Fast parameterless prototype-based co-clustering

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

© The Author(s) 2023

Abstract

Tensor co-clustering algorithms have been proven useful in many application scenarios, such as recommender systems, biological data analysis and the analysis of complex and evolving networks. However, they are significantly affected by wrong parameter configurations, since, at the very least, they require the cluster number to be set for each mode of the matrix/tensor, although they typically have other algorithm-specific hyper-parameters that need to be fine-tuned. Among the few known objective functions that can be optimized without setting these parameters, the Goodman–Kruskal τ—a statistical association measure that estimates the strength of the link between two or more discrete random variables—has proven its effectiveness in complex matrix and tensor co-clustering applications. However, its optimization in a co-clustering setting is tricky and, so far, has leaded to very slow and, at least in some specific but not unfrequent cases, inaccurate algorithms, due to its normalization term. In this paper, we investigate some interesting mathematical properties of τ, and propose a new simplified objective function with the ability of discovering an arbitrary and *a priori* unspecified number of good-quality co-clusters. Additionally, the new objective function definition allows for a novel prototype-based optimization strategy that enables the fast execution of matrix and higher-order tensor co-clustering. We show experimentally that the new algorithm preserves or even improves the quality of the discovered co-clusters by outperforming state-of-the-art competing approaches, while reducing the execution time by at least two orders of magnitude.

Keywords Parameter free clustering · Matrix methods · Tensor methods

1 Introduction

Clustering is by far one of the most popular machine learning tasks for its ability of extracting useful summaries of the data in the preliminary phases of a knowledge discovery process. When dealing with high-dimensional matrices or tensors, also called N-way data, clustering

---

[1] Department of Computer Science, University of Turin, C.so Svizzera 185, 10149 Turin, Italy

is often superseded by co-clustering (Dhillon, 2001), which consists in processing the modes (the row and the columns for matrices, i.e., 2-way tensors) of the input tensor simultaneously. This roughly corresponds to clustering one mode while performing dimensionality reduction on the other mode, for each mode alternately, and it helps discover more meaningful clusters (called co-clusters) as each cluster on one mode benefits from the clustering on the other modes. Intrinsically, column clusters also improve the interpretability of row clusters by providing useful summary information that might help explain each row grouping in a compact way. Unfortunately, clustering and, in particular, co-clustering, is often misused, since all existing methods produce meaningless clusters if their parameters are not set properly. In fact, most objective functions optimized by state-of-the-art co-clustering algorithms are strongly affected by the number of clusters, which is, typically, an input parameter. This is because two values of the same objective function computed on two different solutions with two different numbers of clusters can not be compared directly. In other words, they belong to two different search spaces. For instance, the loss of mutual information, optimized by Dhillon et al. (2003), is minimum for the discrete partition, i.e., when each cluster contains exactly one data instance. Typically, when the modes of a tensor are N, one has to set N different cluster numbers, plus any other algorithm-specific parameter to make a co-clustering method produce accurate results. As an example, a recent co-clustering method based on deep neural networks (Xu et al., 2019) requires at least four additional user-specified parameters, without considering the number of clusters for each mode and (many) other optional hyperparameters needed to setup and fine-tune the underlying neural network architecture.

To the best of our knowledge, the only co-clustering objective function least dependent on the number of clusters is the Goodman–Kruskal's τ association measure (Goodman and Kruskal, 1954). This measure has been originally designed to estimate the strength of the link between two discrete random variables, but it has shown its potential in several co-clustering algorithms (Robardet and Feschet, 2001; Ienco et al., 2013) as well, even for higher-order tensors (Battaglia and Pensa, 2023). Those algorithms do not require the specification of the correct number of clusters to find (sub)optimal solutions and, in general, are able to identify a meaningful number of clusters during the iterative optimization process. Despite this nice theoretical property, in some occasions, such algorithms find solutions that are deemed optimal according to τ, but provide poor descriptions of the data, as they include one or two very large clusters containing the majority of the input data instances plus some small or singleton clusters. Furthermore their iterative process turns out to be very long, since, to achieve the convergence, many iterations are needed, each one being computationally very expensive. The slow convergence of τ, especially in the very first iterations, counterbalances the advantages of being parameterless and limits its real applicability to small tensors with few modes.

To address these issues, we deeply investigate the mathematical properties and the convergence behavior of the Goodman–Kruskal's τ objective function. By analyzing the behavior in a controlled experimental setting, we show that the normalization term of the original association measure may lead the co-clustering algorithm towards meaningless solutions with almost all the rows grouped together in a unique cluster. We then propose a new objective function, derived from τ—thus preserving its good theoretical properties—that, contrary to the original Goodman–Kruskal's τ definition, allows us to define a similarity measure between a single item and a cluster prototype. This choice enables us to design a new optimization schema that reduces the number of necessary iterations and execution time significantly (by two orders of magnitude, on average), leading to time performances that are similar to those of state-of-the-art matrix co-clustering methods that require the numbers of clusters as input parameters. When applied to higher-order tensors, our algorithm is the fastest among all the competitors

considered in our study. Furthermore, the new definition mitigates the problem exhibited by the original τ formulation, which leads to co-clustering results that, although optimal from the point of view of the objective function, provide poor and thus useless descriptions of the data. The experiments, conducted on both synthetic and real-world matrices, also show that the improved time performances are not at the cost of the overall quality of the co-clusters. The remainder of the paper is organized as follows: a brief related literature review is reported in Sects. 2 and 3 presents some background notion on co-clustering. The fast co-clustering algorithm is introduced in Sect. 4, while its extension to tensors is presented in Sect. 5; the results of our experimental validation are discussed in Sect. 6; finally, we draw some conclusions in Sect. 7.

2 Related works

There exist several co-clustering techniques, tailored for different data types and application scenarios (Madeira and Oliveira, 2004; Govaert and Nadif, 2013; Qiu, 2004). Among the others, co-clustering has proven its effectiveness when applied to contingency tables, i.e. matrices reporting the co-occurrences between two sets of categorical values, such as documents/words, customers/products or users/movies data objects. In fact, these matrices are typically high-dimensional and extremely sparse and, thus, they cannot be handled by classical clustering methods. Furthermore, the columns of these matrices represent instances of the same categorical feature (words, products or movies, in the examples above), thus grouping them in clusters makes sense and provides additional useful information.

The main methods for contingency tables co-clustering can be roughly divided into three groups: spectral methods (Dhillon, 2001; Kluger et al., 2003), which transform the co-clustering problem into a partitioning problem on a bipartite graph; matrix factorization based methods, such as Block Value Decomposition (Long et al., 2005) and Non-Negative Matrix Tri-Factorization (Ding et al., 2006; Yoo and Choi, 2010); probabilistic approaches (Govaert and Nadif, 2010; Ailem et al., 2017), which assume the data to be generated by a probabilistic model and try to recover the underlying model parameters; methods that formulate the co-clustering problem as an optimization problem for some measure of the quality of the co-clustering solution. Example of such measures are the loss of Mutual Information (Dhillon et al., 2003), any Bregman divergence (Banerjee et al., 2007), the Goodman and Kruskal's τ measure (Robardet and Feschet, 2001; Ienco et al., 2013), the graph modularity (Ailem et al., 2016). Instead, Xu et al. (2019) propose a deep learning model for co-clustering, that uses deep autoencoders for dimensionality reduction and Gaussian Mixture Models to infer the clustering assignments. Many extensions of the traditional co-clustering problem have been studied: as few examples, there exist algorithms to perform co-clustering in a distributed-scenario (Papadimitriou and Sun, 2008), algorithms for co-clustering heterogeneous data (Gao et al., 2005; Ienco et al., 2013), or hierarchical methods (Pensa et al., 2014; Wei et al., 2021).

In the last 3 years, the interests of scientists has significantly shifted towards multi-view approaches, with several proposal aimed at solving this type of problem (Du et al., 2021; Hussain et al., 2022). Other recent approaches have focused on improving or speeding up matrix factorization algorithms for co-clustering (Chen et al., 2023b; Wang and Ma, 2021), or to adapt the objective functions to more complex problems (Deng et al., 2021; Tang and Wan, 2021). Wang et al. (2021), instead, address the problem of multiple alternative clustering, by introducing the first algorithm that generates multiple alternative co-clusterings at the same time.

Further recent contributions are aimed at improving model-based text co-clustering (Affeldt et al., 2021b, a), or enhancing bipartite graph partition-based co-clustering (Chen et al., 2023a). Finally, the problem of correctly evaluating co-clusters has also been addressed recently by Robert et al. (2021).

With the exception of the methods based on the Goodman–Kruskal's τ or its derivations (Robardet and Feschet, 2001; Pensa et al., 2014; Ienco et al., 2013), all other approaches requires the specification of the number of desired or expected clusters in each mode. Interestingly, generalizations of some of these methods work on higher-order tensors as well (Papalexakis et al., 2013; Boutalbi et al., 2019a; Battaglia and Pensa, 2023).

In the last decades, in fact, tensors have attracted a lot of attention due to their intrinsic complexity and richness. Several *ad hoc* extensions of 2-way matrix methods have been proposed and, among them, also co-clustering has been naturally extended to handle tensors. The main tensor co-clustering methods are often based on tensor decomposition models (Zhou et al., 2009; Peng and Li, 2010; Papalexakis et al., 2013; Zhang et al., 2013), such as PARAFAC/CANDECOMP or Non-Negative Tucker decomposition. Instead, Wu et al. (2016) introduce a spectral co-clustering method based on a new random walk model for nonnegative square tensors; in Chi et al. (2020) the tensors are represented as weighted hypergraphs, which are cut by an algorithm based on random sampling. 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. Wang and Zeng (2019) present a co-clustering approach for tensors that uses a least-square estimation procedure for identifying a N-way block structure that applies to binary, continuous, and hybrid data instances. Boutalbi et al. (2022) introduce the Tensor Latent Block Model (TLBM) for the co-clustering of tensor data, that exploits the latent block model and is able to consider continuous, binary and count data. According to the authors, the derived algorithms can be also used for the clustering of multiple graphs or multi-view clustering. Finally, Battaglia and Pensa (2023) propose a tensor extension of the 2-way co-clustering algorithm based on the optimization of the Goodman and Kruskal's τ association measure, which is, to the best of our knowledge, the only tensor co-clustering approach not requiring the number of clusters as input parameter. Unfortunately, it is also very slow and, as we will show in the next sections, sometimes leads to degenerate solutions. These are exactly the two issues that we address in this paper, by proposing a novel more effective derivation of the Goodman–Kruskal's τ objective and a more efficient prototype-based algorithm for its fast optimization.

3 Background and problem formulation

Before introducing the tensor co-clustering problem, we briefly describe the notation used in this paper. As a general rule, non indexed (e.g., k) lowercase letters represent scalars, while indexed (e.g, p_{ic}) lowercase letters represent the single entries of a vector/matrix/tensor (with some exceptions, e.g., k_0). Instead, uppercase Latin or Greek symbols (e.g., A, Γ) represent some quantities computed on matrices or tensors, with the exception of X, Y and Z that denote some categorical random variables. Tensor are represented according to two different notations: script uppercase letters (for input data) or bold uppercase letters (for tensors computed during the execution of the algorithms). Matrices are represented similarly. Script uppercase letters are also used to represent sets, in our case, partitions (sets of clusters)

Table 1 Symbols and notation used in the paper

Notation	Description
$\mathcal{A}$	An input data tensor (or matrix)
$(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(m)})$	A co-clustering of a m-mode tensor $\mathcal{A}$
$(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$	A co-clustering of a 3-mode tensor $\mathcal{A}$
$(\mathcal{R}, \mathcal{C})$	A co-clustering of a matrix $\mathcal{A}$
$\mathcal{R}_r$	The rth cluster of a row partition $\mathcal{R}$
$\mathcal{C}_c$	The cth cluster of a column partition $\mathcal{C}$
T	The contingency tensor/matrix associated to the input data $\mathcal{A}$ and to a co-clustering $(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(m)})$ (resp. $(\mathcal{R}, \mathcal{C})$)
$\mathcal{R}^{(t)}$	The row partition at iteration t
$\mathcal{C}^{(t)}$	The column partition at iteration t
$\mathcal{P}^{i\,(t)}$	The i-mode partition at iteration t
P (matrices)	The probability matrix associated to $\mathcal{A}$ given the column clustering $\mathcal{C}$
Q (matrices)	The probability matrix associated to the co-clustering $(\mathcal{R}, \mathcal{C})$ of a matrix $\mathcal{A}$
P (tensors)	The probability tensor associated to one mode of a $\mathcal{A}$ given the clustering on all other modes
Q (tensors)	The probability tensor associated to the co-clustering $(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(m)})$ of a tensor $\mathcal{A}$

and clusters (subsets of points). For a complete detailed distinction, we let the reader refer to Table 1. In the following, we formulate the tensor co-clustering problem as the maximization of an association measure between categorical random variables.

