTCC

A partition $\mathcal{P}$ is locally optimal with respect to a set of neighboring solutions $\mathcal{N}(\mathcal{P})$ if $\mathcal{P}$ is not dominated by any other solution $\mathcal{Q} \in \mathcal{N}(\mathcal{P})$. In Ienco et al. (2013) the authors show that their matrix co-clustering algorithm based on the multi-objective optimization of τ converges to a Pareto local optimum, with respect to the following neighboring function

$$\mathcal{N}_{k,b,x} : \mathcal{P} \longmapsto \{\mathcal{Q} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_n) | \mathcal{Q}_j = \mathcal{P}_j, \forall j \neq k, \text{ and } \exists e \neq b \text{ s.t.}$$
$$\mathcal{Q}_k = (\mathcal{P}_k \setminus \{C_b^k, C_e^k\}) \cup (C_b^k \setminus \{x\}) \cup (C_e^k \cup \{x\})\}.$$

Although the same property holds for τTCC as well, here we prove a slightly stronger local convergence property for three strategies, namely **ALT**, **AVG** and **ALT2**.

Theorem 1 *If τTCC (with selection strategy* **ALT**, **AVG** *or* **ALT2***) ends within $t < N_{iter}$ iterations, then it returns a Pareto local optimum with respect to the following neighboring function*

$$\mathcal{N} : \mathcal{P} \longmapsto \{\mathcal{Q} = (\mathcal{Q}_1, \ldots, \mathcal{Q}_n) | \exists k \exists! x \text{ on mode } k \text{ s.t. } \mathcal{Q}_j = \mathcal{P}_j, \forall j \neq k \text{ and}$$
$$\mathcal{Q}_k = (\mathcal{P}_k \setminus \{C_b^k, C_e^k\}) \cup (C_b^k \setminus \{x\}) \cup (C_e^k \cup \{x\})\}$$

which considers, as neighboring partitions of $\mathcal{P}$, all those differing from $\mathcal{P}$ in the cluster assignment of a unique element x in a unique mode k.

Proof We demonstrate the property for every selection strategy.

– **AVG**. When the algorithm ends in less than N_{iter} iterations, all objects on all modes have been considered for a move, but no move been actually performed: this means that any co-clustering $\mathcal{Q}$ obtainable by moving one single element in one single mode has $avg(\tau^{\mathcal{Q}}) \leq avg(\tau^{\mathcal{P}})$. This implies that $\mathcal{P}$ is not dominated by $\mathcal{Q}$, for any $\mathcal{Q} \in \mathcal{N}(\mathcal{P})$, i.e., it is a Pareto local optimum w.r.t. $\mathcal{N}$
– **ALT**. Let $\mathcal{Q} \in \mathcal{N}(\mathcal{P})$. $\mathcal{Q}$ differs from $\mathcal{P}$ in the cluster assignment of a unique element x on mode k. Object x has been selected by the algorithm in one of the last $max_{i=1,\dots,n}(m_i)$ iterations, but no move has been done: it means that either $\tau_k^{\mathcal{P}} > \tau_k^{\mathcal{Q}}$ or $\tau_k^{\mathcal{P}} = \tau_k^{\mathcal{Q}}$ and $avg(\tau^{\mathcal{P}}) \geq avg(\tau^{\mathcal{Q}})$ (because ties are solved in favor of the partition with highest $avg(\tau)$). In both cases $\mathcal{P} \nprec \mathcal{Q}$. Thus $\mathcal{P}$ is a Pareto local optimum w.r.t. $\mathcal{N}$.
– **ALT2**. The proof is identical to the ALT case. □

While the convergence to a local optimum w.r.t. neighboring function $\mathcal{N}_{k,b,x}$ is always guaranteed, the convergence w.r.t. neighboring function $\mathcal{N}$ can be proved only when the algorithm ends within $t < N_{iter}$ iterations. As a rule of thumb, we suggest to set N_{iter} equal to ten times the sum of the dimensions on all the modes of the tensor. According to our experiments this is a "safe" threshold: although there is no theoretical prove that the algorithm will reach the convergence within this number of iterations, it always happens in our exeriments and with a large margin of tolerance (see Sect. 5.2).

4.4 Optimized computation of τ

In step 19 of Algorithm ,1 fixed a mode k, the following quantities are computed:

$$\tau_{X_j}^{\mathcal{Q}_e} \qquad \text{for each } j = 1, \dots n, \text{ for each } e = 1, \dots c_k$$

where c_k is the number of clusters on mode k (including the empty set) and $\mathcal{Q}^e$ is the co-clustering obtained by moving an object x from cluster C_b^k to cluster C_e^k.

A way to compute these quantities is to fix an arrival cluster C_e^k, move x in C_e^k obtaining partition $\mathcal{Q}^e$, compute the contingency tensor $\mathcal{T}^e$ associated to that partition (using Eq. 2) and compute vector τ^e associated to tensor $\mathcal{T}^e$ (using Eq. 3 for strategies **ALT**, **ALT2**, and **AVG** and, additionally, Eq. 4 for **AGG** and **AGG2**). By repeating these steps for every $e \in \{1, \dots, c_k\}$, we obtain a matrix $\mathcal{V} = (\tau_{X_j}(T^e))_{ej}$ of shape $c_k \times n$. If the variant of the algorithm is **AGG** or **AGG2**, matrix $\mathcal{V}$ has shape $c_k \times 2$, because instead of computing all the τ_{X_j} we only compute τ_{X_k} and $\tau_{(X_j)_{j \neq k}|X_k}$, where k is the mode considered at that moment. Then we pass matrix $\mathcal{V}$ as input to one of the variants of function *SelectBestPartition*, which determines where to move x. In order to obtain $\mathcal{V}$ in a more efficient way, we can reduce the amount of calculations by only computing the variation of τ^e from one step to another. We take advantage of the fact that a large part in the τ formula remains the same when moving a single element from a cluster to another. Hence, an important part of the computation of τ can be saved.

Imagine that x has been selected in cluster C_b^1 and that we want to move it in cluster C_e^1 (for simplicity we consider x on the first mode, but all the computations below are analogous on any other mode k). Object x is a row on the first mode (let's say the j-th row) of tensor $\mathcal{X}$ and so x can be expressed as a tensor $\mathcal{M} \in \mathbb{R}_+^{m_2 \times \dots \times m_n}$ with $n-1$ modes, whose generic entry is $\mu_{k_2 \dots k_n} = x_{jk_2 \dots k_n}$. We will denote with M the sum of all elements

of $\mathcal{M}$. Let $\mathcal{T}$ and $\tau(\mathcal{T})$ be the tensor and the measure associated to the initial co-clustering and $\mathcal{S}$ and $\tau(\mathcal{S})$ the tensor and the measure associated to the final co-clustering obtained after the move. Tensor $\mathcal{S}$ differs from $\mathcal{T}$ only in those entries having index $k_1 \in \{b, e\}$. In particular, for each $k_i = 1, \ldots, c_i$ and $i = 2, \ldots, n$:

$$
\begin{aligned}
s_{bk_2 \ldots k_n} &= t_{bk_2 \ldots k_n} - \mu_{k_2 \ldots k_n} \\
s_{ek_2 \ldots k_n} &= t_{ek_2 \ldots k_n} + \mu_{k_2 \ldots k_n} \\
s_{k_1 k_2 \ldots k_n} &= t_{k_1 k_2 \ldots k_n}, \ \text{if } k_1 \notin \{b, e\}.
\end{aligned}
$$

Replacing these values in Eq. 1, we can compute the variation of τ_{X_1} moving object x from cluster C_b^1 to cluster C_e^1 as:

$$
\Delta\tau_{X_1}(\mathcal{T}, x, b, e, k = 1) = \tau_{X_1}(\mathcal{T}) - \tau_{X_1}(\mathcal{S})
$$

$$
= \frac{\Gamma_1 \left[\frac{2M}{T^2}(M + t_{(e)}^{(1)} - t_{(b)}^{(1)}) \right] - \Omega_1 \left[\frac{2}{T} \sum_{k_2, \ldots, k_n} \frac{\mu_{k_2 \ldots k_n}(\mu_{k_2 \ldots k_n} + t_{ek_2 \ldots k_n} - t_{bk_2 \ldots k_n})}{t_{(k_2 \ldots k_n)}^{(2 \ldots n)}} \right]}{\Omega_1^2 - \Omega_1 \left[\frac{2M}{T^2}(M + t_{(e)}^{(1)} - t_{(b)}^{(1)}) \right]}.
$$

where $\Omega_1 = 1 - \sum_{k_1} \frac{\left(t_{(k_1)}^{(1)}\right)^2}{T^2}$ and $\Gamma_1 = 1 - \sum_{k_1, \ldots, k_n} \frac{t_{k_1 \ldots k_n}^2}{T \cdot t_{(k_2 \ldots k_n)}^{(2 \ldots n)}}$ only depend on $\mathcal{T}$ and then can be computed once (before choosing b and e). Thanks to this approach, instead of computing m_i times τ_{X_i} with complexity $O(m_1 \cdot m_2 \cdot \ldots \cdot m_n)$, we compute $\Delta\tau_{X_i}(\mathcal{T}, x, b, e, k = i)$ with a complexity in $O(m_1 \cdot m_2 \cdot \ldots \cdot m_{i-1} \cdot m_{i+1} \cdot \ldots \cdot m_n)$ in the worst case with the discrete partition. Computing Γ_i is in $O(m_1 \cdot m_2 \cdot \ldots \cdot m_n)$, while Ω_i is in $O(m_i)$, and both operations are executed only once for each mode in each iteration.

In a similar way, we can compute the variation of τ_{X_j} for any $j \neq 1$ (this computation is needed only when variants **AVG** and **ALT2** are used):

$$
\Delta\tau_{X_j}(\mathcal{T}, x, b, e, k = 1) = \tau_{X_j}(\mathcal{T}) - \tau_{X_j}(\mathcal{S})
$$

$$
= \frac{1}{\Omega_j T} \sum_{k_2 \ldots k_n} \left(\frac{t_{ek_2 \ldots k_n}^2}{t_{(k_i)_{i \neq j, k_1 = e}}^{(i)_{i \neq j}}} - \frac{(t_{ek_2 \ldots k_n} + \mu_{k_2 \ldots k_n})^2}{t_{(k_i)_{i \neq j, k_1 = e}}^{(i)_{i \neq j}} + \mu_{(k_i)_{i \neq j}}^{(i)_{i \neq j}}} \right.
$$

$$
\left. + \frac{t_{bk_2 \ldots k_n}^2}{t_{(k_i)_{i \neq j}}^{(i)_{i \neq j}}} - \frac{(t_{bk_2 \ldots k_n} - \mu_{k_2 \ldots k_n})^2}{t_{(k_i)_{i \neq j, k_1 = b}}^{(i)_{i \neq j}} - \mu_{(k_i)_{i \neq j}}^{(i)_{i \neq j}}} \right)
$$

where $\Omega_j = 1 - \sum_{k_j} \frac{\left(t_{(k_j)}^{(j)}\right)^2}{T^2}$ only depends on $\mathcal{T}$ and can be computed once for all e. Consequently, instead of computing m_i times τ_{X_j} in Algorithm 1 with a complexity in $O(m_1 \cdot m_2 \cdot \ldots \cdot m_n)$, we compute $\Delta\tau_{X_j}(\mathcal{T}, x, b, e, k = i)$ with a complexity in $O(m_1 \cdot m_2 \cdot \ldots \cdot m_{i-1} \cdot m_{i+1} \cdot \ldots \cdot m_n)$ in the worst case with the discrete partition. Computing Ω_j is in $O(m_j)$ and is done only once for each mode in each iteration.

