Hierarchical Co-Clustering: Off-line and Incremental Approaches

Hierarchical Co-Clustering: Off-line and Incremental Approaches

Ruggero G. Pensa · Dino Ienco · Rosa Meo

Abstract Clustering data is challenging especially for two reasons. The dimensionality of the data is often very high which makes the cluster interpretation hard. Moreover, with high-dimensional data the classic metrics fail in identifying the real similarities between objects. The second challenge is the evolving nature of the observed phenomena which makes the datasets accumulating over time. In this paper we show how we propose to solve these problems.

To tackle the high-dimensionality problem, we propose to apply a co-clustering approach on the dataset that stores the occurrence of features in the observed objects. Co-clustering computes a partition of objects and a partition of features simultaneously. The novelty of our co-clustering solution is that it arranges the clusters in a hierarchical fashion, and it consists of two hierarchies: one on the objects and one on the features. The two hierarchies are coupled because the clusters at a certain level in one hierarchy are coupled with the clusters at the same level of the other hierarchy and form the co-clusters. Each cluster of one of the two hierarchies thus provides insights on the clusters of the other hierarchy.

Another novelty of the proposed solution is that the number of clusters is possibly unlimited. Nevertheless, the produced hierarchies are still compact and therefore more readable because our method allows multiple splits of a cluster at the lower level.

Ruggero G. Pensa · Rosa Meo
Department of Computer Science, University of Torino, 10149 Torino, Italy
E-mail: {pensa,meo}@di.unito.it

Dino Ienco
IRSTEA Montpellier, UMR TETIS, 34093 Montpellier, France
E-mail: dino.ienco@teledetection.fr

Dino Ienco
LIRMM Montpellier, UMR CNRS, 34095 Montpellier, France

As regards the second challenge, the accumulating nature of the data makes the datasets intractably huge over time. In this case, an incremental solution relieves the issue because it partitions the problem. In this paper we introduce an incremental version of our algorithm of hierarchical co-clustering. It starts from an intermediate solution computed on the previous version of the data and it updates the co-clustering results considering only the added block of data. This solution has the merit of speeding up the computation with respect to the original approach that would recompute the result on the overall dataset. In addition, the incremental algorithm guarantees approximately the same answer than the original version, but it saves much computational load. We validate the incremental approach on several high-dimensional datasets and perform an accurate comparison with both the original version of our algorithm and with the state of the art competitors as well. The obtained results open the way to a novel usage of the co-clustering algorithms in which it is advantageous to partition the data into several blocks and process them incrementally thus "incorporating" data gradually into an on-going co-clustering solution.

Keywords Co-clustering · Hierarchical clustering · Incremental clustering

1 Introduction

Clustering is a popular data mining technique that partitions data into groups (called clusters) in such a way that objects inside a group are similar to each other, while objects belonging to different groups are dissimilar [15]. When data are represented in a high-dimensional space, traditional clustering algorithms fail in finding an optimal partitioning because of the problem known as the curse of dimensionality. Some distance metrics have been proposed to deal with high-dimensional data (e.g., cosine similarity) and feature selection tries to solve the problem by a reduction in the number of features [26]. However, novel approaches have emerged in the last years. One of the most appealing approach is co-clustering [7, 16, 25] whose solution provides contemporaneously a clustering of the objects and a clustering of the features. Co-clustering algorithms are powerful because they exploit similarity measures on the clusters in one dimension of the problem in order to cluster the other dimension: that is, clusters of objects are evaluated by means of the clusters on the features and vice versa. In this way objects are clustered on the basis of a reduced space - the clusters of features - and not clustered on the original features whose high number is the source of the problem.

One of the classical aims of clustering is to provide a description of the data by means of an abstraction process. In many applications, the end-user is used to study natural phenomena by the relative proximity relationships existing among the analyzed objects. For instance, he/she compares animals by means of the relative similarity in terms of the common features w.r.t. a same referential example. Many hierarchical algorithms have the advantage that are able to produce a dendrogram which stores the history of the merge operations (or split) between clusters. As a result they produce a hierarchy of clusters and the relative position of clusters in this hierarchy is meaningful because it implicitly tells the user about the relative similarity between the cluster elements. This hierarchy is often immediately understandable: it constitutes a helpful conceptual tool to understand the inner, existing relationships among objects in the domain; it provides a visual representation of the clustering result and explains it. Furthermore, it provides a ready to use tool to organize the conceptual domain, to browse and search objects, discover their common features or differences, etc. It is a conceptual tool especially advisable if one cluster hierarchy - built on one dimension of the problem, the objects - gives insights to study the other dimension of the problem - the features - and gives information to produce the feature hierarchy. In this paper we propose a co-clustering algorithm for co-occurrence data that simultaneously produces a hierarchical organization in both of the problem dimensions: the objects and the features. In many applications both of the hierarchies are extremely useful and are searched for: in text mining, for instance, documents are organized in categories grouping related documents. The resulting object hierarchy is useful because it gives a meaningful structure to the collection of documents. On the other side, keywords are organized in groups of synonyms or words with related meaning and this hierarchy provides a semantic network with meaningful insights on the relationships between keywords. In bioinformatics and in other applications, a similar discussion applies: genes or proteins are grouped into clusters sharing a similar behavior while biological experiments by their related, involved functionalities.