Given a tensor $\mathcal{A}$ with m modes, a co-clustering of $\mathcal{A}$ is a tuple $(\mathcal{P}^{(1)}, \dots, \mathcal{P}^{(m)})$ where $\mathcal{P}^{(i)}$ is a partition of the elements of the ith mode of $\mathcal{A}$, for each $i = 1, \dots, m$. As a special case, if $\mathcal{A}$ is a matrix (i.e. a 2-way tensor), a co-clustering of $\mathcal{A}$ is a pair $(\mathcal{R}, \mathcal{C})$, where $\mathcal{R}$ is a partition of the rows and $\mathcal{C}$ a partition of the columns of the matrix. In this paper we present a new (tensor) co-clustering algorithm based on the optimization of the Kruskal and Goodman's τ measure, such as τCC (Ienco et al., 2013) and τTCC (Battaglia and Pensa, 2023), but using a different optimization strategy that makes the algorithm noticeably faster.

The objective function of τCC is Kruskal and Goodman's τ (Goodman and Kruskal, 1954), 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:

$$\tau_{X|Y} = \frac{e_X - \mathbb{E}[e_{X|Y}]}{e_X} \tag{1}$$

where e_X is the error in predicting X, i.e., the probability that two independent realizations of the random variable differ, and $\mathbb{E}[e_{X|Y}]$ is the expected value of the error in predicting X when Y is known. The values of $\tau_{X|Y}$ stay between 0 (when there is no association between X and Y) and 1 (when X is function of Y). Function τ can be used to evaluate a co-clustering solution: given a matrix $\mathcal{A} = (a_{ij})$ with non-negative values and a co-clustering $(\mathcal{R}, \mathcal{C})$ of $\mathcal{A}$, the association of the row clustering $\mathcal{R}$ with the column clustering $\mathcal{C}$ can be expressed as

$$\tau_{R|C}(\mathcal{R}, \mathcal{C}) = \frac{\sum_{r=1}^{|\mathcal{R}|} \sum_{c=1}^{|\mathcal{C}|} \frac{t_{rc}^2}{t_{..}t_{.c}} - \sum_{r=1}^{|\mathcal{R}|} \frac{t_{r.}^2}{t_{..}^2}}{1 - \sum_{r=1}^{|\mathcal{R}|} \frac{t_{r.}^2}{t_{..}^2}} \tag{2}$$

where $\mathbf{T} = (t_{rc})$ is the contingency table associated to the co-clustering, i.e. $t_{rc} = \sum_{i \in \mathcal{R}_r} \sum_{j \in \mathcal{C}_c} a_{ij}$, $t_{..} = \sum_{r=1}^{|\mathcal{R}|} \sum_{c=1}^{|\mathcal{C}|} t_{rc}$, $t_{.c} = \sum_{r=1}^{|\mathcal{R}|} t_{rc}$, and $t_{r.} = \sum_{c=1}^{|\mathcal{C}|} t_{rc}$. Analogously, the association of the column clustering $\mathcal{C}$ to the row clustering $\mathcal{R}$ can be evaluated through the function $\tau_{C|R}(\mathcal{R}, \mathcal{C})$. Notice that τ is not symmetric, thus $\tau_{R|C}(\mathcal{R}, \mathcal{C}) \neq \tau_{C|R}(\mathcal{R}, \mathcal{C})$ in general. The best co-clustering solutions are those that simultaneously maximize $\tau_{R|C}$ and $\tau_{C|R}$. An intuitive explanation of the meaning of $\tau_{R|C}(\mathcal{R}, \mathcal{C})$ and $\tau_{C|R}(\mathcal{R}, \mathcal{C})$ and its implication for a co-clustering solution, is given in the example below.

Example 1 Suppose that $\mathcal{A}$ is a sparse matrix over a set of 10 customers (the rows of $\mathcal{A}$) and a set of 8 products (the columns of $\mathcal{A}$). Assume that $\mathcal{A}$ is as follows

$$\mathcal{A} = \begin{pmatrix} & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 \\ r_1 & 3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ r_2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ r_4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_5 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ r_6 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 1 \\ r_7 & 0 & 0 & 0 & 0 & 5 & 0 & 1 & 0 \\ r_8 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 1 \\ r_9 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ r_{10} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 3 \end{pmatrix},$$

where each entry (r_i, c_j) is the count of products c_j purchased by customer r_i. A possible co-clustering over $\mathcal{A}$ is $(\mathcal{R}, \mathcal{C})$, where $\mathcal{R} = \{\{r_1, r_2, r_3\}, \{r_4, r_5, r_6\}, \{r_7, r_8\}, \{r_9, r_{10}\}\}$, is the partition of customers and $\mathcal{C} = \{\{c_1, c_2\}, \{c_3, c_4\}, \{c_5, c_6\}, \{c_7, c_8\}\}$ is the partition of products. We can associate, to the triple $(\mathcal{A}, \mathcal{R}, \mathcal{C})$, the following contingency table $\mathbf{T}$ obtained by summing all the rows (and columns) belonging to the same cluster together:

$$\mathbf{T} = \begin{pmatrix} 5 & 2 & 0 & 0 \\ 1 & 10 & 0 & 1 \\ 0 & 0 & 10 & 2 \\ 0 & 1 & 0 & 10 \end{pmatrix}.$$

The contingency matrix $\mathbf{T}$ shows that $(\mathcal{R}, \mathcal{C})$ is a meaningful co-clustering solution, with well defined block co-clusters. The values of the two association measures for this co-clustering are $\tau_{R|C}(\mathcal{R}, \mathcal{C}) = 0.630$ and $\tau_{C|R}(\mathcal{R}, \mathcal{C}) = 0.625$. Now, consider a different co-clustering solution $(\mathcal{R}', \mathcal{C}')$, with $\mathcal{R}' = \{\{r_1, r_5, r_{10}\}, \{r_2, r_6, r_9\}, \{r_3, r_8\}, \{r_4, r_7\}\}$ and $\mathcal{C}' = \{\{c_1, c_8\}, \{c_3, c_7\}, \{c_2, c_6\}, \{c_4, c_5\}\}$

$$\mathbf{T}' = \begin{pmatrix} 6 & 0 & 0 & 8 \\ 3 & 7 & 0 & 4 \\ 1 & 1 & 5 & 0 \\ 0 & 1 & 1 & 5 \end{pmatrix}.$$

In this case, the contingency matrix $\mathbf{T}'$ exhibits a meaningless co-clustering structure, with less crisp block co-clusters. This is confirmed by the values of the two association measures: in fact, $\tau_{R|C}(\mathcal{R}', \mathcal{C}') = 0.300$ and $\tau_{C|R}(\mathcal{R}', \mathcal{C}') = 0.270$.

The algorithm τCC belongs to the family of co-clustering algorithms that iteratively maximize a measure of the quality of the co-clustering (the most famous example of is the Information Theoretic Co-Clustering or ITCC (Dhillon et al., 2003). However, τCC has three peculiarities that distinguish it from the other methods.

1. Instead of maximizing a unique objective function, τCC's goal is to maximize two objective functions at the same time, $\tau_{R|C}$ and $\tau_{C|R}$. Thus, co-clustering is formulated as a multi-objective optimization problem.
2. A stochastic local search approach is used to solve the maximization problem: at each iteration, a row is randomly selected and is moved from its original cluster to the row cluster that maximizes $\tau_{R|C}$. Then, a column is randomly selected and moved to the column cluster that maximizes $\tau_{C|R}$. Instead, the other methods use a "k-means like" optimization strategy that constructs prototypes for each cluster and, at each iteration, assigns each object to the closest prototype, using some *ad hoc* function to measure the closeness, opportunely designed to increase the objective function.
3. Unlike other objective functions, τ exhibits no monotonicity w.r.t. the number of clusters. This fact enables the algorithm τCC to compare co-clustering solutions with a different number of clusters and choose the best one: as a consequence, τCC is one of the few co-clustering algorithms able to identify the final number of clusters on its own. Instead, the magnitude of other objective functions, such as the Loss of Mutual Information adopted by ITCC, depends on the number of clusters and thus the other methods only consider co-clustering solutions with a specified number of clusters on each mode.

Despite the desirable property (3) and the proven effectiveness of the algorithm, the applicability of τCC is limited by the fact that its current optimization strategy requires an elevated number of iterations, resulting in execution times that are sensibly higher than those of the other co-clustering methods. Furthermore, in some cases, the algorithm fails in identifying a meaningful number of co-clusters, ending in a solution with a very small number of very large co-clusters unable to faithfully describe the data. In the next sections, we will present a slightly different objective function and a new optimization strategy for the algorithm τCC which, similarly to that of ITCC, is based on the assignment of the rows and columns to the most similar prototype.

In a recent work (Battaglia and Pensa, 2023), we have extended the association measure τ to handle a generic number m of categorical random variables. For instance, it is possible to measure the association among three random variables X, Y and Z as the proportional reduction of the error in predicting one of them knowing the joint distribution of the other two, obtaining three different association measures $\tau_{X|(Y,Z)}$, $\tau_{Y|(X,Z)}$ and $\tau_{Z|(X,Y)}$. If X, Y, Z denote the random variables representing to the co-clustering problem and $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a co-clustering of a 3-way tensor $\mathcal{A}$, the association measure can be written as

$$\tau_{X|(Y,Z)}(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = \frac{\sum_{i=1}^{|\mathcal{X}|} \sum_{j=1}^{|\mathcal{Y}|} \sum_{k=1}^{|\mathcal{Z}|} \frac{t_{ijk}^2}{t_{\cdot jk}} - \sum_{i=1}^{|\mathcal{X}|} \frac{t_{i\cdot\cdot}^2}{t_{\cdot\cdot\cdot}^2}}{1 - \sum_{i=1}^{|\mathcal{X}|} \frac{t_{i\cdot\cdot}^2}{t_{\cdot\cdot\cdot}^2}} \tag{3}$$

where $\mathbf{T} = (t_{ijk})$ is the contingency tensor associated to the co-clustering. The tensor co-clustering problem is then formulated as a multi-objective optimization problem of the functions $\tau_{X|(Y,Z)}$, $\tau_{Y|(X,Z)}$ and $\tau_{Z|(X,Y)}$. Analogously to τCC, the tensor co-clustering algorithm, called τTCC, uses a stochastic local search approach to solve the maximization problem. Notice that, when $m = 2$, i.e., the contingency tensor is a matrix, τTCC corresponds to τCC. The advantages and disadvantages of the 2-way version of the algorithms

hold for the tensor version as well, but their effect becomes more pronounced as the number of modes of the tensors increases. The algorithm does not require the specification of any parameter, while all existing tensor co-clustering methods require at least the specification of the number of clusters on each mode of the tensor and, very often, other parameters such as the rank of the decomposition. On the other hand, the number of iterations performed by the algorithm and its computational complexity increase with the number of modes, limiting the practical application of the algorithm τTCC to relatively small tensors with few modes.

4 Fast parameterless co-clustering

In this section, we present a new objective function that leads to an arbitrary number of more meaningful co-clusters and allows for a more efficient and faster optimization strategy leveraging a prototype-based iterative algorithm whose theoretical convergence can be proved formally.