Similarly, when using variants **AGG** and **AGG2**, instead of calculating $\tau_{(X_j)_{j \neq k}|X_k}$ entirely, we can compute:

$$\Delta\tau_{(X_j)_{j\neq k}|X_k}(\mathcal{T}, x, b, e, k = 1) = \tau_{(X_j)_{j\neq k}|X_k}(\mathcal{T}) - \tau_{(X_j)_{j\neq k}|X_k}(\mathcal{S})$$

$$= \frac{1}{\Omega_{j\neq k}T} \sum_{k_2\ldots k_n} \left(\frac{t^2_{ek_2\ldots k_n}}{t^{(1)}_e} - \frac{(t_{ek_2\ldots k_n} + \mu_{k_2\ldots k_n})^2}{t^{(1)}_e + M} \right.$$

$$\left. + \frac{t^2_{bk_2\ldots k_n}}{t^{(1)}_b} - \frac{(t_{bk_2\ldots k_n} - \mu_{k_2\ldots k_n})^2}{t^{(1)}_b - M} \right)$$

where $\Omega_{j\neq 1} = 1 - \sum_{k_2\ldots k_n} \frac{\left(t^{(j)_{j\neq 1}}_{(k_j)_{j\neq 1}}\right)^2}{T^2}$ only depends on $\mathcal{T}$ and can be computed once for all e. Thus, instead of computing m_i times $\tau_{(X_j)_{j\neq k}|X_k}$ with a complexity in $O(m_1 \cdot m_2 \cdot \ldots \cdot m_n)$, we compute $\Delta\tau_{(X_j)_{j\neq k}|X_k}(\mathcal{T}, x, b, e, k = i)$ with a complexity in $O(m_1 \cdot m_2 \cdot \ldots \cdot m_{i-1} \cdot m_{i+1} \cdot \ldots \cdot m_n)$ in the worst case with the discrete partition. Computing $\Omega_{j\neq i}$ is in $O(m_1 \cdots m_{i-1} \cdot m_{i+1} \cdots m_n)$ and is done only once for each mode in each iteration.

Hence, at each iteration and for each mode k, instead of computing matrix $\mathcal{V} = (\tau_{X_j}(\mathcal{T}^e))_{ej}$ with computational complexity $O((\max_i m_i) \cdot m_1 \cdot m_2 \cdot \ldots \cdot m_n)$ for each τ_{X_j}, we can equivalently compute matrix $\Delta\tau = (\Delta\tau_{X_j}(\mathcal{T}, x, e, k))_{ej}$ with computational complexity $O(m_1 \cdot m_2 \cdot \ldots \cdot m_n)$ for each τ_{X_j}.

Based on the above considerations, for a generic square tensor with n modes, each consisting of m dimensions, the overall complexity is in $O(\mathcal{I}nm^n)$ for strategies **ALT**, **AGG** and **AGG2** and in $O(\mathcal{I}n^2m^n)$ for strategies **AVG** and **ALT2**, where $\mathcal{I}$ is the number of iterations. This difference is due to the fact that the first group of strategies require the computation of just a fixed number of τ's for each mode (one in the **ALT** case, two in the **AGG** and **AGG2** cases), independently of the number of modes n, while **ALT2** and **AVG** require the computation of all the n τ's for each mode. The computational complexity of the two groups of methods differs by a factor of $O(n)$: this could be a discriminant factor in the choice of the method only for tensors with a large number of modes.

5 Experiments

In this section, we report the results of the experiments we conducted to evaluate the performance of our tensor co-clustering algorithm. The section is organized as follows: first, we describe both the synthetic and real-world datasets used in our experiments; second, we compare the different variants of our algorithm by also analyzing their convergence behavior; third, we report the quantitative results of the comparative analysis between our algorithm and some state-of-the-art competitors; finally, we provide some qualitative insights on the co-clusters obtained in one specific case.

5.1 Datasets

The synthetic data we use to assess the quality of the clustering performance are boolean tensors with n modes, created as follows. We fix the dimensions $m_1, \ldots, m_n$ of the tensor

Table 1 Dataset characteristics

Dataset	# Modes	Tensor shape	Main mode	# Classes on main mode
DBLP	3	$6044 \times 4057 \times 20$	authors (4057)	4
MovieLens1	3	$215 \times 181 \times 142$	movies (181)	3
MovieLens2	3	$74 \times 145 \times 115$	movies (145)	3
YelpTOR	3	$628 \times 178 \times 458$	restaurants (628)	3
YelpPHG	3	$237 \times 95 \times 544$	restaurants (237)	3

and the number of embedded clusters $c_1, \ldots, c_n$ on each mode. Then, we first construct a block tensor of dimensions $m_1 \times m_2 \times \cdots \times m_n$ with $c_1 \times c_2 \times \ldots c_n$ blocks. The blocks are created so that there are "perfect" clusters in each mode, i.e., all rows on each mode belonging to the same cluster are identical, while rows in different clusters are different. Then we add noise to the "perfect" tensor, by randomly selecting some element $t_{k_1 \ldots k_n}$, with $k_i \in \{1, \ldots, m_i\}$, for each $i \in \{1, \ldots, n\}$, and changing its value (from 0 to 1 or vice versa). The amount of noise is controlled by a parameter $\epsilon \in [0, 1]$, indicating the fraction of elements of the original tensor we change. We generate tensors of different number of modes, size, number of clusters and value of noise ($\epsilon = 0.05$ to 0.3 with a step of 0.05).

We also apply the algorithms to three real-world datasets (see Table 1). The first dataset is the "four-area" DBLP dataset[1]. It is a bibliographic information network extracted from DBLP data, downloaded in the year 2008. The dataset includes all papers published in twenty representative conferences of four research areas (database, data mining, machine learning and information retrieval), five in each area. Each element of the dataset corresponds to a paper and contains the following information: authors, venue and terms in the title. The original dataset contains 14376 papers, 14475 authors and 13571 terms. Part of the authors (4057) are labeled in four classes, roughly corresponding to the four research areas. We select only these authors and their papers and perform stemming and stop-words removal on the terms by using the functions provided by the NLTK Python library[2] (in particular, we use the Porter stemmer). We obtain a dataset with 14328 papers, from which we create a ($6044 \times 4057 \times 20$)-dimensional tensor, highly sparse (99.98% of entries are equal to zero); the generic entry t_{ijk} of the tensor counts the number of times term i was used by author j in conference k.

The second dataset is the "hetrec2011-movielens-2k" dataset[3] published by Cantador et al. (2011). It is an extension of MovieLens10M dataset, published by GroupLeans research group[4]. It links the movies of MovieLens dataset with their corresponding web pages at the Internet Movie Database (IMDb[5]) and the Rotten Tomatoes movie review systems[6]. From the original dataset, only those users with both rating and tagging information are retained, for a total of 2113 users, 10197 movies (classified in 20 overlapping genres) and 13222 tags. Then, we select the users that have tagged at least two different movies, the

---

tags that have been used for at least ten different movies, and the movies that received a tag from at least five different users. Finally, starting from the remaining data, we create two different tensors:

– *MovieLens1*: it includes all the movies classified as 'Animation', 'Documentary', or 'Horror'. Since we need unique labels to assess the quality of our co-clustering algorithm, we keep only the movies with a unique genre label. At the end we obtain a $(215 \times 181 \times 142)$-dimensional tensor. The class on the movie mode are quite imbalanced: there are only 11 documentaries, while the remaining movies are divided into Animation (63) and Horror (117).
– *MovieLens2*: it includes all the movies (uniquely) classified as 'Adventure'(33), 'Comedy'(10), or 'Drama'(102). The final tensor has dimensions $(74 \times 145 \times 115)$.

The last dataset is a subset of the Yelp dataset[7]. It is a subset of Yelp's businesses, reviews, and user data. Among all available data, we select only the reviews about Italian, Mexican and Chinese restaurants with at least ten reviews and the users who write at least five reviews. Finally, we pre-process the text of the remaining reviews by performing both stemming and stop-word removal and by retaining the words appearing at least 5 times in at least one category of restaurants, plus the 150 most frequent words (regardless of the category). At the end, we obtain two different tensors, with restaurants on the first mode, users on the second mode, and words used in the reviews on the third mode:

– *yelpTOR*: it includes the restaurants of the city of Toronto. The final tensor has shape (626, 178, 458) and contains 1885 reviews about 234 Italian restaurants, 288 Chinese restaurants and 104 Mexican restaurants. We consider the type of restaurant (Italian, Chinese or Mexican), as the labels on the first mode.
– *yelpPGH*: it includes the restaurants of the city of Pittsburgh. The final tensor has shape (237, 95, 544), containing data about 104 Italian restaurants, 64 Chinese and 63 Mexican restaurants.

5.2 Comparison of the different variants of τTCC

In this section we apply the five different versions of τTCC (**ALT**, **AVG**, **ALT2**, **AGG**, **AGG2**) to synthetic and real-world data with the aim of comparing their overall performances and convergence behavior. We first apply the algorithms on synthetic data, varying the number of modes and the shape of the tensor ($100 \times 100 \times 100$, $1000 \times 100 \times 20$, $100 \times 100 \times 100 \times 100$, and $1000 \times 100 \times 20 \times 20$) and the number of embedded clusters on each mode ((5,5,5), (5,3,2), (10,5,3), and (10,5,3,2)), with a medium level of noise of 0.15. As shown in Fig. 3, the **AVG** variant of τTCC provides less accurate results than the other variants: in cubic tensors (tensors with the same dimensionality on all modes) all the methods achieve similar levels of $avg(\tau)$, but **AVG** requires more iterations. On asymmetric tensors, **AVG** ends in a solution with lower average τ compared with the one obtained by the other methods. The algorithms have a similar behavior on real-world tensors (Fig. 4): the one with the overall best results in terms of $avg(\tau)$ is **ALT2**, followed by **AGG2** and **ALT**. It is worth noting that the $avg(\tau)$ grows even with variants that do
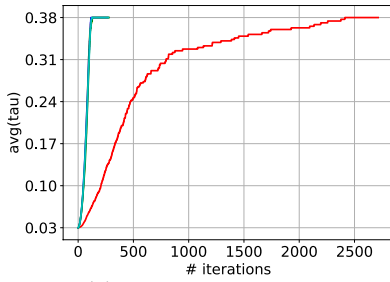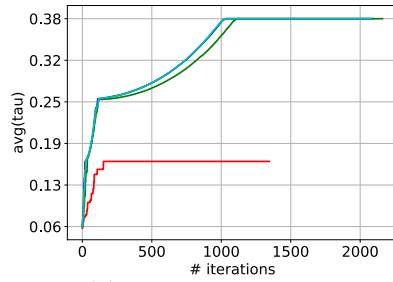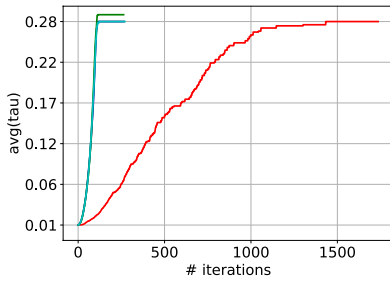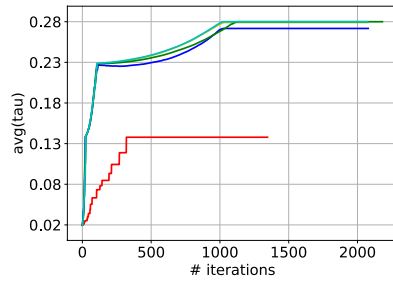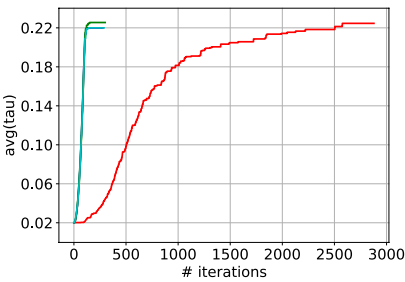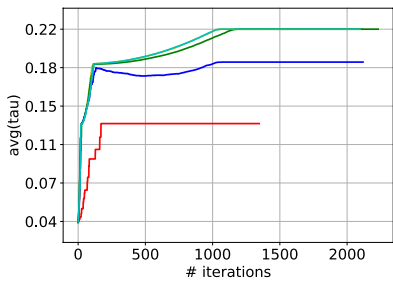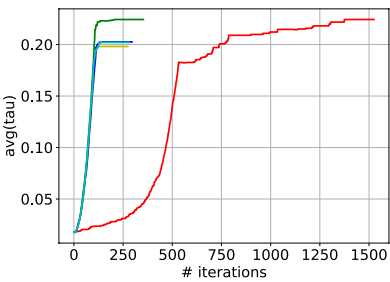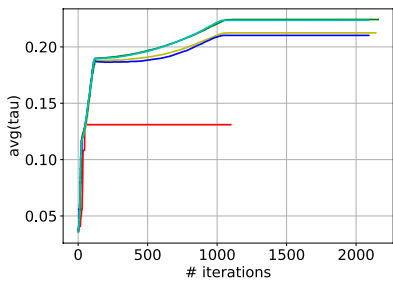
Fig. 3 $Avg(\tau)$ per iteration, for all the τTCC variants, on synthetic data, varying the shape of the tensor and the number of embedded co-clusters
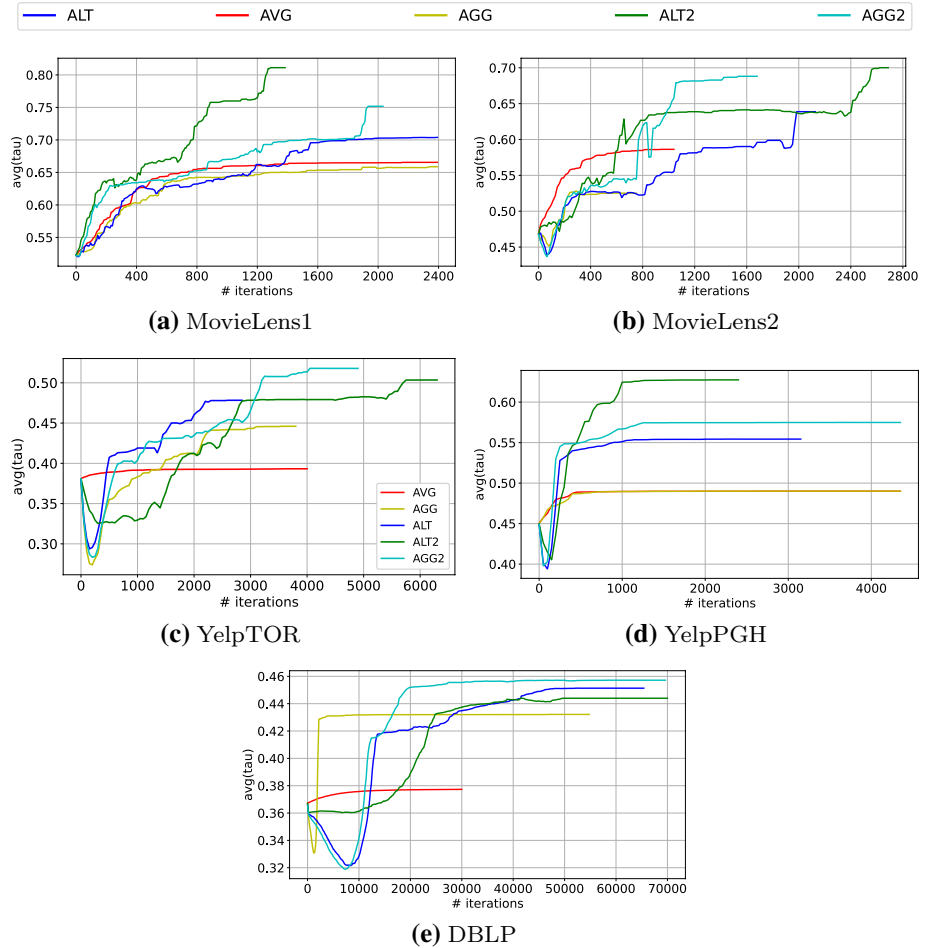
Fig. 4 $Avg(\tau)$ per iteration, for all the τTCC variants, on MovieLens, Yelp and DBLP datasets