In this paper, we tackle the problem of managing an evolving collection of objects. As a matter of fact, in common applications, repositories of documents are not static, but they may evolve over time. New documents may be added to the collection at any moment, and may require to be inserted in the hierarchical structure, following the co-clustering schema. The higher is the number of documents that are added to the collection the more the clustering structure becomes obsolete. The insertion of new documents in the clustering structure is not without consequence for both the hierarchies which might need even to be completely restructured in order to suitably host the new documents. In this paper we deal with this insertion process that updates the co-clustering schema and the hierarchies. As re-executing the whole co-clustering process is computationally expensive, a solution may consist in executing this task less frequently than needed. However, if the frequency of object arrival is quite high, the hierarchies may become inconsistent with the new data, and may lead to erroneous data analysis. Thus, in this paper, we propose an incremental approach which updates existing hierarchical co-clustering structures. We show that the results obtained by an incremental processing of portions of the dataset are substantially similar to those obtained by processing the whole dataset at once. Moreover, the incremental approach is significantly less time-consuming than the complete hierarchical co-clustering process.

The key contributions of this paper are the following. We present *HiCC*, a co-clustering algorithm for co-occurrence data, that are datasets in which the objects are represented by the occurrence of features. *HiCC* simultaneously produces two hierarchies of clusters: one on the objects and the other one on the features. *HiCC* employs an unusual cluster association measure in co-clustering: Goodman-Kruskal τ. It has been considered as a good choice in a comparative study on several evaluation functions for co-clustering [29,28]. As we have shown also in [20] and will show also here, in the experimental section, these hierarchies are meaningful to the end-user and are valid also under the viewpoint of several objective measures. In fact, *HiCC* is able to produce compact hierarchies because it produces $n-$ary splits in the hierarchy instead of the usual binary splits. This compactness improves the readability of the hierarchy. One of the interesting novelties of *HiCC* is that it is parameter-less: the user is not asked to provide the number of clusters for the two dimensions, which is not easy to set and usually requires a pre-processing stage. Last but not least, in this paper we formulate an incremental version of our algorithm which requires less computational time, but guarantees almost the same results than the standard version.

The paper is organized as follows: Section 2 presents an example that motivates the choice of using Goodman-Kruskal's association measure for co-clustering. We present the core of our approach in Section 3, and its incremental version in Section 4. The related experimental validation is detailed in Section 5. Section 6 analyzes the literature related to co-clustering and to the hierarchical models. Finally Section 7 concludes and presents some future perspectives of our work.

2 An introduction to Goodman-Kruskal τ

Goodman-Kruskal τ [13] has been originally proposed as a measure of association between two categorical variables: it is a measure of proportional reduction in the prediction error of a dependent variable given information on an independent variable. In Figure 1 we present a contingency table storing the distribution of the values for two categorical variables in a set of observations. The two categorical variables of the example are *Job* and *Salary* whose values have been categorized into salary levels. In this table, d_{ij} denotes the frequency of observations in database examples having respectively the i-th value of the row variable (*Salary*), and the j-th value of the column variable *Job*.

	Job=Clerk	*Job=Teacher*	*Job=Manager*	*Job=Journalist*
Salary=Low	d_{11}	d_{12}	d_{13}	d_{14}
Salary=Medium	d_{21}	d_{22}	d_{23}	d_{24}
Salary=High	d_{31}	d_{32}	d_{33}	d_{34}

Fig. 1 A contingency table of two categorical variables.