4.1 An alternative to Goodman and Kruskal's τ

Let X and Y be two categorical random variables. The function $\tau_{X|Y}$ (Eq. 1) depends on two quantities: the error in predicting X, denoted by e_X, and the expected error in predicting X when Y is known, denoted by $\mathbb{E}[e_{X|Y}]$, both laying in [0, 1]. The association between X and Y is strong if the knowledge of Y drastically reduces the error e_X. Notice that e_X does not depend on Y and its presence in the definition of τ serves as a starting point from which to calculate the improvement in the prediction due to the knowledge of Y. Thus, intuitively, the most important component in $\tau_{X|Y}$ should be $\mathbb{E}[e_{X|Y}]$. However, since $\tau_{X|Y}$ measures the *proportional* reduction of the error, e_X contributes to both the numerator and denominator of (1). Consequently, it has a great impact on the value of $\tau_{X|Y}$. As an example, imagine that X is a random variable with a very small e_X. Since, by definition, $\mathbb{E}[e_{X|Y}] < e_X$, then the expected error, when Y is known, will be very small as well, i.e., $\mathbb{E}[e_{X|Y}] \approx e_X$. However, although the knowledge about the distribution of Y hardly improves the prediction of X, the value of the association measure is high, due to the small denominator of $\tau_{X|Y}$. Such a situation happens also when τ, formulated as in Eq. 2, is employed to measure the quality of a co-clustering $(\mathcal{R}, \mathcal{C})$: in particular, e_R is small when the partition $\mathcal{R}$ is composed by a unique cluster, or by a huge cluster and few very small extra-clusters. In fact, the function $e_R(\mathcal{R}) = 1 - \sum_{r=1}^{|\mathcal{R}|} t_{r.}^2$ has a maximum when $|\mathcal{R}| = 1$. This fact may lead the algorithm τCC towards a meaningless clustering solution with (almost) all the rows grouped together into a unique cluster. A simple solution to this problem is to leave out the denominator of τ and only consider, as objective function of the co-clustering algorithm, the numerator. From now on, we will call this simplified objective function $\hat{\tau}_{R|C}$, defined as

$$\hat{\tau}_{R|C} = \sum_{r=1}^{|\mathcal{R}|} \sum_{c=1}^{|\mathcal{C}|} \frac{t_{rc}^2}{t_{..} t_{.c}} - \sum_{r=1}^{|\mathcal{R}|} \frac{t_{r.}^2}{t_{..}^2}. \tag{4}$$

To better investigate the differences in the behavior of $\tau_{R|C}$ and $\hat{\tau}_{R|C}$, the two measures are plotted in Fig. 1 as functions of the two variables e_R and $\mathbb{E}[e_{R|C}]$. For the moment, R and C are just two families of generic categorical random variables. As expected, $\tau_{R|C}$ is very high for almost all values of e_R, when combined to low values of $\mathbb{E}[e_{R|C}]$. In the lower left corner of the two heatmaps are instances of the family R with low prediction error e_R. Hence,
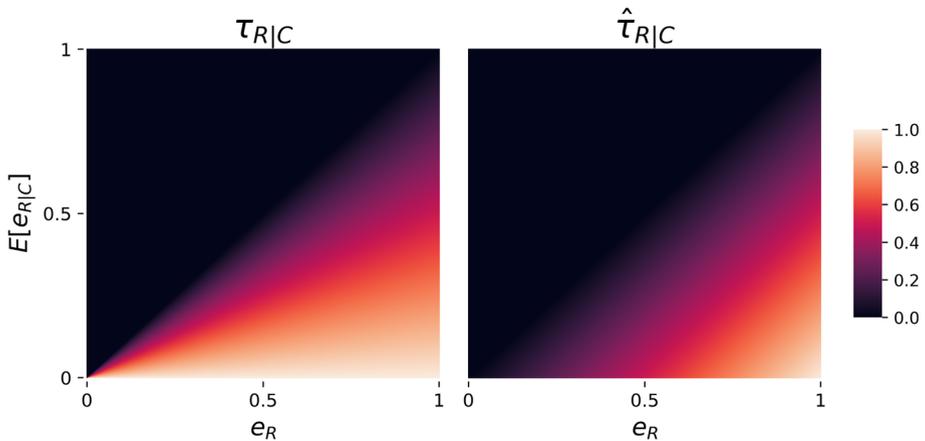
Fig. 1 Comparison between $\tau_{R|C}$ and $\hat{\tau}_{R|C}$. The two measures are considered as functions of variables e_R and $\mathbb{E}[e_{R|C}]$

when R is the family of random variables associated to each possible clustering solution, in the lower left corner there are the clustering solutions with very few clusters: according to association measure τ, these clustering solutions are optimal and the algorithm τCC could end in one of these solutions. Instead, $\hat{\tau}_{X|Y}$ rewards more the pair of solutions (R, C) with e_R much bigger than $\mathbb{E}[e_{R|C}]$. Since $\mathbb{E}[e_{R|C}] = 1 - \sum_{rc} \frac{t_{rc}^2}{t_{.c}}$ is minimum when the contingency matrix of the co-clustering is diagonal, the newly proposed objective function $\hat{\tau}_{X|Y}$ is more suitable when we want to obtain co-clustering solutions with approximately the same number of clusters on rows and columns and block diagonal co-clusters. The following example will show the differences between the two functions when used to evaluate co-clustering.

Example 2 Let us consider the customers $\times$ products matrix $\mathcal{A}$, first introduced in Example 1, together with the first related co-clustering solution represented by the contingency matrix **T**. The values of the two association measures for this co-clustering are $\tau_{R|C}(\mathcal{R}, \mathcal{C}) = 0.630$ and $\hat{\tau}_{R|C}(\mathcal{R}, \mathcal{C}) = 0.466$. Consider now a different co-clustering solution $(\mathcal{R}'', \mathcal{C})$ on $\mathcal{A}$, where $\mathcal{R}'' = \{\{r_1, r_2, r_4\}, \{r_3, r_5, r_6, r_7, r_8, r_9, r_{10}\}\}$ (notice that the partition on columns is the same as in the previous solution), with contingency matrix

$$\mathbf{T}'' = \begin{pmatrix} 6 & 1 & 0 & 0 \\ 0 & 12 & 10 & 13 \end{pmatrix}.$$

Here we have $\tau_{R|C}(\mathcal{R}'', \mathcal{C}) = 0.842$ and $\hat{\tau}_{R|C}(\mathcal{R}'', \mathcal{C}) = 0.234$. Therefore, according to measure τ the second partition is better than the first one, while $\hat{\tau}$ assigns an higher score to the first solution. Clearly, without any other knowledge about the dataset, it is not possible to say whether a solution is better than the other. However, it is evident that the first solution provides a more detailed summary of the data and it can also be used to interpret the column clustering more accurately. Instead, the second solution contains a large row cluster of customers that purchase many different products, described by many different column clusters.

The new association measure $\hat{\tau}_{R|C}$ has another advantage: the simplification of the original objective function allows us to use a more efficient optimization schema in the co-clustering algorithm, as we will show in the next section.

4.2 Prototype-based τCC

Let $\mathcal{A} = (a_{ij}) \in \mathbb{R}_+^{n \times m}$ be a non-negative data matrix, with n rows and m columns, and $\mathbf{T}$ the contingency matrix associated to the co-clustering $(\mathcal{R}, \mathcal{C})$, where $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_k)$ and $\mathcal{C} = (\mathcal{C}_1, \ldots, \mathcal{C}_l)$, with $1 < k \leq n$ and $1 < l \leq m$. The generic element of matrix $\mathbf{T}$ is $t_{rc} = \sum_{i \in \mathcal{R}_r} \sum_{j \in \mathcal{C}_c} a_{ij}$, for $r = 1, \ldots, k$ and $c = 1, \ldots, l$. We will denote the sum of all the elements of $\mathcal{A}$ with A and the sum of all the elements of $\mathbf{T}$ with T. It follows that $A = T$. Let us define the following quantities:

$$p_{ij} = \frac{a_{ij}}{A} \quad p_{i\cdot} = \sum_{j=1}^{m} \frac{a_{ij}}{A} \quad p_{\cdot j} = \sum_{i=1}^{n} \frac{a_{ij}}{A} \tag{5}$$

$$q_{rc} = \frac{t_{rc}}{T} = \frac{t_{rc}}{A} = \sum_{i \in \mathcal{R}_r} \sum_{j \in \mathcal{C}_c} \frac{a_{ij}}{A} = \sum_{i \in \mathcal{R}_r} \sum_{j \in \mathcal{C}_c} p_{ij} \tag{6}$$

$$q_{r\cdot} = \sum_{c=1}^{l} \frac{t_{rc}}{T} = \sum_{c=1}^{l} \frac{t_{rc}}{A} = \sum_{c=1}^{l} \sum_{i \in \mathcal{R}_r} \sum_{j \in \mathcal{C}_c} \frac{a_{ij}}{A} = \sum_{i \in \mathcal{R}_r} p_{i\cdot} \tag{7}$$

$$q_{\cdot c} = \sum_{r=1}^{k} \frac{t_{rc}}{T} = \sum_{r=1}^{k} \frac{t_{rc}}{A} = \sum_{r=1}^{k} \sum_{i \in \mathcal{R}_r} \sum_{j \in \mathcal{C}_c} \frac{a_{ij}}{A} = \sum_{j \in \mathcal{C}_c} p_{\cdot j} \tag{8}$$

where $i = 1, \ldots, n$, $j = 1, \ldots, m$, $r = 1, \ldots, k$ and $l = 1, \ldots, l$. Intuitively, p_{ij} can be interpreted as the probability of observing a_{ij}, while q_{rc} is the probability of observing a generic co-cluster $(\mathcal{R}_r, \mathcal{C}_c)$. Consequently, $p_{i\cdot}$ and $p_{\cdot j}$ are, respectively, the marginal probabilities of the object represented by the ith row and of the item represented by the jth column of $\mathcal{A}$. Finally, $q_{r\cdot}$ and $q_{\cdot c}$ are, respectively, the probability that a generic row is in cluster $\mathcal{R}_r$ and the probability that a generic column falls in cluster $\mathcal{C}_c$.

Hence, both $p = (p_{ij})$ and $q = (q_{rc})$ are probability distributions and $(p_{\cdot j})$, $(p_{i\cdot})$, $(q_{\cdot c})$, $(q_{r\cdot})$ are their marginal distributions. However, while p only depends on the original data matrix $\mathcal{A}$, q also depends on the partitions $\mathcal{R}$ and $\mathcal{C}$ on the rows and columns of $\mathcal{A}$, respectively. Let us consider the partition $\mathcal{C}$ fixed and suppose we want to identify the partition $\mathcal{R}$ that maximizes $\hat{\tau}_{R|C}(\cdot, \mathcal{C})$. We will use the following notation:

$$p_{ic} = \sum_{j \in \mathcal{C}_c} p_{ij} \quad p_{\cdot c} = \sum_{j \in \mathcal{C}_c} p_{\cdot j} = q_{\cdot c} \tag{9}$$

for each $i = 1, \ldots, n$ and $c = 1, \ldots, l$. The objective function can be written as follows:

$$\begin{aligned}
\hat{\tau}_{R|C}(\mathcal{R}, \mathcal{C}) &= \sum_{r=1}^{k} \sum_{c=1}^{l} \frac{q_{rc}^2}{q_{\cdot c}} - \sum_{r=1}^{k} q_{r\cdot}^2 \\
&= \sum_{r=1}^{k} \sum_{c=1}^{l} \left(\sum_{i \in \mathcal{R}_r} \frac{p_{ic}}{p_{\cdot c}} \right) q_{rc} - \sum_{r=1}^{k} \left(\sum_{i \in \mathcal{R}_r} p_{i\cdot} \right) q_{r\cdot} \\
&= \sum_{c=1}^{l} \sum_{i=1}^{n} \frac{p_{ic}}{p_{\cdot c}} q_{\mathcal{R}(i)c} - \sum_{i=1}^{n} p_{i\cdot} q_{\mathcal{R}(i)\cdot} \tag{10}
\end{aligned}$$

where $\mathcal{R}(i)$ denotes the cluster assignment of the ith row in the partition $\mathcal{R}$.

Here we propose an iterative strategy to find the best partition $\mathcal{R}$, when $\mathcal{C}$ is fixed. Let us suppose that, at iteration t, a partition $\mathcal{R}^{(t)}$ has been selected and let $q^{(t)}$ be its associated distribution. The objective function $\hat{\tau}_{R|C}(\cdot, \mathcal{C})$ computed for this particular partition takes value

$$\hat{\tau}_{R|C}(\mathcal{R}^{(t)}, \mathcal{C}) = \sum_{i=1}^{n} \left(\sum_{c=1}^{l} \frac{p_{ic}}{p_{\cdot c}} q^{(t)}_{\mathcal{R}^{(t)}(i)c} - p_{i\cdot} q^{(t)}_{\mathcal{R}^{(t)}(i)\cdot} \right) \tag{11}$$

We can interpret each $q_r^{(t)}$ as a prototype of the rth cluster and define a function

$$sim(p_i, q_r^{(t)}) = \sum_{c=1}^{l} \frac{p_{ic}}{p_{\cdot c}} q^{(t)}_{rc} - p_{i\cdot} q^{(t)}_{r\cdot} \tag{12}$$

that measures the similarity between each "point" p_i and each cluster prototype $q_r^{(t)}$.