not optimize it directly. In real-world data, although during the very first iterations $avg(\tau)$ decreases, it begins to grow monotonically, with the exception of some small low peaks. Of course, **AVG** variant is not affected by this behavior, since it optimizes $avg(\tau)$ directly. Moreover, as anticipated in Sect. 4.2, the direct optimization of $avg(\tau)$ often results in a relatively poor local optimum. This is because a relaxed constraint on the neighborhood search allows the algorithm to explore more solution subspaces, thus ending up with a better final objective function value.

After these preliminary experiments we conclude that **ALT2** seems to be the most effective method, with **ALT2**, **AGG2** and **ALT** outperforming the other two variants of algorithm τTCC. We also conduct a Friedman statistical test followed by a Nemenyi post-hoc test (Demsar 2006) in order to assess whether the differences among the best three variants are statistically significant. At confidence level $\alpha = 0.01$, the null hypothesis of the Friedman test (stating that the differences are not statistically significant) can be rejected for $avg(\tau)$ values; we then proceed with the post-hoc Nemenyi test. The results show that the

differences between the average rank of **ALT2** and those of the other methods are more than the critical difference $CD = 0.19168$ at confidence level $\alpha = 0.01$. Consequently the null hypothesis of the Nemenyi test passed, and we can conclude that **ALT2** is statistically better than **AGG2** and **ALT**. Nevertheless, hereinafter, we will consider all the three best performing variants in our experiments, while we will not report the results for **AVG** and **AGG**.

5.3 τTCC against state-of-the-art competitors

In this section, we compare our results with those of other state-of-the-art tensor co-clustering algorithms, mainly based on CP (Harshman 1970) and Tucker (Tucker 1966) decomposition. Additionally, we include another very recent approach based on the latent block model. Hence, we consider the following algorithms:

– *nnCP*. It is the non-negative CP decomposition and can be used to co-cluster a tensor, as done by Zhou et al. (2009), by assigning each element in each mode to the cluster corresponding to the latent factor with highest value. The algorithm requires as input the number r of latent factors: we set $r = \max_{j=1,\ldots n}(c_j)$, where c_j is the true number of classes on the j-th mode of the tensor. Since the CP model represents the tensor as the sum of r rank-1 decompositions, the number r of latent factors is the same on all modes. However, the rank r of the decomposition represents the maximum number of clusters that can be found on each mode of the tensor, thus the fact that we specify the same number r of latent factors on all the modes does not force the algorithm to identify exactly r clusters on each mode. This is particularly important when the number of embedded clusters c_j differs along the modes, because the algorithm is allowed to identify a number of clusters that is less then the maximum number r.
– *nnCP+kmeans*. It combines CP with a post-processing phase in which k-means is applied on each of the latent factor matrices. Here, we set the rank r to $r = \max_{j=1,\ldots n}(c_j) + 1$ and the number k_i of clusters in each dimension equal to the real number of classes (according to our experiments, this is the choice that maximizes the performances of this algorithm).
– *nnTucker*. It is the non-negative Tucker decomposition. Here we set the ranks of the core tensor equal to $(c_1, \ldots, c_n)$.
– *nnT+kmeans*. It combines Tucker decomposition with k-means on the latent factor matrices, similarly as what has been done by Huang et al. (2008) and Cao et al. (2015).
– *SparseCP*. It consists of a CP decomposition with non-negative sparse latent factors (Papalexakis et al. 2013). We set the rank r of the decomposition equal to the maximum number of classes on the n modes of the tensor. It also requires one parameter λ_i for each mode of the tensor: for the choice of their values we follow the instructions suggested in the original paper.
– *TBM*. It performs tensor co-clustering via the Tensor Block Model (Wang and Zeng 2019). As parameters, it requires the number of clusters on each mode and a penalty

coefficient λ; the number of clusters is set equal to the correct number of classes, while λ is tuned as suggested in the original paper.

The available codes of **SparseCP** and **TBM** only work with 3-way tensors, so we have to exclude these methods when we perform experiments on tensors with more than three modes.

To assess the quality of the clustering performances, we consider two measures commonly used in the clustering literature: normalized mutual information (NMI) (Strehl and Ghosh 2002) and adjusted rand index (ARI) (Hubert and Arabie 1985).

All experiments are performed on a server with 32 2.1GHz Intel Xeon Skylake cores, 256GB RAM, Ubuntu 20.04.02 LTS (kernel release: 5.8.0)[8]. In the following, we first present the comparative results obtained on synthetic data, then we report the performances achieved by our algorithms and the competitors on real-world data.

5.3.1 Results on synthetic data

We test the performances of τTCC against those of its competitors on synthetic data with embedded block co-clusters, constructed as described in Sect. 5.1. We consider tensors with 3, 4 and 5 modes, with different shapes ($100 \times 100 \times 20, 100 \times 100 \times 100, 1000 \times 100 \times 20$ and $1000 \times 500 \times 20$ for 3-way tensors, $100 \times 100 \times 100 \times 100$ and $1000 \times 100 \times 20 \times 20$ for 4-way tensors and $100 \times 100 \times 100 \times 100 \times 100$ and $1000 \times 100 \times 20 \times 20 \times 20$ for 5-way tensors), different numbers and shapes of block co-clusters (combinations of 2,3,5 and 10 clusters on each mode) and with three levels of noise (0.1, 0.2 and 0.3), for a total of 276 tensors. We run all the experiments ten times and compute the average NMI and ARI. Figs. 5, 6, 7 and 8 report the results in terms of average NMI of all the experiments. The results in terms of mean ARI are similar and are presented in the appendix (see Figs. 13, 14, 15 and 16). In these figures we include only the best variant of the algorithm (referred to as τTCC$_{ALT2}$), according to our previous analysis, for sake of clarity. We also omit to show the standard deviation of the experiments in the plots. However, the results are very stable: the standard deviation of τTCC ranges from 0 to 0.001, while the algorithms with the highest variability are *nnCP+kmeans* and *nnT+kmeans*, whose standard deviation ranges from 0 to 0.004. In all the experiments our algorithm achieves quite "perfect" levels of NMI and ARI (always greater than 0.93), meaning that it is able to identify the correct co-clusters embedded in the tensors. The shape and the number of modes of the tensor and the asymmetry in the number of clusters on the different modes do not affect significantly the quality of the co-clustering. Furthermore, τTCC$_{ALT2}$ consistently outperforms *SparseCP*, *TBM*, *nnTucker* and *nnCP* on synthetic data. Finally, the results of τTCC$_{ALT2}$ are comparable with those of *nnCP+kmeans* and *nnT+kmeans*: only when the number of clusters on the three modes is different, τTCC$_{ALT2}$'s results are slightly lower than those of the *kmeans*-based algorithms. This is due to the fact that our algorithm fails in identifying the correct number of clusters in these scenarios: for instance, when the clusters on the three modes are 10, 5 and 3 respectively, τTCC$_{ALT2}$ identifies 9, 5 and 3 clusters. We don't have the same issue with *k*-means, for which, however, the correct number of clusters is given as input.