Let us take one of the two variables (e.g., *Salary*) as the dependent variable whose values should be predicted from the values of the other variable (*Job*), considered as the independent variable. $\tau_{Salary|Job}$ determines the predictive power of the variable *Job* for the prediction of the variable *Salary.* The predictive power of *Job* is computed as a function of the error in the classification of the value of *Salary.*

The prediction error is first computed when we do not have any knowledge on the values of *Job.* E_{Salary} denotes this error here. The reduction of this error allowed by *Job* is obtained by subtraction from E_{Salary} of the error in the prediction of *Salary* that we make when we have the knowledge of the value of *Job* (the independent variable) in any database observation. $E_{Salary|Job}$ denotes this latter error. The proportional reduction in the prediction error of *Salary* given *Job*, here called $\tau_{Salary|Job}$, is computed by:

$$\tau_{Salary|Job} = \frac{E_{Salary} - E_{Salary|Job}}{E_{Salary}}$$

E_{Salary} and $E_{Salary|Job}$ are computed by a predictor which uses information from the cross-classification frequencies (d_{ij}) and tries to reduce as much as possible the prediction error. In the prediction, it also preserves the dependent variable distribution (relative frequencies of the predicted categories of *Salary*) in the following way: when no knowledge is given on *Job*, the i-th value of the row variable *Salary* is predicted by the relative frequency $T_{Salary=i_{value}}/T$ where by $T_{Salary=i_{value}}$ we denote the total frequency of observations with the i-th value of *Salary* and by T the total number of observations. On the contrary, when *Job* is known for an example, the value of $Salary$ is predicted with the relative frequency $d_{ij}/T_{Job=j_{value}}$ where by $T_{Job=j_{value}}$ we denote the total number of observations with the value of *Job* in the j-th column. Therefore, E_{Salary} and $E_{Salary|Job}$ are determined by:

$$E_{Salary} = \sum_i \left((T - T_{Salary=i_{value}}) \cdot \frac{T_{Salary=i_{value}}}{T} \right)$$

$$E_{Salary|Job} = \sum_i \sum_j \left((T_{Job=j_{value}} - d_{ij}) \cdot \frac{d_{ij}}{T_{Job=j_{value}}} \right)$$

Analyzing the formulation of τ, we observe that it satisfies many desirable properties for a measure of association. For instance, it is invariant by rows and columns permutation. Secondarily, it takes values between (0,1) (it is 0 iff there is independence between the values of the variables). Finally, it has an operational meaning: given an observation, it is the relative reduction in the prediction error of the observation's dependent variable, given the knowledge on the observation's independent variable.

We intend to use τ as measure of validation of a co-clustering solution that produces an association between the clusters coming from two partitionings: one on the values of the independent variable, and the other on the values of the dependent variable. We start our discussion on co-clustering by considering the contingency table shown in Figure 2.

The contingency table of Figure 2 represents a co-clustering of the original dataset of Figure 1. Symbol RC_i represents i-th cluster on rows while CC_j

	CC_1	CC_2	
RC_1	t_{11}	t_{12}	T_{RC_1}
RC_2	t_{21}	t_{22}	T_{RC_2}
	T_{CC_1}	T_{CC_2}	T

Fig. 2 A co-clustering solution.

represents j-th cluster on columns. T_{RC_i} is the total counting of rows in cluster RC_i while T_{CC_j} is the total counting of columns in cluster CC_j. T is the global total. In this example we suppose RC_1 has aggregated the rows of the original table indicated by *Salary* in {Low, Medium} while RC_2 has aggregated rows indicated by *Salary=High*.

Similarly, CC_1 has aggregated columns indicated by *Job* in {Clerk, Teacher} while CC_2 contains columns indicated by *Job* in {Manager, Journalist}. Co-cluster (RC_i, CC_j) is represented by the value t_{ij} stored in the cell at the intersection of the i-th row cluster and the j-th column cluster. It has been computed by application of an aggregating function on the columns and rows of the original table.

The aggregation that produces the co-cluster (CC_i, RC_j) is the following:

$$t_{ij} = \sum_{Salary=x_{value} \in RC_i} \Big(\sum_{Job=y_{value} \in CC_j} d_{xy} \Big) \qquad (1)$$

where x ranges over the possible values of the variable *Salary* and y ranges over the possible values of the variable *Job*. In our proposal the aggregation process is guided by the τ measure. This measure allows to evaluate, at each step of the algorithm, the association between the two partitions (one over the rows and one over the columns). As previously shown the τ measure is asymmetric. To take into account the predictive power of both partitions the process alternates the optimization of the prediction of the row clusters given the column clusters $\tau_{RC|CC}$ and vice versa $\tau_{CC|RC}$.

3 Hierarchical Co-Clustering

In this section we present our method, named *HiCC* (Hierarchical Co-Clustering by n-ary split). Let us introduce the notation.