The objective function can now be written as

$$\hat{\tau}_{R|C}(\mathcal{R}^{(t)}, \mathcal{C}) = \sum_{i=1}^{n} sim\left(p_i, q^{(t)}_{\mathcal{R}^{(t)}(i)} \right) \tag{13}$$

For a fixed row i, if there is a cluster $r \neq \mathcal{R}^{(t)}(i)$ such that $sim(p_i, q_r^{(t)}) > sim(p_i, q^{(t)}_{\mathcal{R}^{(t)}(i)})$, this means that the ith object is more similar to the prototype of cluster r than to the prototype of the cluster it currently belongs to, and then it should be moved from its original cluster $\mathcal{R}^{(t)}(i)$ to cluster r. Thus, we can adopt the following iterative procedure, to identify the best partition $\mathcal{R}$: at each iteration, we compute the cluster prototypes $q^{(t)}$ related to the current partition $\mathcal{R}^{(t)}$ and then assign each object i to the cluster r that maximizes $sim(p_i, q_r^{(t)})$, obtaining a new partition $\mathcal{R}^{(t+1)}$. In case of tie, the object is moved to the cluster r with maximum $q_{r\cdot}^{(t)}$, among those that maximize $sim(p_i, q_r^{(t)})$. In order to decide to which cluster object i should be moved, the objective function $sim(p_i, q_r^{(t)})$ is computed for each cluster $r \in \mathcal{R}^{(t)}$. At each iteration, the algorithm is able to reduce the number of clusters: if all the objects of a cluster r are assigned to other pre-existing clusters, then the number of clusters will decrease by one. Otherwise, if all the objects are moved from one cluster to another but no cluster remains empty, the number of clusters remains the same. It is obvious that the final number of clusters will be lower or equal to the number of clusters contained in the initial partition. Algorithm 1 provides a sketch of the optimization strategy just described.

The same procedure can be applied to find the partition $\mathcal{C}$ of the columns when $\mathcal{R}$ is known, by optimizing the objective function $\hat{\tau}_{C|R}(\mathcal{R}, \cdot)$. We will call the analogous algorithm for column clustering τCC-*columns* $(\mathcal{A}, \mathcal{R}, \mathcal{C}^{(0)})$

Theorem 1 *The algorithm τCC-rows (respectively, the algorithm τCC-columns) increases the objective function $\hat{\tau}_{R|C}(\cdot, \mathcal{C})$ (respectively, $\hat{\tau}_{C|R}(\mathcal{R}, \cdot)$) at each iteration and terminates in a finite number of iterations.*

The proof of the theorem is given in "Appendix A". It exploits the fact that sim is a positive semi-definite bilinear form to demonstrate that $\hat{\tau}_{R|C}$ monotonically increases.

The following toy example illustrates how a single iteration of the algorithm τCC-*rows* works.

Algorithm 1 $\tau CC\text{-}rows(\mathcal{A}, \mathcal{C}, \mathcal{R}^{(0)})$

Input: A matrix $\mathcal{A}$ with shape $n \times m$, the column clustering $\mathcal{C}$, the initial row clustering $\mathcal{R}^{(0)}$

Result: $\mathcal{R}^{(t)}$

Compute p_{ic}, $p_{i\cdot}$ and $p_{\cdot c}$ for $i = 1, \ldots, n, c = 1, \ldots, |\mathcal{C}|$

Initialize $\mathcal{R}^{(0)}$

$t \leftarrow 0$

while $\mathcal{R}^{(t)} \neq \mathcal{R}^{(t-1)}$ **do**

    $k \leftarrow |\mathcal{R}^{(t)}|$

    Assign, to each cluster in $\mathcal{R}^{(t)}$, a different integer r, with $1 \leq r \leq k$;

    Compute $q_{rc}^{(t)}$ and $q_{r\cdot}^{(t)}$, for $r = 1, \ldots, k, c = 1, \ldots, |\mathcal{C}|$ using Equations (6), (7)

    for $i = 1, \ldots, n$ **do**

        $MaxSim \leftarrow max_{1 \leq r \leq k} \left(\sum_{c=1}^{|\mathcal{C}|} \frac{p_{ic}}{p_{\cdot c}} q_{rc}^{(t)} \right) - p_{i\cdot} q_{r\cdot}^{(t)}$

        $\mathcal{I} \leftarrow \left\{ r \mid \left(\sum_{c=1}^{|\mathcal{C}|} \frac{p_{ic}}{p_{\cdot c}} q_{rc}^{(t)} \right) - p_{i\cdot} q_{r\cdot}^{(t)} = MaxSim \right\}$

        $\mathcal{R}^{(t+1)}(i) \leftarrow \underset{r \in \mathcal{I}}{\arg\max}(q_{r\cdot}^{(t)})$

    end

    for $r = 1, \ldots, k$ **do**

        if $\mathcal{R}_r^{(t+1)} = \varnothing$ **then**

            $\mathcal{R}^{(t+1)} \leftarrow \mathcal{R}^{(t+1)} \setminus \mathcal{R}_r^{(t+1)}$

        end

    end

    $t \leftarrow t + 1$

end

Example 3 We want to use Algorithm 1 to cluster the customers of a shop according to their purchase habits. Let

$$\mathcal{A} = \begin{pmatrix} 2 & 3 & 1 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 0 & 5 & 2 \end{pmatrix}$$

be the input matrix, whose rows represent the customers and whose columns are the products. Each entry a_{ij} of $\mathcal{A}$ is the number of items of product j bought by customer i. Imagine that the column clustering $\mathcal{C} = \{\{0, 1, 2\}, \{3, 4, 5\}\}$ is known (for instance, the first three products are foodstuffs, while the remaining three are clothing). Suppose that the row clustering resulting from the last iteration of the algorithm is $\mathcal{R}^{(t)} = \{\{0\}, \{1\}, \{2, 3\}\}$. The probability distribution **P** associated to the data matrix given the column clustering, and the probability distribution $\mathbf{Q}^{(t)}$ of the co-clustering $(\mathcal{R}^{(t)}, \mathcal{C})$ are respectively

$$\mathbf{P} = \begin{pmatrix} 0.23 & 0.00 \\ 0.15 & 0.04 \\ 0.00 & 0.27 \\ 0.04 & 0.27 \end{pmatrix} \qquad \mathbf{Q}^{(t)} = \begin{pmatrix} 0.23 & 0.00 \\ 0.15 & 0.04 \\ 0.04 & 0.54 \end{pmatrix}$$

Algorithm 1 assigns each row of **P** to the cluster represented by the most similar row of $\mathbf{Q}^{(t)}$, according to the similarity measure sim. Since the similarity between each row and each prototype is

	q_0	q_1	q_2
p_0	0.07	0.04	−0.11
p_1	0.04	0.02	−0.06
p_2	−0.06	−0.03	0.09
p_3	−0.05	−0.03	0.08

all the rows remain in their original cluster, except p_1 that is moved in the first cluster. Thus, the resulting row clustering will be $\mathcal{R}^{(t+1)} = \{\{0, 1\}, \{2, 3\}\}$, corresponding to the probability distribution

$$\mathbf{Q}^{(t+1)} = \begin{pmatrix} 0.38 & 0.04 \\ 0.04 & 0.54 \end{pmatrix}.$$

Matrix $\mathbf{Q}^{(t+1)}$ is useful to interpret the results of the co-clustering: the customers in the first cluster are those that mainly buy foodstuffs, while the second cluster contains customers that usually buy clothes.

So far, two different algorithms to optimize the row partition $\mathcal{R}$, given the column partition $\mathcal{C}$ and, viceversa, the column partition $\mathcal{C}$, given the row partition $\mathcal{R}$, have been presented. Theorem 1 proves that each one of the two algorithms converges to a local optimum of its respective objective function ($\hat{\tau}_{R|C}$ or $\hat{\tau}_{C|R}$). A co-clustering algorithm that finds the best partitions $\mathcal{R}$ and $\mathcal{C}$ simultaneously, can be obtained by the iterative repetition of the two algorithms.

Algorithm 2 PB-τCC($\mathcal{A}$, t_{max})

Input: A matrix $\mathcal{A}$ with shape $n \times m$, the maximum number of iterations t_{max}
Result: $\mathcal{R}, \mathcal{C}$
Initialize $\mathcal{R}^{(0)}$ and $\mathcal{C}^{(0)}$
$t \leftarrow 0$
while $(\mathcal{R}^{(t)}, \mathcal{C}^{(t)}) \neq (\mathcal{R}^{(t-1)}, \mathcal{C}^{(t-1)})$ **and** $t < t_{max}$ **do**
    $\mathcal{R}^{(t+1)} \leftarrow \tau\text{CC-}rows(\mathcal{A}, \mathcal{C}^{(t)}, \mathcal{R}^{(t)})$
    $\mathcal{C}^{(t+1)} \leftarrow \tau\text{CC-}columns(\mathcal{A}, \mathcal{R}^{(t+1)}, \mathcal{C}^{(t)})$
    $t \leftarrow t + 1$
end

The overall algorithm, named *Prototype-Based τCC* or PB-τCC, is summarized in Algorithm 2. The row and column clustering $\mathcal{R}^{(0)}$ and $\mathcal{C}^{(0)}$ can be initialized in any possible way. However, since Algorithm 2 can only decrease the number of clusters, an initial co-clustering with a number of clusters that are safely higher than the expected/desired number of clusters is preferable.

The initialization strategy we use consists in the creation of k_0 initial row cluster prototypes and k_0 column cluster prototypes. These initial synthetic prototypes are created as follows. The columns are randomly split in k_0 equal-width clusters. The initial row prototypes are the rows of the identity matrix $I \in \mathbb{R}^{k_0 \times k_0}$. Then each row is assigned to the most similar prototype using the usual similarity measure (Eq. 12). If $sim(p_i, q) < 0$ for any cluster prototype q, then the row p_i is assigned to a new empty cluster, because the prototype of the empty cluster is 0 and $sim(p_i, 0) = 0 > sim(p_i, q)$, for any cluster prototype q. This method, however, requires the specification of the initial number of clusters on the two modes of the matrix. Although this may seem contrary to the spirit of the algorithm, whose strong point is to be parameterless, it turns out that the choice of parameter k_0 does not affect the result to a great extent. This is due to the fact that the initialization method can modify the

number of clusters and add a new cluster when the prototypes of the already existing clusters are not able to describe an object of the data matrix. For a more detailed analysis of this topic, the reader is referred to Sect. 6.3.

4.3 Computational complexity of PB-τCC

Here we compute the computational complexity of the new algorithm PB-τCC and compare it with that of τCC, its direct competitor.

Let $(\mathcal{R}^{(t)}, \mathcal{C}^{(t)})$ be the current co-clustering returned by PB-τCC at iteration t. Consider a row i of $\mathcal{A}$. In order to decide whether i should be moved to another cluster or not, and which cluster it should be moved to, PB-τCC computes $sim(p_i, q_r^{(t)}) = \sum_{c=1}^{l_t} \frac{p_{ic}}{p_{\cdot c}} q_{rc}^{(t)} - p_i . q_{r\cdot}^{(t)}$ in $\mathcal{O}(l_t)$, for all clusters $r \in \mathcal{R}$. Then the re-assignment of i is done in $\mathcal{O}(k_t l_t)$, where k_t is the current number of clusters on the rows and l_t is the current number of clusters on the columns. Similarly, the re-assignment of a column j is done in $\mathcal{O}(k_t l_t)$. Since the number of clusters does not increase from an iteration to the following one, $k_t \leq k_0$ and $l_t \leq l_0$, for each $t \geq 0$. In each iteration of the rows the re-assignment step is repeated n times (once per row), while the columns assignment step is repeated m times (once per column). Therefore, if $\mathcal{I}_R$ is the total number of iterations on rows and $\mathcal{I}_C$ is the total number of iterations on columns, the overall complexity of PB-τCC is in $\mathcal{O}(\mathcal{I}_R n k_0 l_0 + \mathcal{I}_C m k_0 l_0) = \mathcal{O}(\mathcal{I} \max(n, m) k_0 l_0)$, where $\mathcal{I} = \mathcal{I}_R + \mathcal{I}_C$ is the total number of iterations. The initialization of the row clusters takes $\mathcal{O}(nmk_0)$ while the initialization of the column clusters takes $\mathcal{O}(nml_0)$. Thus the initialization phase has complexity in $\mathcal{O}(nm \max(k_0, l_0))$.

Instead, the direct competitor, τCC, as reported by Battaglia and Pensa (2023), has complexity in $\mathcal{O}(\mathcal{I}nm)$, where $\mathcal{I}$ is the total number of iterations (here an iteration is the move of a single object, a row or a column). To allow for a fair comparison between the two algorithms, we should consider that the number $\mathcal{I}$ in PB-τCC is in general significantly lower than the number of iterations $\mathcal{I}$ in τCC, because in τCC $\mathcal{I}$ is the total number of moves of a single object (row or column) that have been done by the algorithm, while the number of moves of a single object in PB-τCC is $\mathcal{M} = \mathcal{I}_R \cdot n + \mathcal{I}_C \cdot m$, with $\mathcal{I}_R + \mathcal{I}_C = \mathcal{I}$. For instance, suppose that the two algorithms make the same number $\mathcal{M}$ of total moves of single objects. Then PB-τCC has complexity in $\mathcal{O}(\mathcal{I}_R n k_0 l_0 + \mathcal{I}_C m k_0 l_0) \simeq \mathcal{O}(\mathcal{M} k_0 l_0)$, while τCC has complexity in $\mathcal{O}(\mathcal{M} nm)$.