---

[8] The source code of our algorithm and all data used in this paper are available at: https://github.com/elena battaglia/tensor_cc
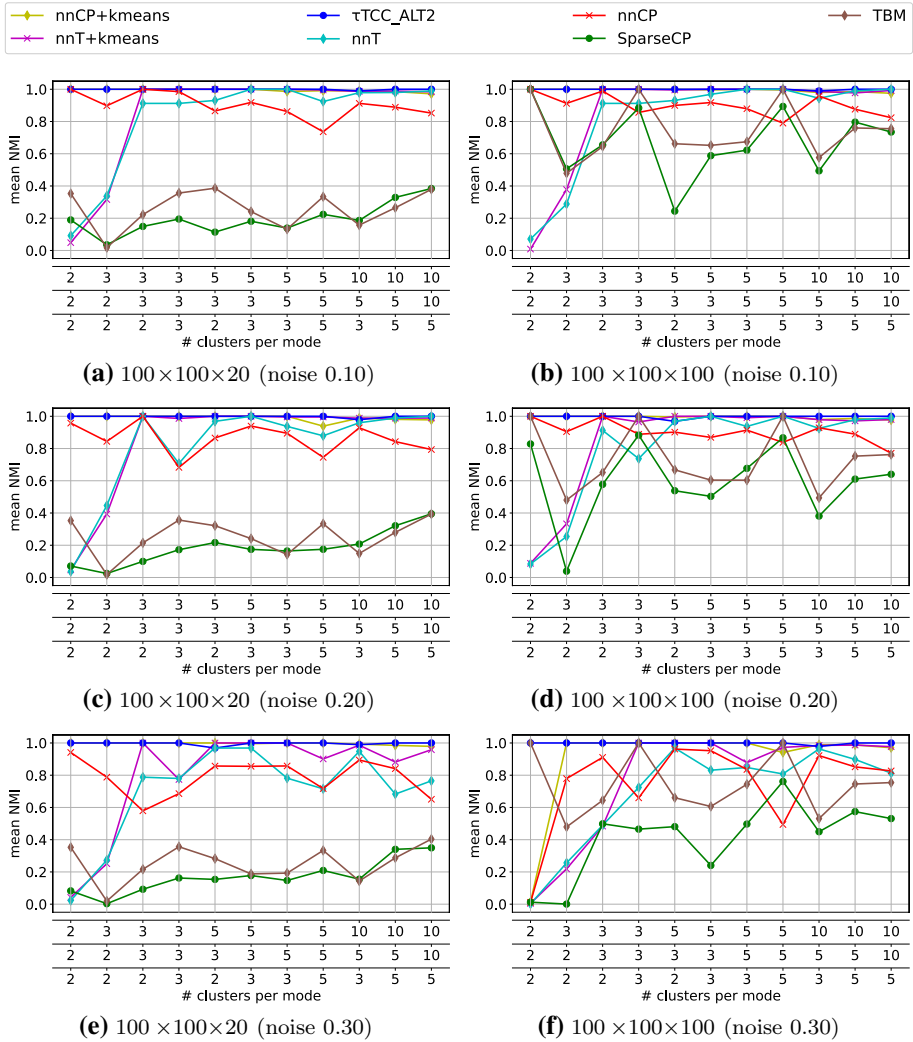
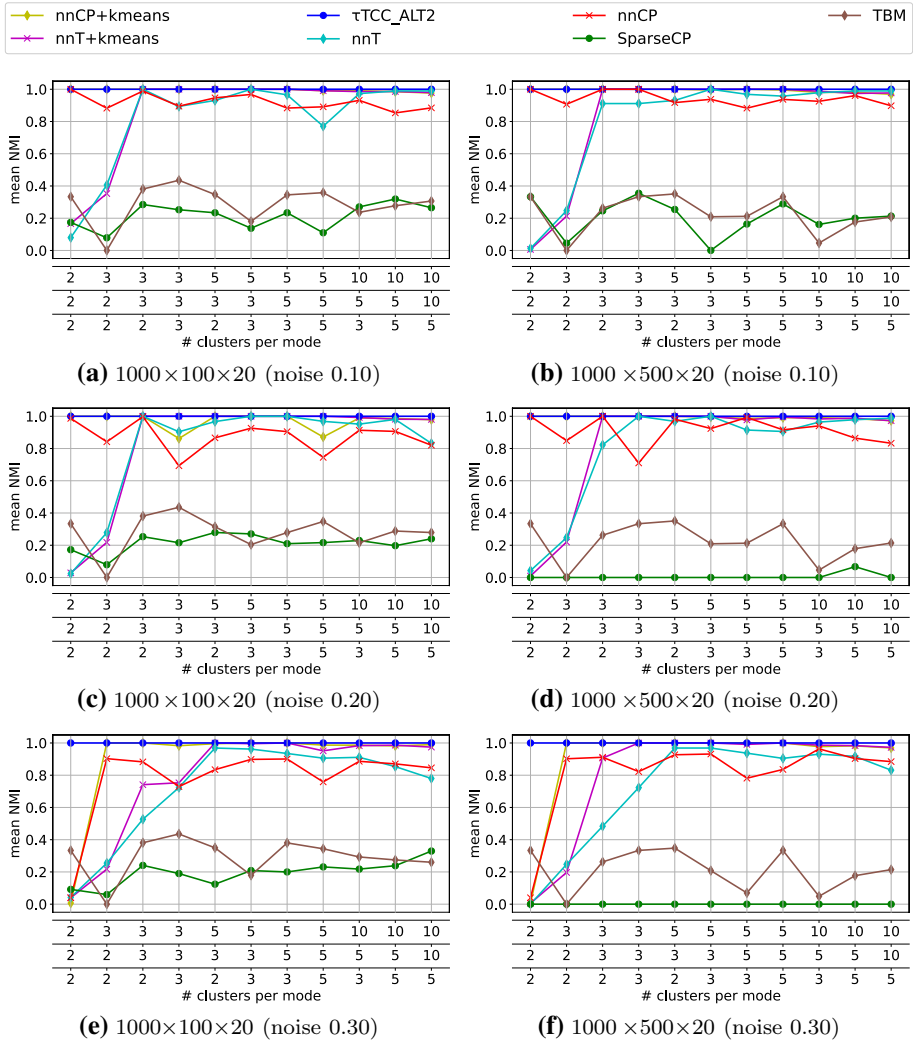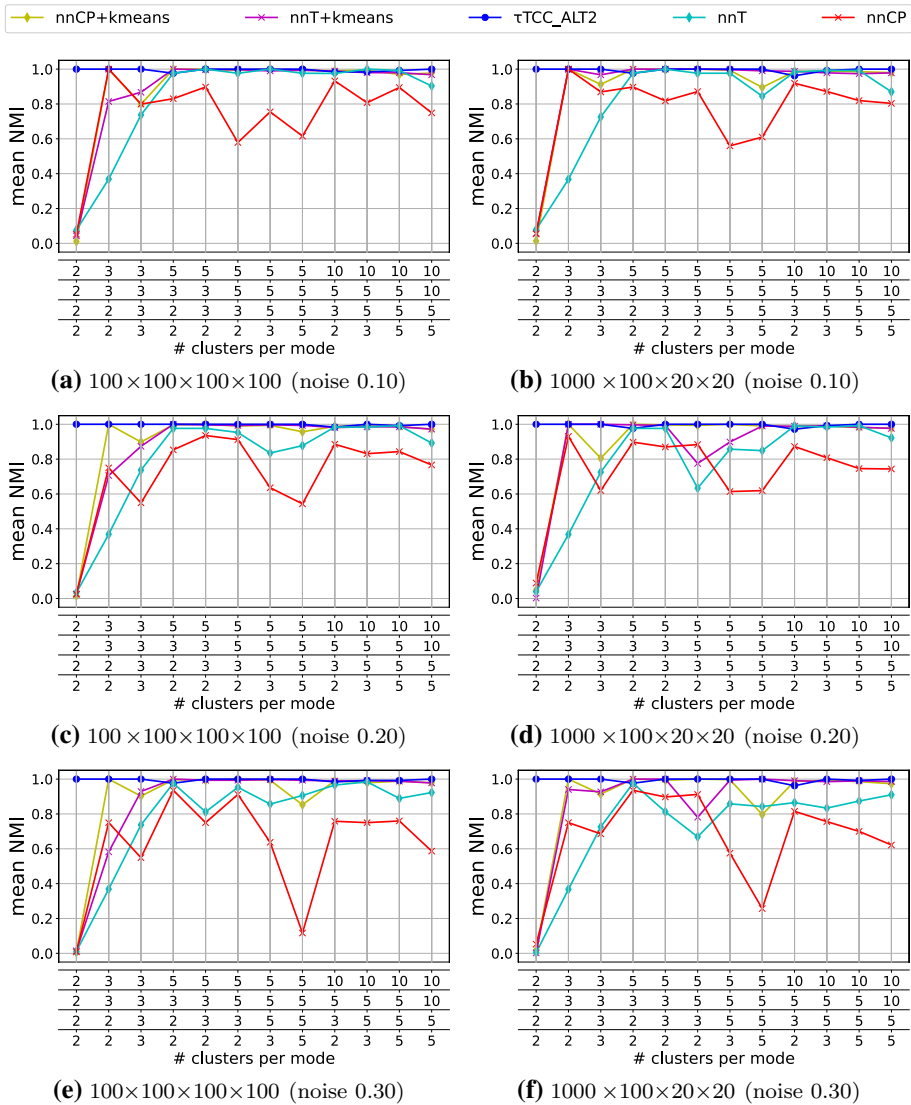**Fig. 5** Mean NMI on the three modes varying the number of embedded clusters on synthetic 3-way tensors with different sizes and levels of noise

To further investigate this behavior, we analyze the results of the three variants of τTCC on synthetic data (the detailed results are reported in the appendix, in Figs. 17, 18, 19 and 20). We find that, while all the variants of our algorithm find the correct clusters when the number of embedded clusters on all the modes are similar, the results degrade when we consider different numbers of clusters across the modes. This issue is more pronounced for ALT and AGG2, while ALT2 is able to find good or perfect clusters even in these scenarios (in particular, when $m_i >> k_i$ for all $i = 1, 2, 3$, where m_i is the dimension of the tensor and k_i the number of clusters on mode i).
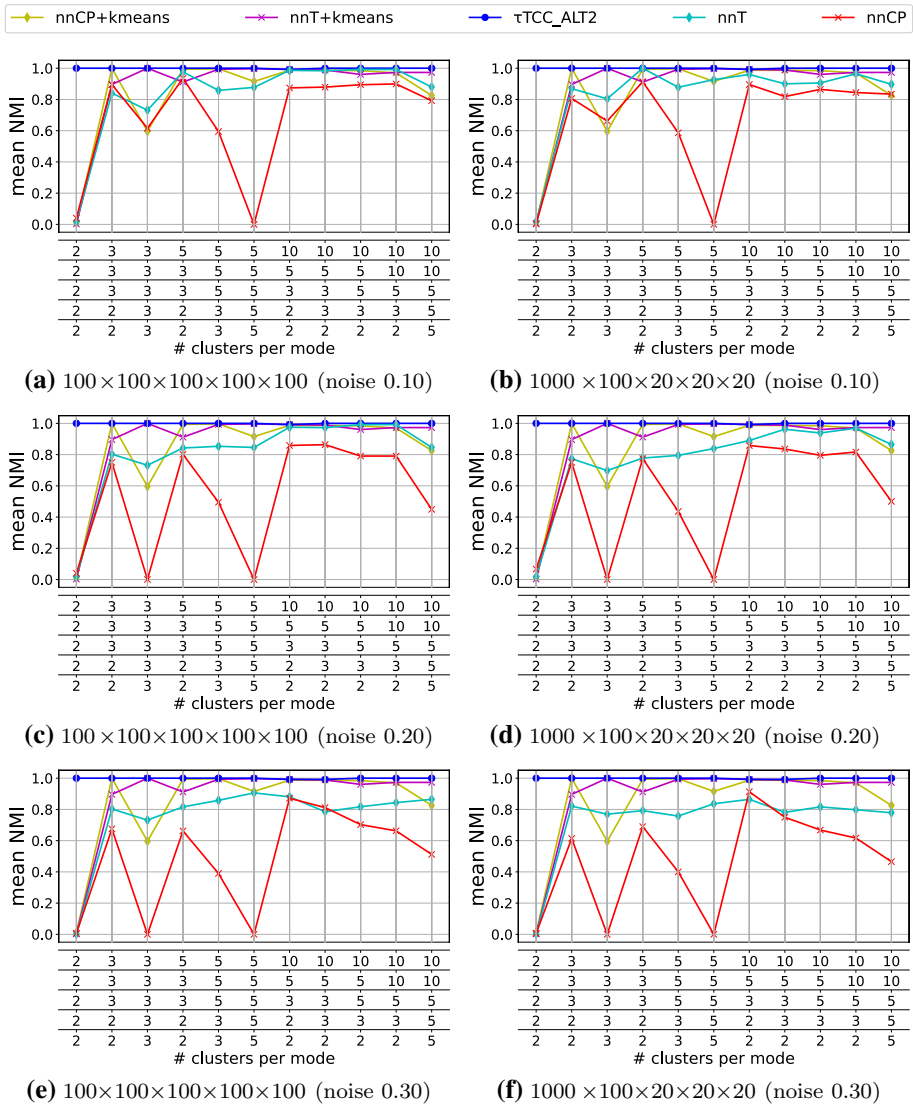
Fig. 6 Mean NMI on the three modes varying the number of embedded clusters on synthetic 3-way tensors with different sizes and levels of noise

5.3.2 Execution time analysis

In this section, we show the execution times of τTCC on tensors with different number of modes and shape. We compare the execution times of τTCC with those of its competitors; for sake of clarity, we exclude from the experiment *nnT+kmeans* and *nnCP+kmeans*, since the execution time of the post-processing K-means algorithm is negligible w.r.t. the execution time of the decomposition. Firstly, we run the different algorithms on 3-way tensors of increasing magnitude, starting from a tensor of shape $100 \times 100 \times 10$ and adding from 100 to 900 dimensions to the first mode, until reaching a tensor of shape $1000 \times 100 \times 10$. Then

Fig. 7 Mean NMI on the four modes varying the number of embedded clusters on synthetic 4-way tensors with different sizes and levels of noise

we add from 100 to 900 dimensions to the second mode, until we reach a tensor of shape $1000 \times 1000 \times 10$. All the tensors have 5 clusters on the larger modes and 2 on the smallest ones. Figure 9a shows that all the variants of τTCC are slower than their competitors (with the exception of *SparseCP*), approximately by a factor of 10. This depends on the fact that the number of iterations until convergence is higher for τTCC than for the other methods. However, the trend of the curves are similar for all methods, as expected by looking at the theoretical complexities reported in Table 2. In the second and third experiments (Figs.9b and 9c) we start with the same 3-way tensor of shape $100 \times 100 \times 10$ and then we increase