In principle, if PB-τCC is initialized with the discrete partitions (where each object forms a singleton cluster), i.e. $k_0 = n$ and $l_0 = m$, the two algorithms have the same complexity $\mathcal{O}(\mathcal{M} nm)$. However, in practice, function sim (Eq. 12) is much faster to compute than the objective function $\Delta\tau$ used by τCC to decide in which a cluster q_r a row p_i currently belonging to cluster q_b should be relocated:

$$\Delta\tau(p_i, q_b, q_r) = \frac{\Gamma \cdot (p_i . (p_i . + q_r . - q_b .)) - \Omega \left(\sum_{c=1}^{l} \frac{p_{ic}}{q_{\cdot c}} (p_{ic} + q_{rc} - q_{bc}) \right)}{\Omega^2 - \Omega \cdot (2 p_i . (p_i . + q_r . - q_b .))}$$

where $\Gamma = 1 - \sum_{ij} \frac{p_{ij}^2}{p_{\cdot j}}$ and $\Omega = 1 - \sum_i p_{i\cdot}^2$. This simplification in the objective function makes our algorithm remarkably more efficient than the other one (as we will show experimentally in Sect. 6).

5 Extension to tensor co-clustering

In this section, we present a higher-order version of algorithm PB-τCC. For sake of clarity, here we introduce a co-clustering algorithm for 3-way tensors, but the same procedure leads to the creation of a co-clustering for tensors with a generic number m of modes.

5.1 Prototype-based tensor co-clustering

Let $\mathcal{A}$ be a tensor with three modes and dimensions (n_X, n_Y, n_Z). Suppose that the partition $\mathcal{Y}$ and $\mathcal{Z}$ on the second and third modes are fixed and we want to find the best partition $\mathcal{X}$ for the first mode. The objective function can be written as

$$\hat{\tau}_{X|(Y,Z)}(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = \sum_{i=1}^{n_X} \left(\sum_{u=1}^{|\mathcal{Y}|} \sum_{v=1}^{|\mathcal{Z}|} \frac{p_{iuv}}{p_{\cdot uv}} q_{\mathcal{X}(i)uv} - p_{i\cdot\cdot} q_{\mathcal{X}(i)\cdot\cdot} \right)$$

where p and q are the probability distributions associated, respectively, to the original data $\mathcal{A}$ and to the co-clustering $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$. They can be computed using the 3-way analogous of Eqs. (5)–(9). We can consider each slice q_w on the first mode of q as a cluster prototype and define a similarity measure analogous to (12) to quantify the similarity between a data point p_i and a cluster prototype q_w:

$$sim(p_i, q_w) = \sum_{u=1}^{|\mathcal{Y}|} \sum_{v=1}^{|\mathcal{Z}|} \frac{p_{iuv}}{p_{\cdot uv}} q_{wuv} - p_{i\cdot\cdot} q_{w\cdot\cdot} \tag{14}$$

Finally, we use this similarity measure to evaluate, at each iteration, the cluster to which each object should be moved. As for the 2-way case (Theorem 1), the iterative repetition of this assignment step monotonically increases the objective function $\hat{\tau}_{X|(Y,Z)}$. Once the best partition on the first mode has been selected and no further moves are possible, we can move to the next mode and repeat the entire process. The procedure just described, which we will call *Prototype-Based τTCC* or PB-τTCC, is the tensor generalization of Algorithm 2.

Algorithm 3 gives a sketch of PB-τTCC for generic m-way tensors. Here, $\mathcal{P}^1, \ldots, \mathcal{P}^m$ are the m partitions (one for each node). Hence, in the 3-way case, $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ corresponds to $(\mathcal{P}^1, \mathcal{P}^2, \mathcal{P}^3)$. Algorithm PB-$\tau$TCC only looks at the objective function related to the mode on which the re-assignment is performed: for instance, in a 3-way tensor, when moving rows only the objective function $\hat{\tau}_{X|(Y,Z)}$ is optimized, while $\hat{\tau}_{Y|(X,Z)}$ and $\hat{\tau}_{Z|(X,Y)}$ are not considered. As for the 2-way case, the partitions on all the modes of the tensor can be initialized in any possible way. However, since the algorithm can only decrease the number of clusters, initial co-clustering configurations with a number of clusters that are safely higher than the number of expected/desired clusters are preferable.

Although the initialization strategy proposed for matrix co-clustering can be used for tensor co-clustering too, the probability of selecting poor initial prototypes would increase with the number of modes (and with the sparsity) of the tensor. Hence, for tensor co-clustering, we adopt a strategy inspired by the classical initialization of k-means clustering and consisting in the random extraction of k_0, l_0 and h_0 elements from each of the three modes of $\mathcal{A}$ (in the case of 3-way tensors). These randomly selected elements are considered as prototypes of k_0, l_0, h_0 clusters in each mode, and each elements of tensor $\mathcal{A}$ is assigned to the most similar prototype using the usual similarity measure (12). If $sim(p_i, q) < 0$ for any cluster prototype q, then the row (column) p_i is assigned to a new empty cluster, since the prototype of the empty cluster is 0 and $sim(p_i, 0) = 0 > sim(p_i, q)$, for any cluster prototype q.

Algorithm 3 PB-τTCC($\mathcal{A}$, t_{max})

Input: A m-way tensor $\mathcal{A}$ with shape $n_1 \times \cdots \times n_m$, the maximum number of iterations t_{max}
Result: $(\mathcal{P}^1, \ldots, \mathcal{P}^m)$
  Initialize $\mathcal{P}^{(0)} = (\mathcal{P}^{1(0)}, \ldots, \mathcal{P}^{m(0)})$
$t \leftarrow 0$
while $\mathcal{P}^{(t)} \neq \mathcal{P}^{(t+1)}$ **and** $t < t_{max}$ **do**
    Compute tensor **P** using the n-way equivalent of p_{ic} (Equation 9)
    for $i = 1, \ldots, m$ **do**
        $\mathcal{X}^0 \leftarrow \mathcal{P}^{i(t)}$
        $h \leftarrow 0$
        while $\mathcal{X}^{(h)} \neq \mathcal{X}^{(h+1)}$ **do**
            Compute tensor **Q** using the equivalent of q_{rc} (Equation 6)
            $\mathcal{X}^{(h+1)} \leftarrow \hat{\tau}\texttt{CC-one\_mode}(\textbf{P}, \textbf{Q}, i)$
            $h \leftarrow h + 1$
        end
    end
    $t \leftarrow t + 1$
end
Function $\hat{\tau}\texttt{CC-one\_mode}(\textbf{P}, \textbf{Q}, i)$:
  $\textbf{P} \leftarrow$ transpose($\textbf{P}$,i); // rotate the tensor to move mode i on the first position
  $\textbf{Q} \leftarrow$ transpose($\textbf{Q}$,i); // analogous rotation
  $k \leftarrow$ # of rows in $\textbf{Q}$
  for $j = 1, \ldots, n_i$ **do**
    $MaxSim \leftarrow \max_{1 \leq r \leq k} sim(p_j, q_r)$
    $\mathcal{I} \leftarrow \{r | sim(p_j, q_r) = MaxSim\}$
    $\mathcal{P}^i(j) \leftarrow \arg\max_{r \in \mathcal{I}}(q_r)$
  end
  Remove eventual empty clusters
  return $\mathcal{P}^i$

Notice that, in the initialization phase the case $sim(p_i, q) < 0$ for all q is possible: although, the proof of Theorem 1 (see "Appendix A") is based on the assumption that the sum of all the prototypes coincides with the sum of all the rows in the matrix, this is assumption does not hold in this case.

5.2 Computational complexity of PB-τTCC

For the tensor co-clustering algorithm PB-τTCC, the computational complexity can be derived reasoning as in the 2-way case (see Sect. 4.3). For the sake of simplicity, here we report the overall complexity for a 3-way input tensor, but the computation can be easily extended to the generic n-way case. We recall that PB-τTCC (see Algorithm 3) works similarly as PB-τCC, i.e., it moves one item in one partition while keeping the other two partitions fixed. Hence, by generalizing the reasoning done in Sect. 4.3 to the 3-way case, we can easily conclude that the overall complexity of PB-τTCC is in $\mathcal{O}((\mathcal{I}_X n_X + \mathcal{I}_Y n_Y + \mathcal{I}_Z n_Z) \cdot k_0 l_0 h_0) \simeq \mathcal{O}(\mathcal{I} \max(n_X, n_Y, n_Z) k_0 l_0 h_0)$, where $\mathcal{I} = \mathcal{I}_X + \mathcal{I}_Y + \mathcal{I}_Z$ is the total number of iterations of the algorithm and k_0, l_0 and h_0 are the number of clusters of the initial partition on, respectively, the first, second and third mode of the tensor. According to Battaglia and Pensa (2023), τTCC (the direct competitor of our algorithm) has complexity in $\mathcal{O}(\mathcal{I} n_X n_Y n_Z)$. As for the 2-way case, it is worth noting that the number of iterations $\mathcal{I}$ in PB-τTCC has a different meaning than in τTCC, as in τTCC at each iteration only one object on one mode of the tensor is re-assigned, while in PB-τTCC all the objects on the same mode are re-assigned at

each iteration. Thus, the total number of re-assignments of a single object is $\mathcal{I}$ in τTCC and is $\mathcal{I}_X n_X + \mathcal{I}_Y n_Y + \mathcal{I}_Z n_Z$ in PB-τTCC. Now, assuming that the two algorithms make the same number $\mathcal{M}$ of total moves of single objects, then PB-τTCC has complexity in $\mathcal{O}(\mathcal{M}k_0 l_0 h_0)$, while τCC has complexity in $\mathcal{O}(\mathcal{M}n_X n_Y n_Z)$. Again, if PB-$\tau$TCC is initialized with the discrete partitions, the two algorithms have the same complexity $\mathcal{O}(\mathcal{M}n_X n_Y n_Z)$: in fact, the computational complexity of a single re-assignment step is, for both algorithms, $\mathcal{O}(k_t l_t h_t)$, where k_t, l_t and h_t are the number of clusters on the three modes at iteration t. However, in practice, the function sim (Eq. 12) is much faster to compute than the objective function $\Delta\tau$ used by τTCC to decide in which a cluster q_r a row p_i currently belonging to cluster q_b should be relocated. In fact, Battaglia and Pensa (2023) define $\Delta\tau$ as:

$$\Delta\tau(p_i, q_b, q_r) = \frac{\Gamma \cdot (p_{i\cdot\cdot}(p_{i\cdot\cdot} + q_{r\cdot\cdot} - q_{b\cdot\cdot})) - \Omega\left(\sum_{u=1}^{|\mathcal{Y}|} \sum_{v=1}^{|\mathcal{Z}|} \frac{p_{iuv}}{q_{\cdot uv}}(p_{iuv} + q_{ruv} - q_{buv})\right)}{\Omega^2 - \Omega \cdot (2p_{i\cdot\cdot}(p_{i\cdot\cdot} + q_{r\cdot\cdot} - q_{b\cdot\cdot}))}$$

where $\Gamma = 1 - \sum_{ijk} \frac{p_{ijk}^2}{p_{\cdot jk}}$ and $\Omega = 1 - \sum_i p_{i\cdot\cdot}^2$. In Sect. 6, we will show the practical gain, in terms of computational time, introduced by our new algorithm.

6 Experiments

In this section, we present and discuss some experimental results that show the effectiveness of the proposed method, PB-τTCC, compared to other state-of the-art algorithms for matrix and tensor co-clustering.

We choose the competitors by considering some fundamental co-clustering algorithms as well as some of the most recent and effective approaches of the state of the art in matrix and tensor co-clustering. The state-of-the-art 2-way co-clustering methods considered here are: ITCC (Dhillon et al., 2003), ModCC (Ailem et al., 2016), PLBM (Govaert and Nadif, 2010), SpectralCC (Dhillon, 2001), SpectralBiC (Kluger et al., 2003), DeepCC (Xu et al., 2019), NMTF (Non-Negative Matrix Tri-Factorization followed by k-means), TLBM (Boutalbi et al., 2022), and τCC (Ienco et al., 2013). For the first three algorithms, we use the python implementation included in package CoClust.[1] For the two spectral methods we use their Scikit-learn implementations.[2] For TLBM we use the python library included in package TensorClus[3] using the Poisson distribution. For DeepCC and τCC, we use the code publicly provided by the authors.[4,5] Finally, for NMTF we use our own python implementation.