Fig. 8 Mean NMI on the four modes varying the number of embedded clusters on synthetic 4-way tensors with different sizes and levels of noise

Table 2 Computational complexities for (hyper)cubic tensors with n modes and dimensionality m on each mode. Where needed, the rank of the decomposition is r on each mode. $\mathcal{I}$ is the number of iterations

Algorithm	Complexity
τTCC_ALT, τTCC_AGG2	$\mathcal{I}nm^n$
τTCC_ALT2	$\mathcal{I}n^2m^n$
nnCP, SparseCP	$\mathcal{I}nrm^n$
nnT, TBM	$\mathcal{I}rm^n$

(a) 3-way tensors

(b) n-way tensors
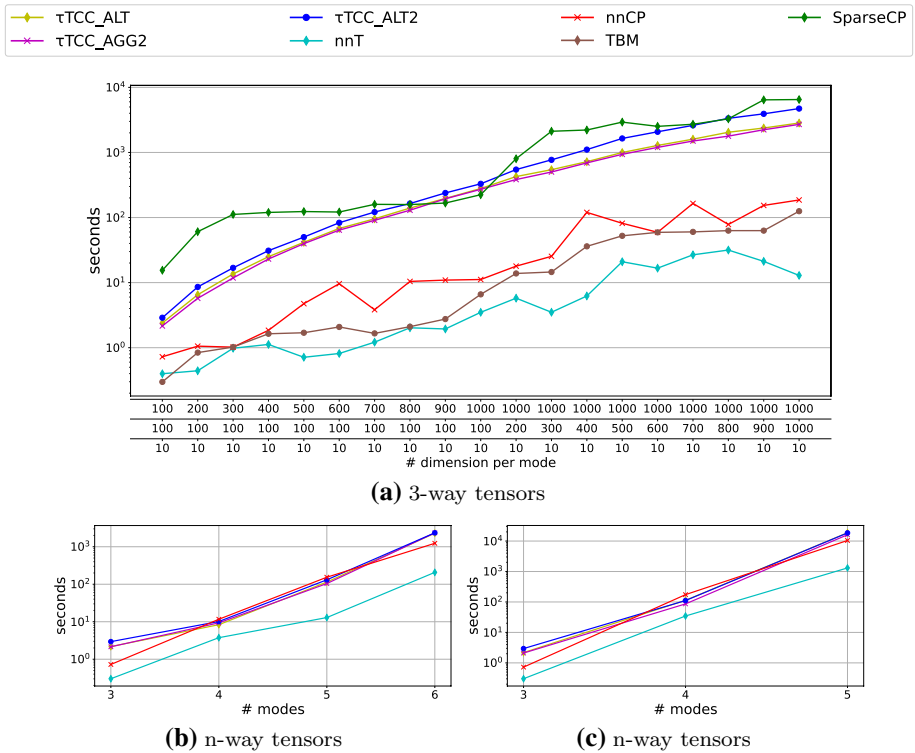
(c) n-way tensors

Fig. 9 Execution time in seconds of τTCC and its competitors on 3-way tensors with different shapes

Table 3 Results achieved by the co-clustering algorithms on the real-world datasets, in terms of NMI. NMI is computed for the main mode (authors in DBLP, movies in MovieLens and restaurants in Yelp)

Algorithm	NMI				
	DBLP	MovieLens1	MovieLens2	yelpTOR	yelpPGH
τTCC$_{ALT2}$	0.706 ±0.003	**0.659±0.018**	**0.403 ± 0.063**	0.381 ± 0.031	**0.373 ± 0.009**
τTCC$_{ALT}$	0.715 ±0.001	0.579 ± 0.121	0.263 ± 0.099	0.364 ± 0.038	0.352 ± 0.019
τTCC$_{AGG2}$	0.749 ±0.006	0.516 ± 0,017	0.382 ± 0.073	**0.433 ± 0.023**	0.339 ± 0.011
nnTucker	**0.782 ± 0.000**	0.421± 0.000	0.242 ± 0.000	**0.429 ± 0.000**	0.275 ± 0.000
nnCP	0.742 ±0.000	0.380 ± 0.000	0.111 ± 0.000	0.425 ± 0.000	0.094 ± 0.000
SparseCP	0.000 ±0.000	0.050± 0.000	0.066 ± 0.000	0.200± 0.000	0.108 ± 0.000
TBM	-	0.305 ± 0.000	0.097 ± 0.000	0.014 ± 0.000	0.107 ± 0.000
nnCP+kmeans	0.238 ±0.001	0.307±0.001	0.127 ± 0.061	0.100 ± 0.019	0.081 ± 0.016
nnT+kmeans	0.246 ±0.003	0.376 ±0.002	0.223 ± 0.012	0.088 ± 0.025	0.085 ± 0.028

The best results are highlighted in bold

the number of modes. More in detail, in the experiment reported in Fig. 9b a new mode of dimension 10 is added at each time, while in the experiment reported in Fig. 9c, at each step, we add a new mode of dimensionality 100. The plots show that the difference in the execution time between τTCC and the other methods (in particular *nnCP*) decreases with

Table 4 Results achieved by the co-clustering algorithms on the real-world datasets, in terms of ARI. ARI is computed for the main mode (authors in DBLP, movies in MovieLens and restaurants in Yelp)

Algorithm	ARI				
	DBLP	MovieLens1	MovieLens2	yelpTOR	yelpPGH
τTCC_{ALT2}	0.728 ±0.003	**0.707±0.032**	**0.506 ± 0.106**	0.390 ± 0.038	**0.272 ± 0.033**
τTCC_{ALT}	0.730 ±0.003	0.539 ± 0.239	0.306 ± 0.108	0.284 ± 0.061	0.195 ± 0.019
τTCC_{AGG2}	0.796 ±0.008	0.339 ± 0.037	0.434 ± 0.145	**0.458 ± 0.026**	0.200 ± 0.023
nnTucker	**0.838 ± 0.000**	0.538± 0.000	0.17 ± 0.000	0.386± 0.000	0.184 ± 0.000
nnCP	0.804 ±0.000	0.338± 0.000	0.032 ± 0.000	0.367 ± 0.000	0.058 ± 0.000
SparseCP	0.000 ±0.000	0.001± 0.000	0.001 ± 0.000	0.155 ± 0.000	0.041 ± 0.000
TBM	-	0.111± 0.000	0.001 ± 0.000	0.004 ± 0.000	0.035 ± 0.000
nnCP+kmeans	0.078 ±0.001	0.206 ±0.001	0.054 ± 0.044	0.001 ± 0.000	0.033 ± 0.012
nnT+kmeans	0.061 ±0.001	0.309 ±0.001	0.087 ± 0.016	0.006 ± 0.007	0.029 ± 0.013

The best results are highlighted in bold

(a) DBLP   (b) YelpTOR
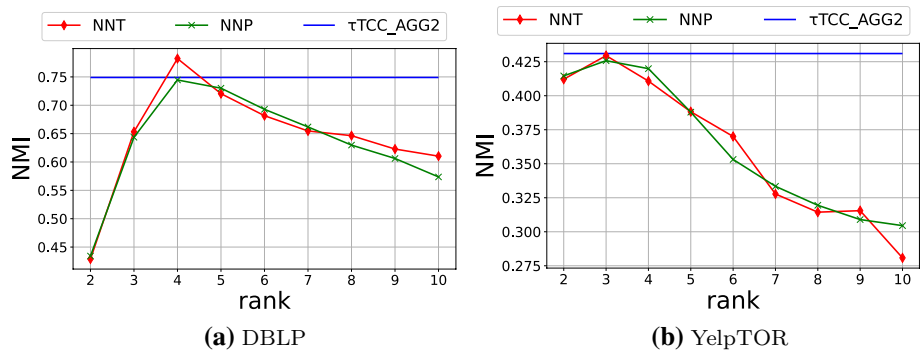
Fig. 10 Variation of nnTucker/nnCP results w.r.t. the rank of the decomposition in DBLP and YelpTOR datasets

the number of modes. We can then conclude that the number of modes affect the execution times of all algorithms to a similar extent.

5.3.3 Results on real-world data

As last experiment, we apply our algorithm and its competitors on real-world datasets. Each algorithm is applied ten times on every dataset and the average results and standard deviations are presented in Tables 3, 4. Algorithms *nnCP*, *nnTucker* and their variant with *k*-means are applied with different parameters: we try different ranks of the decomposition (while *k* of *k*-means is fixed to the correct number of classes in the data) and we report the best result obtained. In this way we are giving a big advantage to our competitors: we choose the rank of the decomposition and the number of clusters by looking at the actual number of categories, which are unknown in standard unsupervised settings. Despite this, *τTCC* (in all its variants) outperforms the other algorithms on all datasets but one (DBLP) and has comparable results on another (YelpTOR). As regards DBLP, non-negative Tucker decomposition (with the number of latent factors set to the correct number of embedded clusters) achieves the best results. Non-negative CP decomposition obtains results that are

(a) Cluster1 - Cartoons

(b) Cluster2 - Wallace&Gromit

(c) Cluster3 - Anime

(d) Cluster4 - Horror

(e) Cluster5 - Documentaries

Fig. 11 First 30 movie in each cluster identified by τTCC on dataset MoveiLens1

comparable with those of τTCC. However, as said before, by fixing the number of latent factors equal to the real number of natural clusters we are facilitating our competitors; when we modify, even if only slightly, the number of latent factors (see Fig.10), the results get immediately worse than those of τTCC. A similar observation holds for YelpTOR: Tucker decomposition achieves the best performances (just in terms of NMI, indeed) only when the number of latent factors equals the number of naturally embedded clusters.

The number of clusters identified by τTCC is usually close to the correct number of embedded clusters: on average, 5 instead of 4 for DBLP, 5 instead of 3 for MovieLens1, the correct number 3 for MovieLens2, 5 instead of 3 for YelpPGH. Only YelpTOR presents a number of clusters (13) that is far from the correct number of classes (3). However, more than the 85% of the objects are classified in 3 large clusters, while the remaining objects form very small clusters: we consider these objects as candidate outliers. The same behavior is even more pronounced in DBLP, where four clusters contain the 99.9% of the objects and only 2 objects stay in the "extra cluster".

5.4 Qualitative evaluation of the results

Here, we provide some insights about the quality of the clusters identified by our algorithm. To this purpose, we choose a co-clustering of the MovieLens1 dataset, obtained with selection strategy **AGG2**. The results obtained by the other variants of τTCC, however, are very similar both in the number and in the composition of the identified clusters.