For tensor co-clustering, the baseline methods are: nnCP (non-negative CP decomposition), nnT (non-negative Tucker decomposition), SparseCP (Papalexakis et al., 2013), TBM (Wang and Zeng, 2019), TLBM (Boutalbi et al., 2022), and τTCC (Battaglia and Pensa, 2023). We use the TensorLy implementations for nnT and nnCP.[6] For TLBM we use the python implementation provided in package TensorClus[7] using the Bernoulli distribution,

---

[1] https://coclust.readthedocs.io/en/v0.2.1/.

[2] https://scikit-learn.org/stable/.

[3] https://pypi.org/project/TensorClus/.

[4] https://github.com/DerronXu/Deep-Co-Clustering.

[5] https://github.com/elenabattaglia/tensor_cc.

[6] http://tensorly.org/stable/index.html.

[7] https://pypi.org/project/TensorClus/.

Table 2 Dataset characteristics

Dataset	# Modes	# Rows	# Columns	# Tubes	# Classes
classic3	2	3891	4303	–	3
cstr	2	475	1000	–	4
tr11	2	414	6429	–	9
tr41	2	878	7454	–	10
hitech	2	2301	10,080	–	6
k1b	2	2340	10,431	–	6
reviews	2	4069	18,483	–	5
sports	2	8280	14,870	–	7
DBLP	3	6044	4057	20	4
MovieLens1	3	181	215	142	3
MovieLens2	3	145	74	115	3
YelpTOR	3	178	628	458	3
YelpPGH	3	95	237	544	3

while for the remaining three methods we use the code provided by the authors, in Matlab,[8] R,[9] and Python,[10] respectively.

In PB-τTCC, the number k_0 of initial prototypes is always set equal to 30 (a reasonable overestimation of the true number of clusters). All other methods, with the exception of τCC and τTCC, require as input the number of clusters on each mode: we set these values equal to the correct number of clusters (on the mode of the tensor for which class labels are given), and set the same number of clusters on the other modes of the tensor as well. In the matrix/tensor factorization-based methods, we tried different ranks and retain the value that gives the best clustering result. Additional parameters, when needed, are set as indicated in the original papers.[11]

The evaluation of the performances, in terms of quality of the partitioning, is done by applying the co-clustering methods on real world datasets: we run the algorithms on 8 document-words co-occurrences matrices from the CLUTO project.[12] For tensor co-clustering, instead, the datasets are the same 3-way tensors used in (Battaglia and Pensa, 2023) to assess the quality of τTCC: DBLP,[13] two tensors extracted from MovieLens[14] and two extracted from Yelp dataset.[15] Table 2 summarizes the main characteristics of these datasets. We assess the quality of the clustering solution through the Normalized Mutual Information, a measure commonly used in the clustering literature. Each algorithm is applied 30 times and the average results in terms of NMI are reported in Table 3. All experiments are executed on a server with 32 Intel Xeon Skylake cores running at 2.1 GHz, 256 GB RAM, and one Tesla T4 GPU.

---

[8] https://www.cs.ucr.edu/~epapalex/code.html.

[9] https://cran.r-project.org/web/packages/tensorsparse/.

[10] https://github.com/elenabattaglia/tensor_cc.

[11] The code and datasets used for the experiments are available at https://github.com/elenabattaglia/Fast-TauCC.

[12] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/overview.

[13] http://web.cs.ucla.edu/~yzsun/data/DBLP_four_area.zip.

[14] https://grouplens.org/datasets/hetrec-2011/.

[15] https://www.yelp.com/dataset.

Table 3 Results of matrix co-clustering algorithms on real-world data, in terms of NMI computed for the labeled mode (rows)

Method	classic3	cstr	tr11	tr41	hitech	k1b	reviews	sports
ITCC	0.84 ± 0.15	0.61 ± 0.07	**0.50 ± 0.04**	**0.55 ± 0.02**	0.25 ± 0.01	0.53 ± 0.03	0.51 ± 0.07	0.51 ± 0.06
CCMod	0.90 ± 0.06	0.58 ± 0.10	0.45 ± 0.06	0.48 ± 0.05	0.22 ± 0.03	0.51 ± 0.06	0.46 ± 0.07	0.50 ± 0.05
SpecCC	0.91 ± 0.00	0.71 ± 0.01	0.43 ± 0.01	**0.55 ± 0.00**	0.23 ± 0.00	**0.66 ± 0.00**	0.41 ± 0.00	0.41 ± 0.00
SpecBiC	0.88 ± 0.00	0.25 ± 0.12	0.08 ± 0.01	0.24 ± 0.04	0.02 ± 0.00	0.16 ± 0.09	0.13 ± 0.01	0.05 ± 0.02
PLBM	0.82 ± 0.14	0.62 ± 0.09	0.49 ± 0.05	0.51 ± 0.04	0.23 ± 0.01	0.56 ± 0.04	0.52 ± 0.08	0.51 ± 0.07
DeepCC	0.89 ± 0.13	0.63 ± 0.06	0.07 ± 0.04	0.26 ± 0.06	0.13 ± 0.02	0.43 ± 0.06	0.31 ± 0.11	0.29 ± 0.10
NMTF	0.83 ± 0.00	0.67 ± 0.01	0.15 ± 0.00	0.42 ± 0.01	0.22 ± 0.01	0.52 ± 0.00	0.43 ± 0.00	0.52 ± 0.00
TLBM	0.56 ± 0.33	0.52 ± 0.13	0.37 ± 0.08	0.47 ± 0.07	0.20 ± 0.04	0.50 ± 0.11	0.39 ± 0.06	0.47 ± 0.07
τCC	**0.93 ± 0.00**	**0.75 ± 0.02**	0.29 ± 0.01	0.51 ± 0.08	0.26 ± 0.01	0.57 ± 0.00	0.29 ± 0.00	0.38 ± 0.00
PB-τCC	0.90 ± 0.02	**0.75 ± 0.02**	**0.50 ± 0.05**	0.49 ± 0.05	**0.31 ± 0.02**	**0.69 ± 0.03**	**0.53 ± 0.02**	**0.59 ± 0.04**

The best results are given in bold

Table 4 Results of tensor co-clustering algorithms on real-world data, in terms of NMI computed for the labeled mode (rows)

Method	DBLP	MovieLens1	MovieLens2	YelpTOR	YelpPGH
nnCP	0.78 ± 0.00	0.38 ± 0.00	0.11 ± 0.00	0.42 ± 0.00	0.10 ± 0.00
nnT	0.78 ± 0.00	0.42 ± 0.00	0.24 ± 0.00	$\mathbf{0.43 \pm 0.00}$	0.27 ± 0.00
SparseCP	0.00 ± 0.00	0.05 ± 0.00	0.10 ± 0.00	0.20 ± 0.00	0.11 ± 0.00
TBM	-	0.30 ± 0.00	0.10 ± 0.00	0.02 ± 0.00	0.11 ± 0.00
LBM	0.49 ± 0.08	0.18 ± 0.10	0.10 ± 0.06	0.12 ± 0.04	0.05 ± 0.04
τTCC	0.71 ± 0.00	$\mathbf{0.58 \pm 0.12}$	0.26 ± 0.09	0.36 ± 0.04	$\mathbf{0.35 \pm 0.02}$
PB-τTCC	$\mathbf{0.79 \pm 0.03}$	0.51 ± 0.03	$\mathbf{0.28 \pm 0.03}$	0.33 ± 0.01	0.31 ± 0.01

The best results are given in bold

6.1 Results of PB-τCC on real-world matrices and tensors

On matrices, our algorithm outperforms all the competing approaches for most datasets (see Table 3). When its NMI is not the highest one, it still ranks among the best algorithms. More in detail, PB-τCC outperforms its direct parameterless competitor τCC for all datasets but *classic3* and *tr41*, on which the two algorithms achieve similar results. Interestingly, the number of clusters identified by PB-τCC is usually more accurate than that of τCC (see Table 5, where we report the average number of clusters founds and its variation for the main mode, as it is the only one whose we dispose of the ground truth and/or some background knowledge). As expected, the latter tends to aggregate the rows excessively. Instead, PB-τCC finds a more reasonable number of clusters, although, sometimes, the final number of detected clusters can be slightly smaller than the number of the given classes. Despite this fact, as observed in Table 3, the quality of the clustering remains high. Table 5 also reports the execution time of the two parameterless approaches and clearly shows that our method is sensibly faster than τCC, by two orders of magnitude, on average. Additionally, in Fig. 2 we report the visualization of the co-clustering (with rows and columns rearranged according to their cluster assignment) computed by PB-τCC on all datasets.

On tensors, the performances of PB-τTCC are almost always better than those of the methods requiring the number of clusters as input parameters. However, the accuracy in terms of NMI is slightly lower than that of τTCC (see Table 4). This is probably a consequence of the fact that PB-τTCC, in some cases, tends to identify an higher number of smaller clusters. However, the marked difference in the execution times between PB-τTCC and τTCC, reported in Table 5, makes the new algorithm much more suitable to handle tensors of high-dimensionality. An example of visualization of the co-clustering computed on DBLP is given in Fig. 3, where we report three representative slices whose rows and columns are rearranged according to the clustering results.
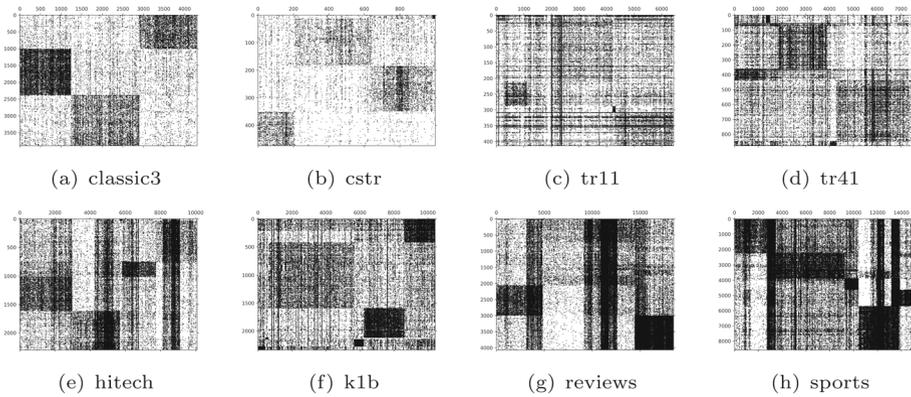
Fig. 2 Visualization of the results of PB-τCC on matrices. Rows and columns are rearranged according to their cluster membership. The dark points represent nonzeros in the original matrix
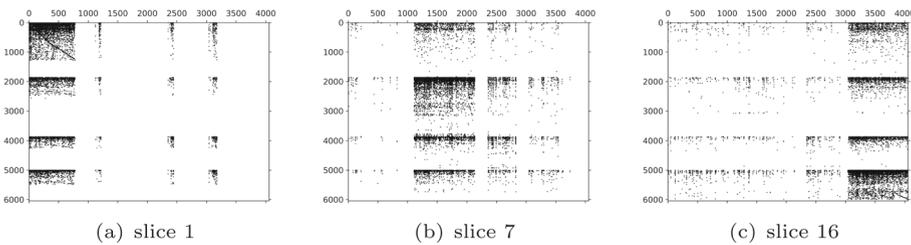
Fig. 3 Visualization of the results of PB-τTCC on DBLP for three slices. Rows and columns are rearranged according to their cluster membership. The dark points represent nonzeros in the original matrix

Table 5 Execution time (in seconds) of τCC and PB-τCC and median number of identified clusters ± IQR (interquartile range)

Dataset	Execution time		# clusters		
	τTCC	PB-τTCC	τTCC	PB-τTCC	# classes
classic3	1708.97	9.22	3 ± 0	3 ± 0	3
cstr	104.92	1.54	4 ± 0	4 ± 1	4
tr11	689.58	16.08	2 ± 0	5 ± 1	10
tr41	1490.72	12.54	5 ± 0	5 ± 2	9
hitech	3736.17	46.83	2 ± 1	5 ± 1	6
k1b	3473.53	37.76	2 ± 0	6 ± 1	6
reviews	13238.48	81.18	3 ± 1	4 ± 2	5
sports	22703.07	202.82	2 ± 0	5 ± 1	7
DBLP	$> 100{,}000$	937.73	5 ± 1	4 ± 0	4
MovieLens1	459.76	8.55	5 ± 1	9 ± 3	3
MovieLens2	297.83	1.77	3 ± 3	7 ± 1	3
YelpTOR	15678.78	41.20	5 ± 1	9 ± 2	3
YelpPGH	397.03	7.55	13 ± 2	15 ± 2	3