When Algorithm 1 terminates, five clusters of movies are identified, instead of the three categories (Animation, Horror and Documentary) we consider as labels. The tag clouds in Fig. 11, illustrate the 30 movies with more tags for each cluster (text size depends on the actual number of tags): it can be easily observed that the first cluster concerns animated movies for children, mainly Disney and Pixar movies; the second one is a little cluster containing animated movies realized with the claymation technique (mainly Wallace and Gromit saga's movies or other films by the same director); the third cluster is still a subset of the animated movies, but it contains anime and animated films from Japan. The fourth cluster is composed mainly by horror movies and the last one contains only documentaries. On the tag mode, our algorithm finds thirteen clusters. Six of them contain more than 90% of the total tags and only 10 uninformative tags are partitioned in other 7 very
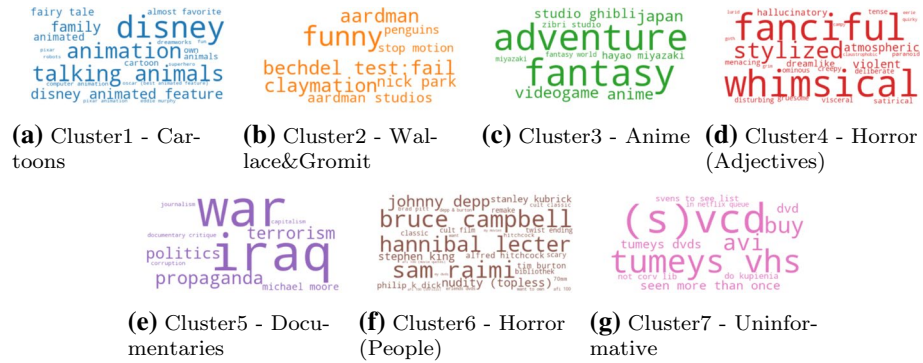
(a) Cluster1 - Cartoons

(b) Cluster2 - Wallace&Gromit

(c) Cluster3 - Anime

(d) Cluster4 - Horror (Adjectives)

(e) Cluster5 - Documentaries

(f) Cluster6 - Horror (People)

(g) Cluster7 - Uninformative

Fig. 12 First 20 tags in each cluster identified by τTCC on dataset MoveiLens1. The last cluster is the union of 7 little clusters of few tags each

Table 5 Comparison of the results obtained on MovieLens1 dataset with 3D-τTCC (on movie-user-tag tensor), 2D-τTCC$_{T1}$ (on movie-user matrix), 2D-τTCC$_{T2}$ (on movie-tag matrix) and *CoStar* on both T1 and T2 matrices simultaneously

Algorithm	NMI	ARI
3D-τTCC	**0.66 ± 0.02**	**0.71 ± 0.01**
2D-τTCC$_{T1}$	0.47 ±0.01	0.29 ±0.01
2D-τTCC$_{T2}$	0.36 ±0.02	0.27 ±0.01
CoStar	0.56 ±0.02	0.48 ±0.02

The best results are highlighted in bold

small clusters, and could be considered as outliers. There is a one-to-one correspondence between four clusters of movies (Cartoons, Anime, Wallace&Gromit and Documentary) and four of the tag clusters; cluster Horror, instead, can be put in relation with two different tag clusters, the first containing names of directors, actors or characters of popular horror movies, the second composed by adjectives typically used to describe disturbing films. For more details, see Fig. 12.

In a few cases, the cluster group of a movie does not coincide with the category label: for instance, Tim Burton's movies *The Nightmare Before Christmas* and *Corpse Bride*, which are labeled as "Animation" in the original dataset, have been included in the horror cluster by τTCC algorithm. These movies, indeed, have more similarities with non-animated horror movies than with cartoons for children, and our co-clustering algorithm was able to capture that (even if they have been also given tags as "animated" and "claymation" that are typical of the first two clusters). This is probably due to the fact that τTCC takes advantage of the tensor structure of the data, having the opportunity to look at both the tag and user modes when partitioning the movies: besides being tagged with the same words, similar films are also appreciated by the same kind of users. Unfortunately, we do not have any latent class information about the users.

To better understand how much the tensor structure helps to find better clusters on the main mode, we execute a further experiment: we try τTCC on two 2-way tensors, *T*1 having movies and users as modes, and *T*2 with movies and tags as modes. Each cell of the matrix counts the number of times a movie has been tagged by a particular user (in *T*1) and the number of times a movie received a particular tag (in *T*2).

We apply τTCC algorithm on the two matrices independently and, finally, on the two matrices simultaneously, using the 2-way co-clustering algorithm for multi-view data based on the optimization of τ (CoStar) proposed by Ienco et al. (2013). The results are summarized in Table 5: they clearly show that the quality of the results (in terms of both NMI and ARI) is higher for the 3-way version of the algorithm than for the 2-way versions. Considering multiple views helps, but not to a great extent, indeed. These results suggest that movie clustering benefits from the tensorial structure of the data, drawing information not only from the movie-user or movie-tag relationships but also from the user-tag relationship.
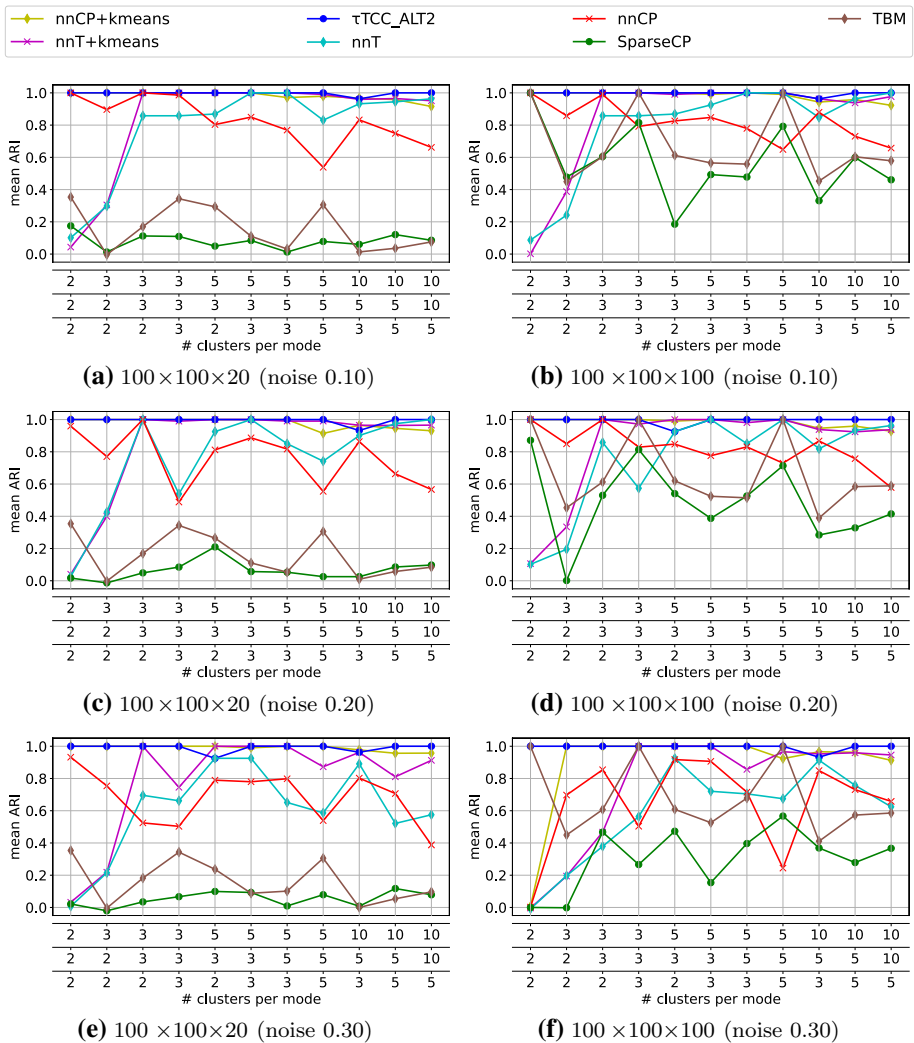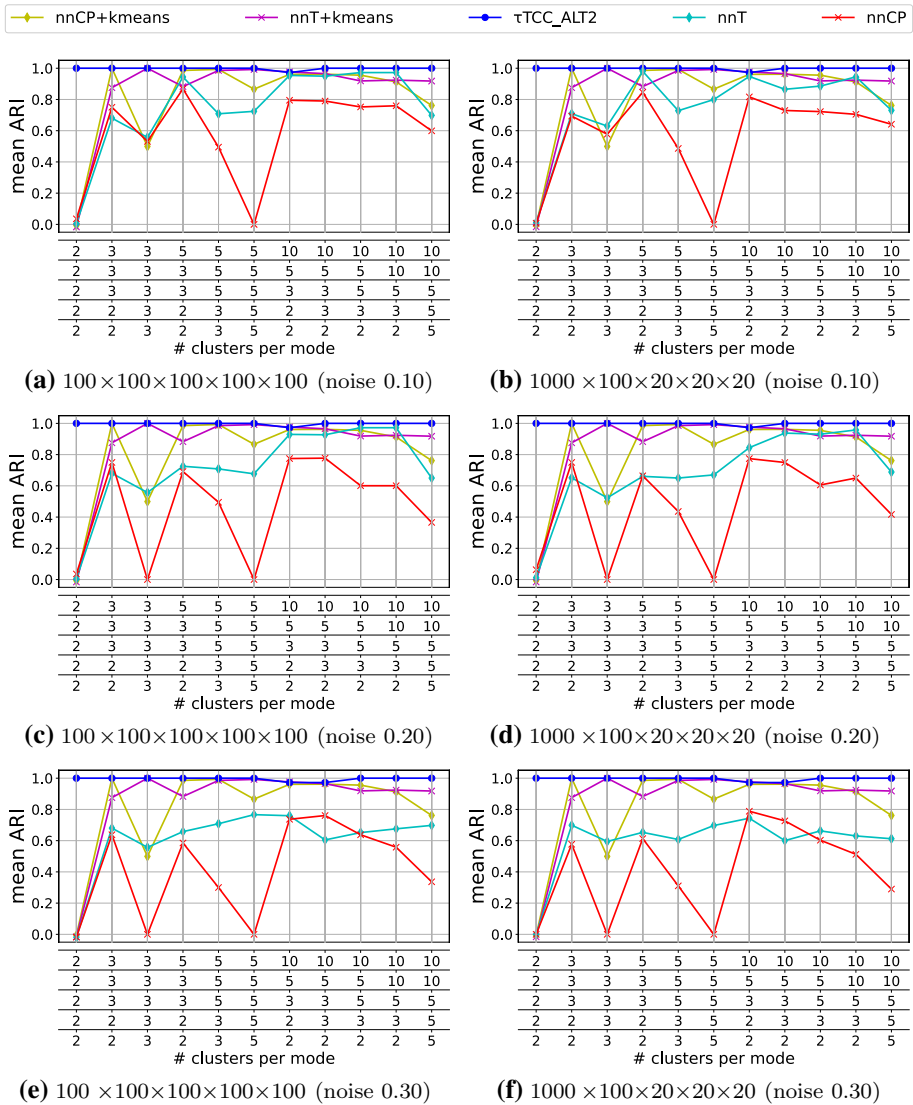

Fig. 13 Mean ARI on the three modes varying the number of embedded clusters on synthetic 3-way tensors with different sizes and levels of noise
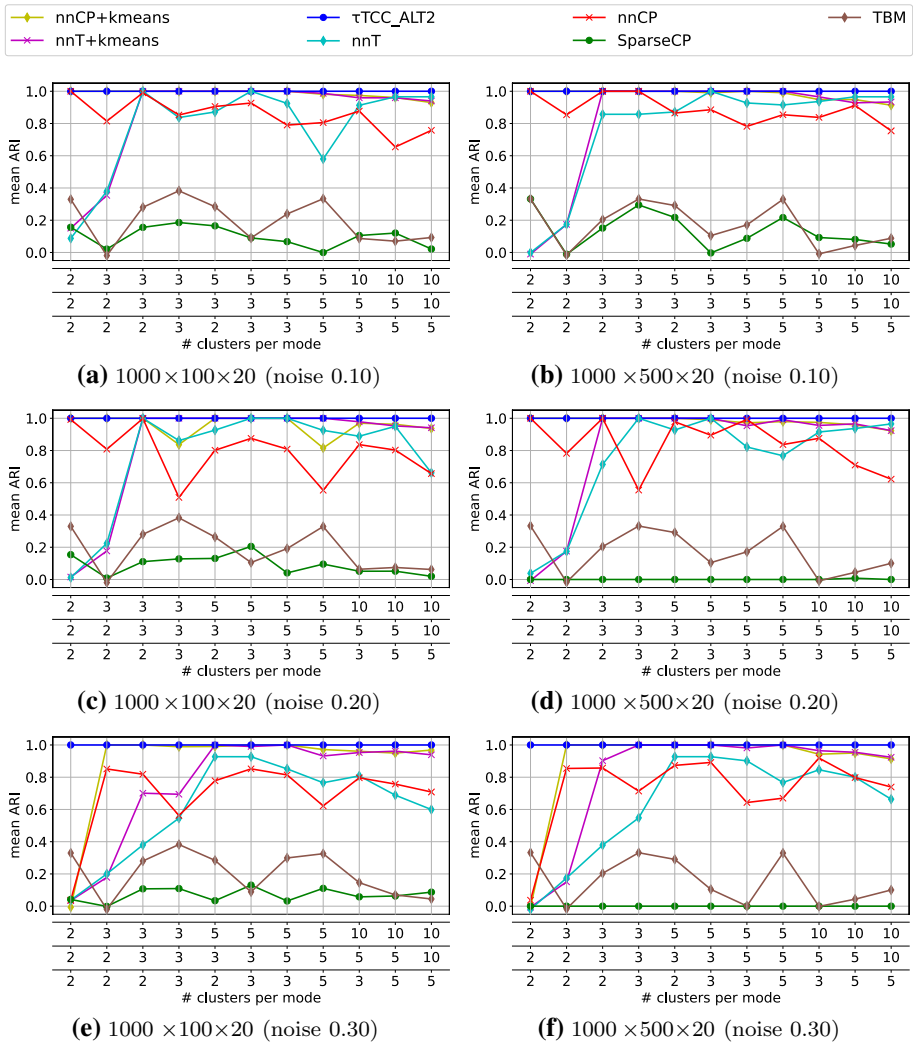
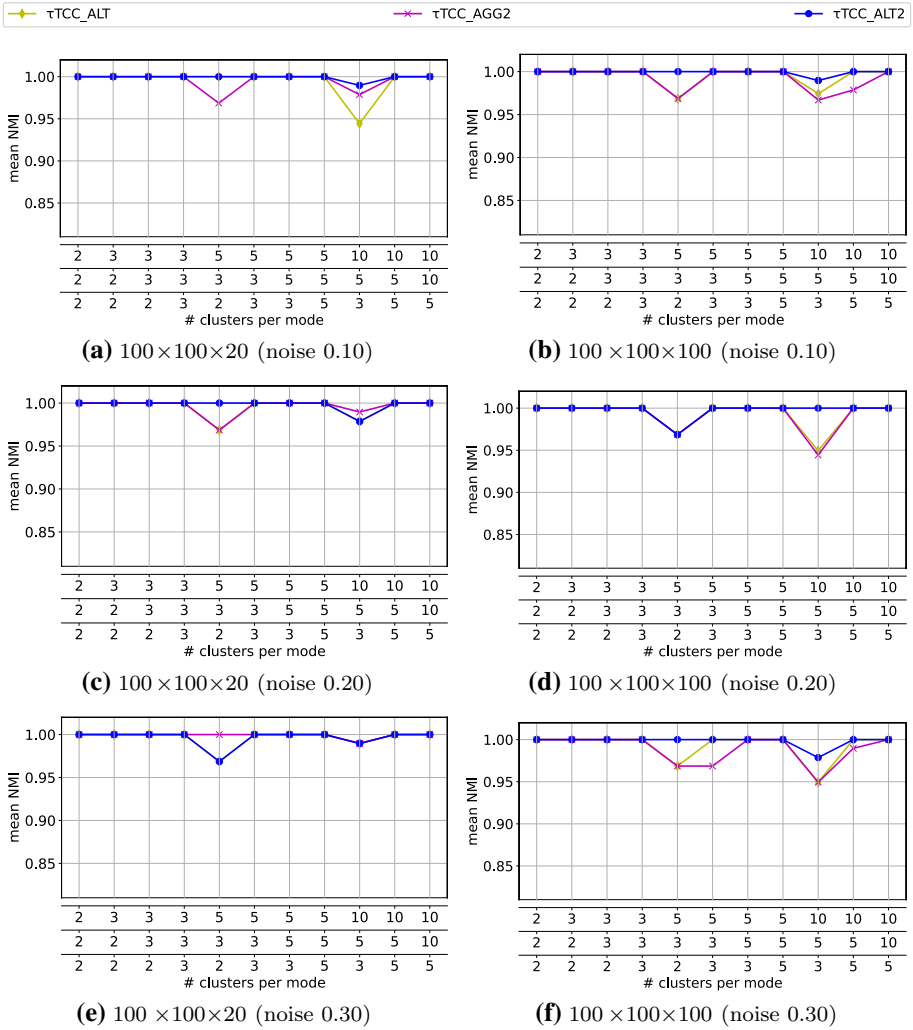**Fig. 14** Mean ARI on the three modes varying the number of embedded clusters on synthetic 3-way tensors with different sizes and levels of noise

6 Conclusions

The majority of tensor co-clustering algorithms optimizes objective functions that strongly depend on the number of co-clusters. This limits the correct application of such algorithms in realistic unsupervised scenarios. To address this limitation, we have
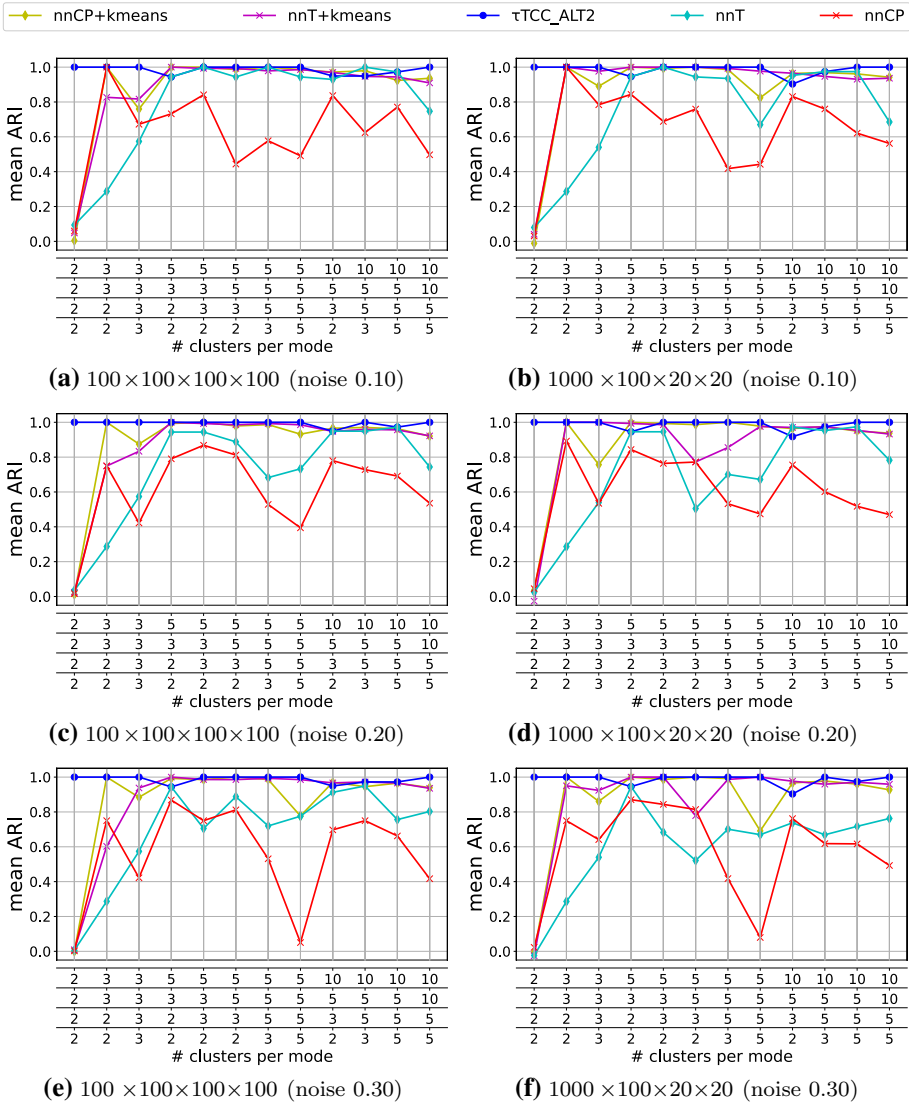
Fig. 15 Mean ARI on the four modes varying the number of embedded clusters on synthetic 4-way tensors with different sizes and levels of noise

introduced a new co-clustering algorithm specifically designed for tensors that does not require the desired number of clusters as input. We have proposed different variants of the algorithm, showing their theoretical and/or experimental convergence properties. Our experimental validation has shown that our approach outperforms state-of-the-art methods for most datasets. Even when our algorithms are not the best ones, we have
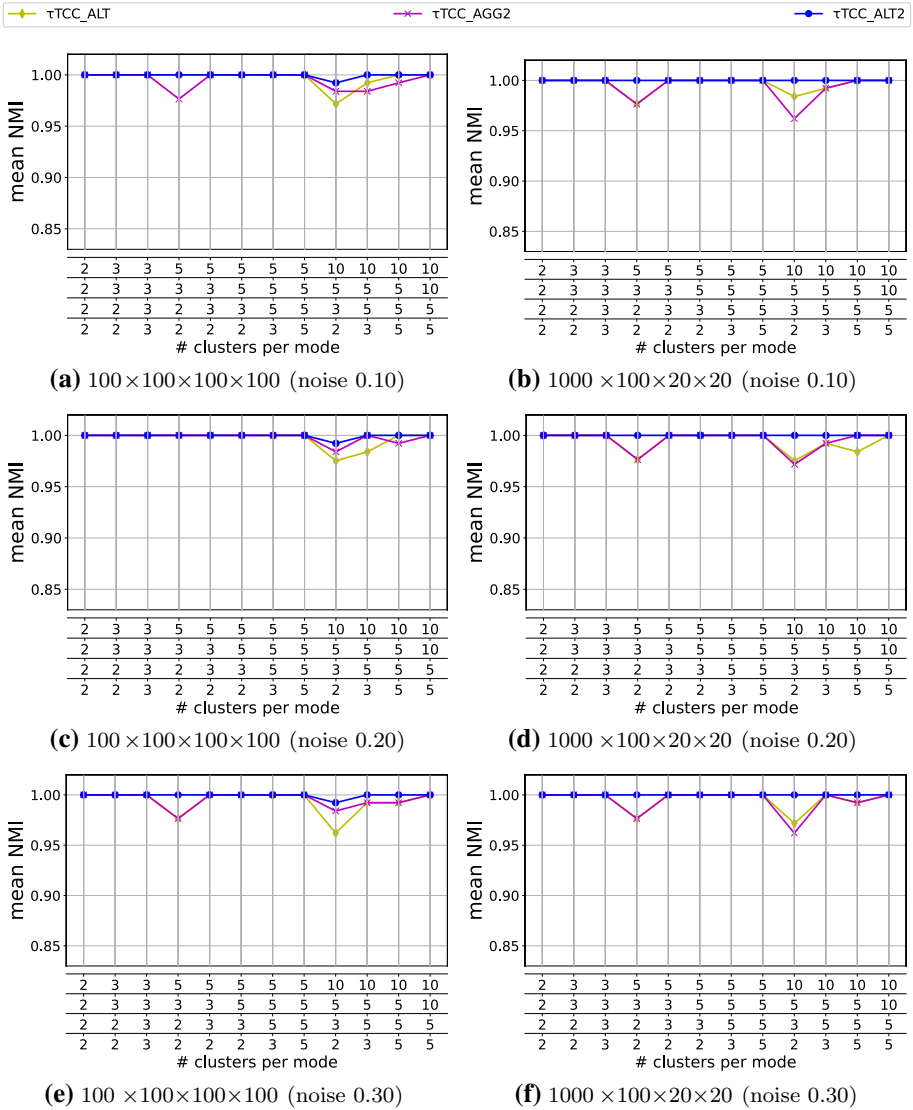
Fig. 16 Mean ARI on the four modes varying the number of embedded clusters on synthetic 4-way tensors with different sizes and levels of noise

found that the competitors can not work properly without specifying a correct number of clusters for each mode of the tensor. As future work, we will design a specific algorithm for sparse tensors with the aim of reducing the overall computational complexity of the approach. Finally, we will further investigate the ability of our method to identify candidate outliers as small clusters in the data.