(a) Average execution times on matrices

(b) Average execution times on 3-way tensors
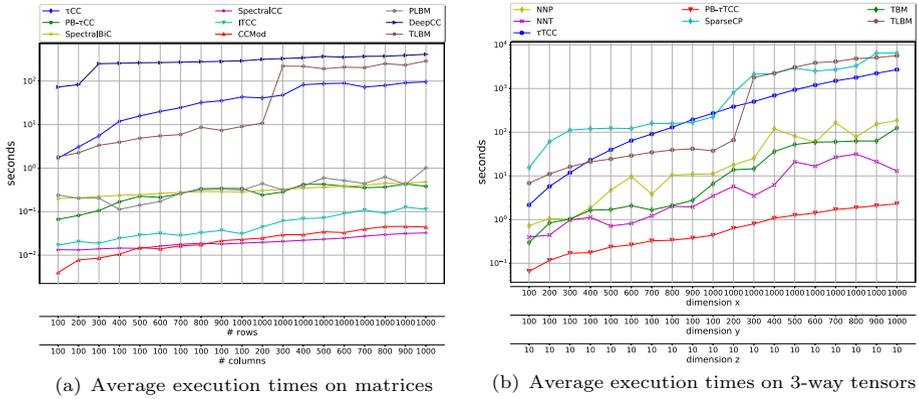
Fig. 4 Average execution times on matrices (**a**) and 3-way tensors (**b**), varying the shape of the dataset

(a)   (b)
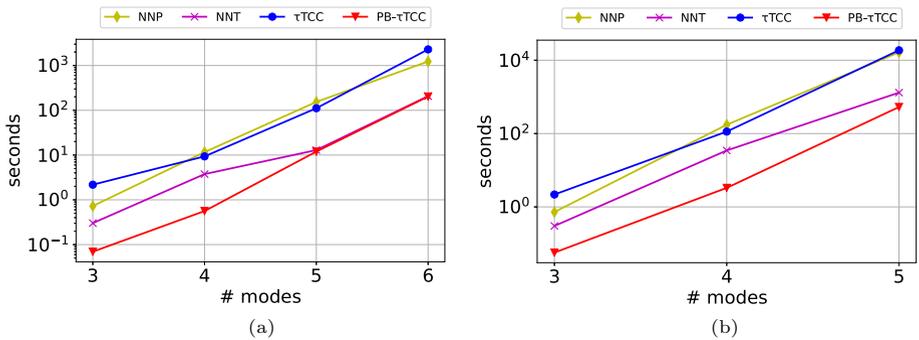
Fig. 5 Average execution times on n-way tensors, stating with a $100 \times 100 \times 10$ tensor and adding a mode of dimension 10 (**a**) or 100 (**b**) each time

6.2 Execution time on synthetic data

To better investigate the difference in the execution times of the various methods, we run the different algorithms on synthetic datasets of controlled size. We construct binary matrices with three embedded block co-clusters. For matrix co-clustering, we let the shape of the matrix vary from 100×100 to 1000×1000 by increasing the number of rows and columns. Figure 4a shows the performances expressed in seconds for this experiment. Instead, Fig. 4b shows the results of a similar experiment conducted on 3-way tensors: here the shape of the tensors varies from $100 \times 100 \times 10$ to $1000 \times 1000 \times 10$. In all these datasets, 20% of the matrix/tensor entries are randomly selected and their binary value is swapped (from 0 to 1, and viceversa).

The results on matrices show that, as expected, PB-τCC is significantly faster than τCC, but slower than other competitors that requires the number of clusters as input parameter (ITCC, CCMod and SpecCC). Not surprisingly, algorithm DeepCC is the slowest algorithm among the ones selected for our experiments. Instead, when 3-way tensor co-clustering is considered (Fig. 4b), PB-τTCC outperforms by far all the other competing approaches in terms of execution time.
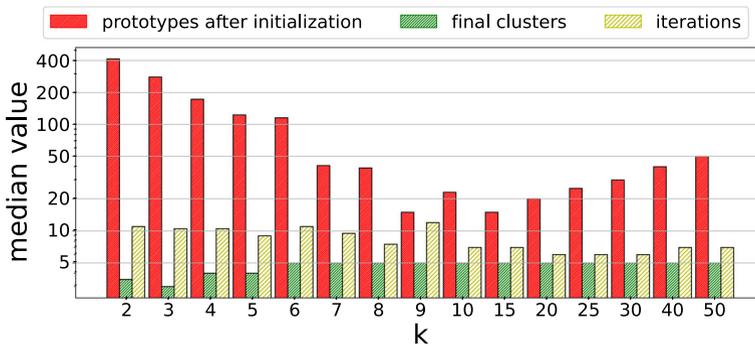
Fig. 6 Impact of the choice of k on algorithm PB-τCC. The figure shows the variation in the number of initial row prototypes, of the number of final row clusters, and of the number of iterations w.r.t k. The synthetic data matrix has shape 1000×1000 and 5 row clusters

Finally, we also investigate the impact of the number of modes on the execution times. We start with a 3-way tensor with shape $100 \times 100 \times 10$ and we add a new mode of dimension 10 (Fig. 5a) or of dimension 100 (Fig. 5b) at each time. In these experiments, we do not include algorithms SparseCP, TBM and TLBM, because the available implementations only handle 3-way tensors. The two plots show that PB-τTCC is the fastest algorithm, among the ones considered in this study; however, the difference in the execution times between our approach and the competing methods (nnT in particular), slightly decreases as the number of modes grows.

6.3 Choice of the initial number of prototypes

The initialization strategy we propose creates k synthetic row prototypes and l synthetic columns prototypes, where k and l can be considered as input parameters. Here we show that, in practice, they do not have any sensible impact on the results, provided that they are sufficiently high. Let K be the true number of row clusters embedded in the data. Intuitively, if $k < K$ there will be many rows that are not sufficiently similar to any extracted prototype and they will be put in singleton clusters. On the other hand, when k is very high, there will be a high number of singleton clusters as well. A good choice for parameter k is a number that is higher than the number of expected clusters embedded in the data but also significantly smaller than the number of rows of the matrix. For instance, if matrix $\mathcal{A}$ has 1000 rows and we would like to obtain a partition of the data in few clusters, a good choice for the initial number of clusters could be $10 \leq k \leq 50$. Notice that, once we have decided the magnitude of the parameter, the exact value of k is not very important, as shown in Fig. 6, which provides the impact of the choice of the initial parameter k on the number of final clusters and execution times on a synthetic 1000×1000 matrix with 5 embedded co-clusters.

As we said at the beginning of this section, in all the experiments, the number of clusters is always set equal to 30, but, as a rule of thumb, $k = \max\{10, n/20\}$, where n is the number of rows. The same rule can be applied for the initial number of column clusters as well. The rationale behind this rule is that, in typical data analysis tasks, when the number of elements in one mode (e.g, the rows in matrix) is less then 200, one does not expect the final number of clusters to be much more than five. For larger datasets, instead, the rule suggests setting the number of initial prototypes in such a way that, on average, each cluster initially contains 20 elements, which we consider a rather conservative choice. In practice, all these

parameters can be automatically determined from the dimensionality of the input data, with little or no impact on the final results. Of course, it could happen that, by applying the rule, the initial number of prototypes is underestimated. In this case, PB-τCC may end up with an underclustered solution. This is a limitation of our algorithm, which, by construction, can only decrease the number of clusters during the optimization process. As future work, we plan to explore the possibility of dynamically adjusting the number of clusters during the execution of the iterative steps of the algorithm, although we suspect that this could be detrimental to the good convergence properties shown by PB-τCC.

As a final remark, to be fair, we must say that, for some of the competitors, the authors also provide some criterion to identify the "best" number of clusters [this is done, for instance, by Ailem et al. (2016)]. However, the related methods always involve the repeated execution of the original algorithm, which can be unfeasible on very high-dimensional data, especially if the number of input parameters is also high.

7 Conclusion

We have introduced a prototype-based co-clustering approach optimizing a simplified version of the Goodman–Kruskal's τ association measure, an objective function that is minimally affected by the choice of the number of clusters, a typical input parameter of any (co-)clustering algorithm. Differently from the previous attempts of using this measure to perform parameterless co-clustering on matrices and tensors, our algorithm is much faster, while at least preserving, if not improving, the quality of the extracted clusters. When compared to state-of-the-art co-clustering algorithms that require the number of clusters as input parameter, our algorithm also shows its effectiveness and feasibility. On tensor data, it is even faster than any other competitor considered in this study.

A limitation of our approach (as well as of all other algorithms optimizing the same family of objective functions) is that it is not well suited for dense numeric matrices (e.g., gene expression data), because, in this case, the association captured by the τ measure could be meaningless. As future work, we will investigate ways to exploit the good properties of the new version of τ in such more challenging contexts. Furthermore, we will investigate more in detail the suspected intrinsic ability of identifying outliers in data, which could make the overall approach even more robust.

Availability of data and material All data are available online and accessible to everyone.

Code availability Source code and scripts used in our experiments are available at https://github.com/elenabattaglia/Fast-TauCC.

Declarations

Appendix A Proof of Theorem 1

Before proving the theorem, we need to introduce some important lemmas. The first one states that sim (Eq. 12) is a positive semi-definite symmetric bilinear form.

Lemma 1 *Let $x = (x_j)_{j=1}^n \in \mathbb{R}^n$ be a vector such that $x_j > 0$ for each j and $\sum_{j=1}^n x_j = 1$. The function*

$$sim_x : (u, v) \longmapsto \sum_{j=1}^n \frac{u_j \cdot v_j}{x_j} - u_\cdot \cdot v_\cdot \tag{A1}$$

is a positive semi-definite symmetric bilinear form.

Proof The symmetry and the linearity of sim_x in both the variables follow straightforwardly from its definition. It still remains to prove that $sim(u, u) \geq 0$ for each $u \in \mathbb{R}^n$. Consider vectors $a, b \in \mathbb{R}^n$ defined as $a_j = \frac{u_j}{\sqrt{x_j}}$ and $b_j = \sqrt{x_j}$. Then

$$0 \leq \left(\sum_{j=1}^n a_j^2\right)\left(\sum_{j=1}^n b_j^2\right) - \left(\sum_{j=1}^n a_j b_j\right)^2 =$$

$$= \left(\sum_{j=1}^n \frac{u_j^2}{x_j}\right)\left(\sum_{j=1}^n x_j\right) - \left(\sum_{j=1}^n \frac{u_j}{\sqrt{x_j}}\sqrt{x_j}\right)^2 =$$

$$= \sum_{j=1}^n \frac{u_j^2}{x_j} - \left(\sum_{j=1}^n u_j\right)^2 =$$

$$= \sum_{j=1}^n \frac{u_j^2}{x_j} - u_\cdot^2 = sim_x(u, u)$$

where the first inequality is the Cauchy-Schwarz inequality and the second equality follows from $\sum_j x_j = 1$. $\qquad\square$

The function sim through which the algorithm $\tau CC\text{-}rows$ evaluates the similarity between each row of the matrix and each cluster prototype is a special case of the function family sim_x, with $x = (p_{\cdot c})_{c=1}^l$, because $\sum_{c=1}^l p_{\cdot c} = 1$.

Lemma 2 *Let C be the fixed partition of the columns of a matrix A. Let $\mathcal{R}^{(t)}$ be the partition found by the algorithm τCC-rows at the tth iteration and let $q^{(t)}$ be its corresponding distribution. Then, the function $\hat{\tau}_{R|C}$ can be written as*

$$\hat{\tau}_{R|C}(\mathcal{R}^{(t)}, C) = \sum_{r=1}^{|\mathcal{R}^{(t)}|} sim(q_r^{(t)}, q_r^{(t)}).$$

Lemma 3 *Partition $\mathcal{R}^{(t+1)}$ selected by the algorithm τCC-rows at the following iteration is the partition that maximizes*

$$f_C(\mathcal{R}^{(t)}, \mathcal{R}^{(t+1)}) = \sum_{r=1}^{|\mathcal{R}^{(t)}|} sim(q_r^{(t)}, q_r^{(t+1)})$$

where $q^{(t+1)}$ is the distribution associated to $\mathcal{R}^{(t+1)}$.