Fig. 17 Mean NMI of the variants of τTCC on the three modes varying the number of embedded clusters on synthetic tensors with different sizes and levels of noise
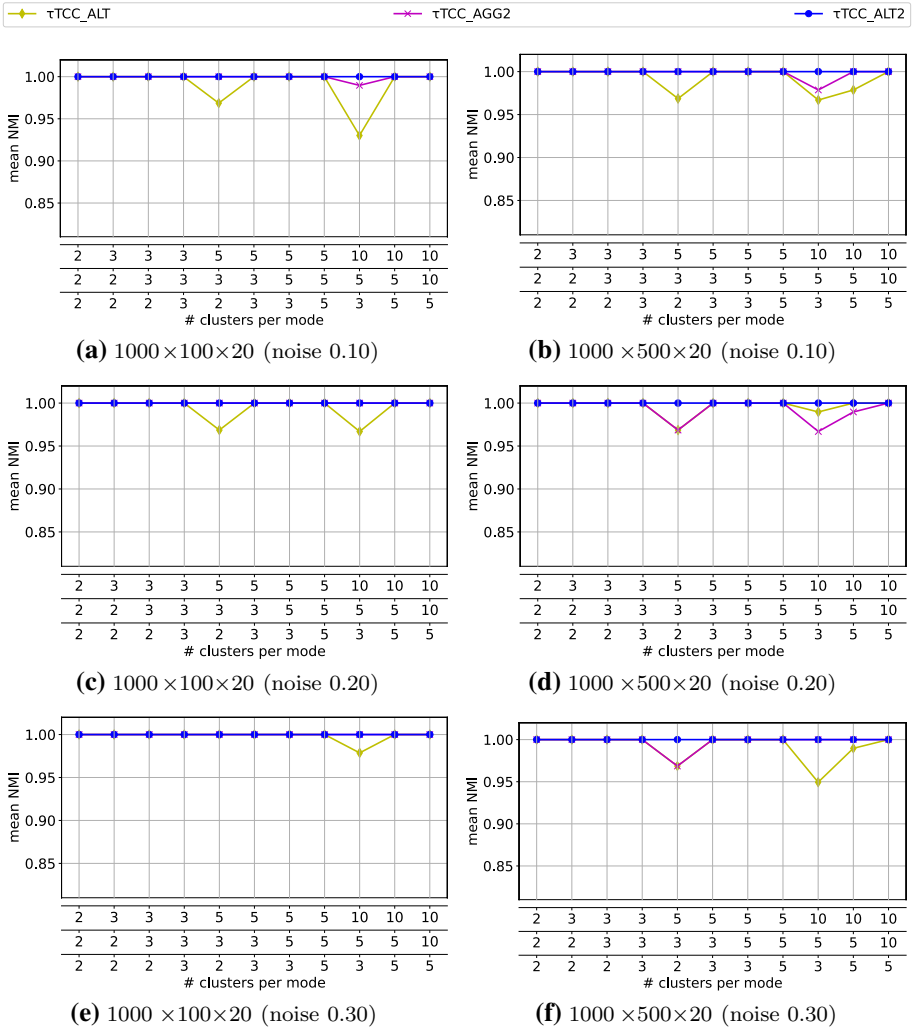
Fig. 18 Mean NMI of the variants of τTCC on the three modes varying the number of embedded clusters on synthetic tensors with different sizes and levels of noise

Fig. 19 Mean NMI of the variants of τTCC on the four modes varying the number of embedded clusters on synthetic 4-way tensors with different sizes and levels of noise
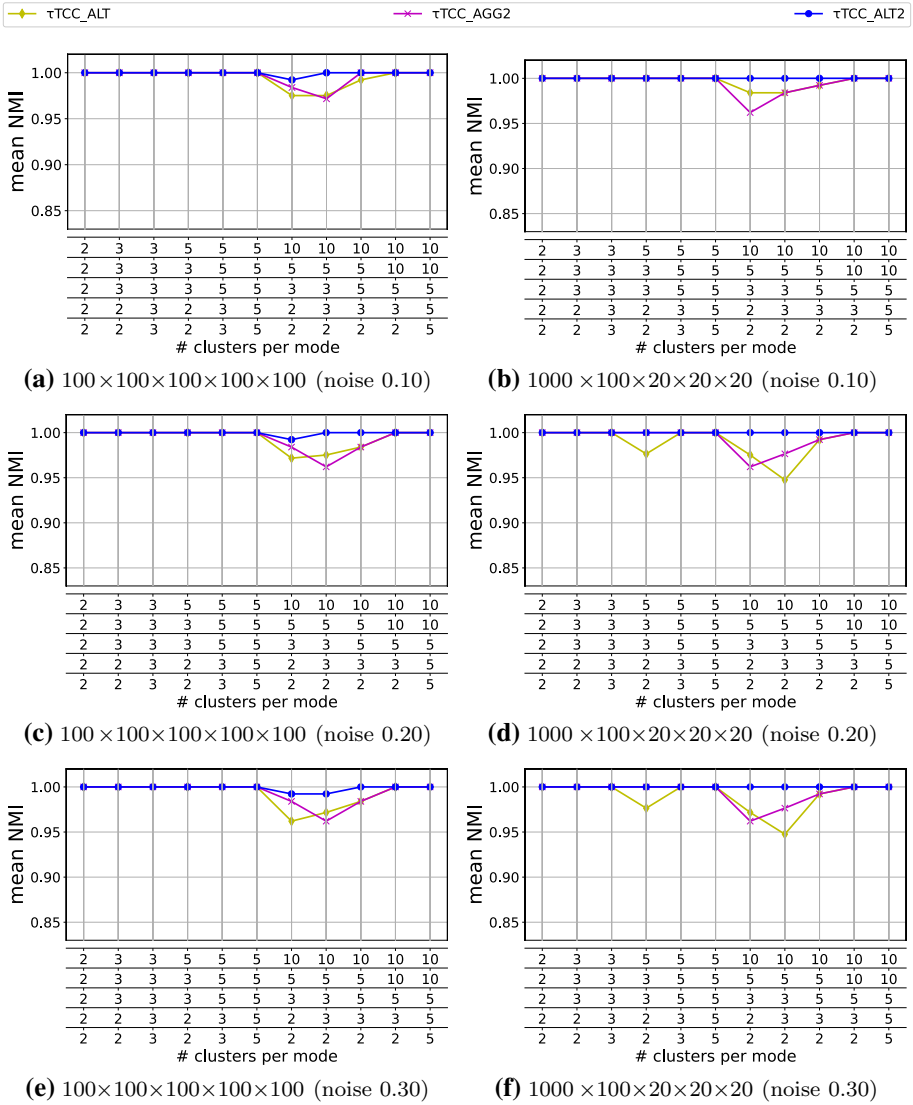
Fig. 20 Mean NMI of the variants of τTCC on the four modes varying the number of embedded clusters on synthetic 5-way tensors with different sizes and levels of noise

Appendix

Additional results

References

Araujo, M., Ribeiro, P. M. P., & Faloutsos, C. (2018). Tensorcast: Forecasting time-evolving networks with contextual information. *Proceedings of IJCAI, 2018, 5199*–5203.

Banerjee, A., Basu, S., & Merugu, S. (2007). Multi-way clustering on relation graphs. *Proceedings of SIAM SDM, 2007, 145*–156.

Battaglia, E. & Pensa, R. G. (2019). Parameter-Less Tensor Co-clustering. In: Proceedings of Discovery Sciences 2019, pp 205-219. Springer

Boutalbi, R., Labiod, L., Nadif, M. (2019a). Co-clustering from tensor data. In: Advances in Knowledge Discovery and Data Mining - 23rd Pacific-Asia Conference, PAKDD 2019, Macau, China, April 14-17, 2019, Proceedings, Part I, Springer, Lecture Notes in Computer Science, vol 11439, pp 370–383.

Boutalbi, R., Labiod, L., Nadif, M. (2019b). Sparse tensor co-clustering as a tool for document categorization. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019, ACM, pp 1157–1160.

Cantador, I., Brusilovsky, P., Kuflik, T., & (2011) 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec, . (2011). *In: Proceedings of the 5th ACM conference on Recommender systems* (p. 2011). New York, NY, USA, RecSys: ACM.

Cao, X., Wei, X., Han, Y., & Lin, D. (2015). Robust face clustering via tensor decomposition. *IEEE Trans Cybernetics, 45*(11), 2546–2557.

Cerf, L., Besson, J., Robardet, C., & Boulicaut, J. (2009). Closed patterns meet *n*-ary relations. *TKDD, 3*(1), 3:1-3:36.

Cerf, L., Besson, J., Nguyen, K., & Boulicaut, J. (2013). Closed and noise-tolerant patterns in n-ary relations. *Data Mining and Knowledge Discovery, 26*(3), 574–619.

Chakrabarti, D., Papadimitriou, S., Modha, D. S., & Faloutsos, C. (2004). Fully automatic cross-associations. *Proceedings of ACM SIGKDD, 2004, 79*–88.

Cho, H., Dhillon, I. S., Guan, Y., & Sra, S. (2004). Minimum sum-squared residue co-clustering of gene expression data. *Proceedings of SIAM SDM, 2004, 114*–125.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research, 7, 1*–30.

Dhillon, I. S., Mallela, S., & Modha, D. S. (2003). Information-theoretic co-clustering. *Proceedings of ACM SIGKDD, 2003, 89*–98.

Ding, C. H. Q., Li, T., Peng, W., & Park, H. (2006). Orthogonal nonnegative matrix t-factorizations for clustering. *Proceedings of ACM SIGKDD, 2006, 126*–135.

Ermis, B., Acar, E., & Cemgil, A. T. (2015). Link prediction in heterogeneous data via generalized coupled tensor factorization. *Data Mining and Knowledge Discovery, 29*(1), 203–236.

Goodman, L. A., & Kruskal, W. H. (1954). Measures of association for cross classification. *Journal of the American Statistical Association, 49, 732*–764.

Harshman, R.A. (1970). Foundation of the parafac procedure: models and conditions for an" explanatory" multimodal factor analysis. UCLA Working Papers in Phonetics 16:1–84.

He, J., Li, X., Liao, L., & Wang, M. (2018). Inferring continuous latent preference on transition intervals for next point-of-interest recommendation. *Proceesings of ECML PKDD, 2018, 741*–756.

Hong, M., & Jung, J. J. (2018). Multi-sided recommendation based on social tensor factorization. *Information Sciences, 447, 140*–156.

Huang, H., Ding, C.H.Q., Luo, D., Li, T. (2008). Simultaneous tensor subspace selection and clustering: The equivalence of high order svd and k-means clustering. In: Proceedings of the 14th ACM SIGKDD, pp 327–335.

Huang, W., Pan, Y., & Wu, J. (2012). Goodman-kruskal measure associated clustering for categorical data. *International Journal of Data Mining, Modelling and Management, 4*(4), 334–360.

Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification, 2*(1), 193–218.

Ienco, D., Robardet, C., Pensa, R. G., & Meo, R. (2013). Parameter-less co-clustering for star-structured heterogeneous data. *Data Mining and Knowledge Discovery, 26*(2), 217–254.

Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review, 51*(3), 455–500.

Nguyen, K., Cerf, L., Plantevit, M., Boulicaut, J. (2011). Multidimensional association rules in boolean tensors. In: Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011(April), pp. 28–30. (2011). *Mesa* (pp. 570–581). Arizona: USA, SIAM / Omnipress.

Papalexakis, E.E., Dogruöz, A.S. (2015). Understanding multilingual social networks in online immigrant communities. In: Proceedings of MWA 2015 (co-located with WWW 2015), pp 865–870.

Papalexakis, E. E., Sidiropoulos, N. D., & Bro, R. (2013). From K-means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE Trans Signal Processing, 61*(2), 493–506.

Pensa, R. G., Ienco, D., & Meo, R. (2014). Hierarchical co-clustering: off-line and incremental approaches. *Data Mining and Knowledge Discovery, 28*(1), 31–64.

Robardet, C., & Feschet, F. (2001). Efficient local search in conceptual clustering. *Proceedings of DS, 2001,* 323–335.

Shashua, A., Hazan, T. (2005). Non-negative tensor factorization with applications to statistics and computer vision. In: Proceedings of (ICML 2005, pp 792–799.

Strehl, A., & Ghosh, J. (2002). Cluster ensembles - A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research, 3,* 583–617.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika, 31,* 279–311.

Wang, M., & Zeng, Y. (2019). Multiway clustering via tensor block models. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8–14 December 2019* (pp. 713–723). BC, Canada: Vancouver.

Wu, T., Benson, A. R., & Gleich, D. F. (2016). General tensor spectral co-clustering for higher-order data. *Proceedings of NIPS, 2016,* 2559–2567.

Yu, K., He, L., Yu, P. S., Zhang, W., & Liu, Y. (2019). Coupled tensor decomposition for user clustering in mobile internet traffic interaction pattern. *IEEE Access, 7,* 18113–18124.

Zhang, T., & Golub, G. H. (2001). Rank-one approximation to high order tensors. *SIAM Journal on Matrix Analysis and Applications, 23*(2), 534–550.

Zhang, Z., Li, T., & Ding, C. H. Q. (2013). Non-negative tri-factor tensor decomposition with applications. *Knowledge and Information Systems, 34*(2), 243–265.

Zhao, L., Zaki, M.J. (2005). Tricluster: An effective algorithm for mining coherent clusters in 3d microarray data. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005, ACM, pp 694–705.

Zhou, Q., Xu, G., & Zong, Y. (2009). Web co-clustering of usage network using tensor decomposition. *Proceedings of ECBS, 2009,* 311–314.