For the sake of brevity, we omit the simple proofs of the two lemmas, that exploit the linearity and symmetry of sim to rearrange the sums in the definition of $\hat{\tau}_{R|C}$. We can finally prove a major result of our paper.

Proof of Theorem 1 To increase readability, we use the following simplified notation: $\hat{\tau}(t) = \hat{\tau}_{R|C}(\mathcal{R}^{(t)}, C)$ and $f(t) = f_C(\mathcal{R}^{(t)}, \mathcal{R}^{(t+1)})$. We want to prove that the algorithm τCC-*rows* monotonically increases the function $\hat{\tau}(t)$ and that $\hat{\tau}(t) = \hat{\tau}(t+1)$ implies either $\mathcal{R}^{(t)} = \mathcal{R}^{(t+1)}$ or $B(\mathcal{R}^{(t)}) < B(\mathcal{R}^{(t+1)})$, where $B(R^{(t)}) = \sum_{i=1}^{n} p_i \cdot q_{R^{(t)}(i)}^{(t)}$. This proves the Theorem, since $\hat{\tau}$ always goes through different co-clustering solutions, and the number of distinct co-clustering solutions is finite. We will show that, for each t,

$$\hat{\tau}(t) \leq f(t) \leq \hat{\tau}(t+1)$$

The first inequality directly follows from Lemmas 2 and 3, because

$$\hat{\tau}(t) = \sum_r sim\left(q_r^{(t)}, q_r^{(t)}\right) \leq \sum_r sim\left(q_r^{(t)}, q_r^{(t+1)}\right) = f(t).$$

Furthermore, moving from partition $\mathcal{R}^{(t)}$ to partition $\mathcal{R}^{(t+1)}$, for each cluster there will be some rows that remain in the same cluster, some rows that leave the original cluster to another and some rows that enter in the cluster from another one, i.e. for each r

$$q_r^{(t+1)} = q_r^{(t)} - q_r^{\mathbf{out}} + q_r^{\mathbf{in}} \tag{A2}$$

where $q_r^{\mathbf{out}}$ is the sum of all the points p_i such that $\mathcal{R}^{(t)}(i) = r$ and $\mathcal{R}^{(t+1)}(i) \neq r$ and $q_r^{\mathbf{in}}$ is the sum of the points p_i such that $\mathcal{R}^{(t)}(i) \neq r$ and $\mathcal{R}^{(t+1)}(i) = r$. Thus,

$$\hat{\tau}(t) \leq f(t) = \sum_r sim\left(q_r^{(t)}, q^{(t)} - q_r^{\mathbf{out}} + q_r^{\mathbf{in}}\right) =$$

$$= \sum_r sim\left(q_r^{(t)}, q_r^{(t)}\right) + \sum_r sim\left(q_r^{(t)}, q_r^{\mathbf{in}} - q_r^{\mathbf{out}}\right) =$$

$$= \hat{\tau}(t) + \sum_r sim\left(q_r^{(t)}, q_r^{\mathbf{in}} - q_r^{\mathbf{out}}\right)$$

which implies

$$\sum_r sim\left(q_r^{(t)}, q_r^{\mathbf{in}} - q_r^{\mathbf{out}}\right) \geq 0. \tag{A3}$$

Now, if we assume *ad absurdum* that $\hat{\tau}(t+1) < f(t)$, it would be

$$\hat{\tau}(t+1) < f(t) = \sum_r sim\left(q_r^{(t)} + q_r^{\mathbf{out}} - q_r^{\mathbf{in}}, q_r^{(t+1)}\right) =$$

$$= \hat{\tau}(t+1) + \sum_r sim\left(q_r^{\mathbf{out}} - q_r^{\mathbf{in}}, q_r^{(t+1)}\right)$$

where the first equality follows from (A2). Then

$$\sum_r sim\left(q_r^{\mathbf{out}} - q_r^{\mathbf{in}}, q_r^{(t+1)}\right) > 0. \qquad (A4)$$

By putting together (A3) and (A4) and exploiting the symmetry and linearity of sim, we have that

$$0 < \sum_r sim\left(q_r^{(t)}, q_r^{\mathbf{in}} - q_r^{\mathbf{out}}\right) + \sum_r sim\left(q_r^{\mathbf{out}} - q_r^{\mathbf{in}}, q_r^{(t+1)}\right)$$

$$= -\sum_r sim\left(q_r^{\mathbf{in}} - q_r^{\mathbf{out}}, q_r^{\mathbf{in}} - q_r^{\mathbf{out}}\right)$$

which is impossible because $sim(v, v) \geq 0$ for all $v \in \mathbb{R}^l$.

For the second part of the theorem, consider $\mathcal{R}^{(t)} \neq \mathcal{R}^{(t+1)}$ with $\hat{\tau}(t) = \hat{\tau}(t+1)$. Then, it must be

$$\sum_{i=1}^n sim\left(p_i, q_{R^{(t)}(i)}^{(t)}\right) = \hat{\tau}(t) = f(t) = \sum_{i=1}^n sim\left(p_i, q_{R^{(t+1)}(i)}^{(t)}\right)$$

Moreover, $sim\left(p_i, q_{R^{(t)}(i)}^{(t)}\right) \leq sim\left(p_i, q_{R^{(t+1)}(i)}^{(t)}\right)$ for all i, by definition of $\mathcal{R}^{(t+1)}$. It follows that $sim\left(p_i, q_{R^{(t)}(i)}^{(t)}\right) = sim\left(p_i, q_{R^{(t+1)}(i)}^{(t)}\right)$ for all i and, since the re-assignment step resolves ties in favor of the cluster with the higher $q_{r\cdot}$, it must be $q_{R^{(t)}(i)\cdot}^{(t)} \leq q_{R^{(t+1)}(i)\cdot}^{(t)}$, with at least one strict inequality. Then

$$B(t) = B(\mathcal{R}^{(t)}) = \sum_{i=1}^n p_i \cdot q_{R^{(t)}(i)\cdot}^{(t)} < \sum_{i=1}^n p_i \cdot q_{R^{(t+1)}(i)\cdot}^{(t)} = g(t)$$

To conclude the proof, it is enough to show that $g(t) \leq B(t+1)$. Once noticed that $B(t)$ can be written as $B(t) = \sum_r q_r^{(t)} \cdot q_r^{(t)}$ and that the function $g(t)$ can be written as $g(t) = \sum_r q_r^{(t)} \cdot q_r^{(t+1)}$, the proof is analogous to the proof that $f(t) < \hat{\tau}(t)$, considering the function $B(t)$ instead of the function $\hat{\tau}(t)$, the function $g(t)$ instead of $f(t)$ and the scalar product $u \cdot v = u_\cdot \cdot v_\cdot$ instead of the bilinear form $sim(u, v)$. □

References

Affeldt, S., Labiod, L., & Nadif, M. (2021a). Regularized bi-directional co-clustering. *Statistics and Computing, 31*(3), 32.

Affeldt, S., Labiod, L., & Nadif, M. (2021b). Regularized dual-PPMI co-clustering for text data. In *Proceedings of SIGIR 2021, ACM* (pp. 2263–2267).

Ailem, M., Role, F., & Nadif, M. (2016). Graph modularity maximization as an effective method for co-clustering text data. *Knowledge-Based Systems, 109*, 160–173.

Ailem, M., Role, F., & Nadif, M. (2017). Model-based co-clustering for the effective handling of sparse data. *Pattern Recognition, 72*, 108–122.

Banerjee, A., Dhillon, I. S., Ghosh, J., Merugu, S., & Modha, D. S. (2007). A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. *Journal of Machine Learning Research, 8*, 1919–1986.

Battaglia, E., & Pensa, R. G. (2023). A parameter-less algorithm for tensor co-clustering. *Machine Learning, 112*(2), 385–427.

Boutalbi, R., Labiod, L., & Nadif, M. (2019a). Co-clustering from tensor data. In *Proceedings of PAKDD 2019* (pp. 370–383).

Boutalbi, R., Labiod, L., & Nadif, M. (2019b). Sparse tensor co-clustering as a tool for document categorization. In *Proceedings of ACM SIGIR 2019* (pp. 1157–1160).

Boutalbi, R., Labiod, L., & Nadif, M. (2022). Tensorclus: A python library for tensor (co)-clustering. *Neurocomputing, 468*, 464–468.

Chen, W., Wang, H., Long, Z., & Li, T. (2023a). Fast flexible bipartite graph model for co-clustering. *IEEE Transactions on Knowledge and Data Engineering, 35*(7), 6930–6940.

Chen, Y., Lei, Z., Rao, Y., Xie, H., Wang, F. L., Yin, J., & Li, Q. (2023b). Parallel non-negative matrix tri-factorization for text data co-clustering. *IEEE Transactions on Knowledge and Data Engineering, 35*(5), 5132–5146.

Chi, E. C., Gaines, B. J., Sun, W. W., Zhou, H., & Yang, J. (2020). Provable convex co-clustering of tensors. *Journal of Machine Learning Research, 21*, 214:1-214:58.

Deng, P., Li, T., Wang, H., Horng, S., Yu, Z., & Wang, X. (2021). Tri-regularized nonnegative matrix tri-factorization for co-clustering. *Knowledge-Based Systems, 226*, 107101.

Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings ACM SIGKDD 2001* (pp. 269–274).

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

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

Du, S., Liu, Z., Chen, Z., Yang, W., & Wang, S. (2021). Differentiable bi-sparse multi-view co-clustering. *IEEE Transactions on Signal Processing, 69*, 4623–4636.

Gao, B., Liu, T.-Y., Zheng, X., Cheng, Q.-S., & Ma, W.-Y. (2005). Consistent bipartite graph co-partitioning for star-structured high-order heterogeneous data co-clustering. In *Proceedings of ACM SIGKDD 2005* (pp. 41–50).

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

Govaert, G., & Nadif, M. (2010). Latent block model for contingency table. *Communications in Statistics—Theory and Methods, 39*(3), 416–425.

Govaert, G., & Nadif, M. (2013). *Co-clustering: Models, algorithms and applications*. Hoboken: Wiley.

Hussain, S. F., Khan, K., & Jillani, R. M. (2022). Weighted multi-view co-clustering (WMVCC) for sparse data. *Applied Intelligence, 52*(1), 398–416.

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.

Kluger, Y., Basri, R., Chang, J. T., & Gerstein, M. (2003). Spectral biclustering of microarray cancer data: Co-clustering genes and conditions. *Genome Research, 13*, 703–716.

Long, B., Zhang, Z. M., & Yu, P. S. (2005). Co-clustering by block value decomposition. In *Proceedings of ACM SIGKDD 2005* (pp. 635–640).

Madeira, S. C., & Oliveira, A. L. (2004). Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 1*(1), 24–45.

Papadimitriou, S., & Sun, J. (2008). Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *Proceedings of IEEE ICDM 2008* (pp. 512–521).

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

Peng, W., & Li, T. (2010). Temporal relation co-clustering on directional social network and author-topic evolution. *Knowledge and Information Systems, 26*, 467–486.

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.

Qiu, G. (2004). Image and feature co-clustering. In *Proceedings of ICPR 2004.* (Vol. 4, pp. 991–994).

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

Robert, V., Vasseur, Y., & Brault, V. (2021). Comparing high-dimensional partitions with the co-clustering adjusted rand index. *Journal of Classification, 38*(1), 158–186.

Tang, J., & Wan, Z. (2021). Orthogonal dual graph-regularized nonnegative matrix factorization for co-clustering. *Journal of Scientific Computing, 87*(3), 66.

Wang, J., Wang, X., Yu, G., Domeniconi, C., Yu, Z., & Zhang, Z. (2021a). Discovering multiple co-clusterings with matrix factorization. *IEEE Transactions on Cybernetics, 51*(7), 3576–3587.

Wang, M., & Zeng, Y. (2019). Multiway clustering via tensor block models. In *Proceesings of NeurIPS 2019* (pp. 713–723).

Wang, Y., & Ma, X. (2021b). Joint nonnegative matrix factorization and network embedding for graph co-clustering. *Neurocomputing, 462*, 453–465.

Wei, J., Ma, H., Liu, Y., Li, Z., & Li, N. (2021). Hierarchical high-order co-clustering algorithm by maximizing modularity. *International Journal of Machine Learning and Cybernetics, 12*(10), 2887–2898.

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

Xu, D., Cheng, W., Zong, B., Ni, J., Song, D., Yu, W., & Zhang, X. (2019). Deep co-clustering. In *Proceedings of SIAM SDM 2019* (pp. 414–422).

Yoo, J., & Choi, S. (2010). Orthogonal nonnegative matrix tri-factorization for co-clustering: Multiplicative updates on Stiefel manifolds. *Information Processing and Management, 46*(5), 559–570.

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.

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