3.1 Notations

Before introducing our approach, we specify the notation used in this paper. Let $X = \{x_1, \ldots, x_m\}$ denote a set of m objects (rows) and $Y = \{y_1, \ldots, y_n\}$ denote a set of n features (columns). Let D denote a $m \times n$ data matrix built over X and Y, each element of D representing a frequency, a count or a binary (presence/absence) information.

Given the above described matrix D defined on the set of objects X and on the set of features Y, the goal of our hierarchical co-clustering algorithm is to find a hierarchy of row clusters $\mathcal{R}$ over X, and a hierarchy of column clusters $\mathcal{C}$ over Y. Supposing that $\mathcal{R}$ has K levels, and that $\mathcal{C}$ has L levels, $\mathcal{R}$ and $\mathcal{C}$ are defined as $\mathcal{R} = \{R_1, \ldots, R_K\}$ and $\mathcal{C} = \{C_1, \ldots, C_L\}$ (where R_1 and C_1 are the clustering at the roots of the respective hierarchy).

Each $R_k \in \mathcal{R}$ is a set of disjoint row clusters including all the rows in X. Formally $R_k = \{r_{k1}, \ldots, r_{k|R_k|}\}$ where $|R_k|$ is the total number of clusters in R_k, $r_{ki} \subseteq X$, $\bigcup_i r_{ki} = X$ and $\forall i, j$ s.t. $i \neq j$ $r_{ki} \cap r_{kj} = \emptyset$. Similarly, $C_l = \{c_{l1}, \ldots, c_{l|C_l|}\}$, and the analogous conditions hold for C_l too. $\mathcal{R}$ defines a hierarchy on the row clusters. Each R_k must also satisfy the following conditions:

1. $\forall R_{k_1}, R_{k_2}$ s.t. $k_1 < k_2$, $|R_{k_1}| \leq |R_{k_2}|$;
2. $\forall R_k$ $(k > 1)$, $\forall r_{ki} \in R_k$, $\forall R_{k_0}$ $(k_0 < k)$, $\exists! r_{k_0 j} \in R_{k_0}$ s.t. $r_{ki} \subseteq r_{k_0 j}$;

Two similar conditions must hold for $\mathcal{C}$ too.

Moreover, for any pair of levels $k \in [1, K]$ and $l \in [1, L]$ determining, respectively, a partition $R_k = \{r_{k1}, \ldots, r_{k|R_k|}\}$ of rows of D and a partition $C_l = \{c_{l1}, \ldots, c_{l|C_l|}\}$ of columns of D, we define a contingency table T^{kl}, i.e., a $|R_k| \times |C_l|$ matrix such that each element $t^{kl}_{ij} \in T^{kl}$ ($i \in 1 \ldots |R_k|$ and $j \in 1 \ldots |C_l|$) is computed as specified in Section 2.

Our approach first computes the first level of the hierarchy (R_1 and C_1). Then, it builds R_2 by optimization of $\tau_{R_2|C_1}$, having set C_1. In general, given a generic hierarchy level h, Algorithm *HiCC* alternates the optimization of $\tau_{R_h|C_{h-1}}$ and $\tau_{C_h|R_h}$, finding respectively the appropriate row clusters in R_h and the column clusters in C_h, constrained by the clusters more recently formed in the other dimension (respectively, in the partitions C_{h-1} and R_h). To compute each level of the hierarchy, we adopt an iterative algorithmic approach inspired by *τCoClust* [28], whose goal is to find a partition of rows R and a partition of columns C such that Goodman-Kruskal's $\tau_{C|R}$ and $\tau_{R|C}$ are optimized [29,30]. This choice is motivated by the fact that this algorithm produces good quality partitions with an arbitrary and non predefined number of clusters on rows and columns. Before presenting in details our algorithm we briefly introduce *τCoClust*.

3.2 Co-clustering with Goodman-Kruskal's τ

Algorithm *τCoClust* [28] can be formulated as a bi-objective combinatorial optimization problem [22] which aims at optimizing two objective functions based on Goodman-Kruskal's τ measure. The goal of *τCoClust* is to find a partition C over the feature set Y, and a partition R of the object set X such that

$$\max_{R \cup C \in \mathcal{P}} \tau(R \cup C) = \left(\tau_{R|C}, \tau_{C|R}\right) \tag{2}$$

where $\mathcal{P}$ is the discrete set of candidate partitions and τ is a function from $\mathcal{P}$ to $\mathbf{Z}$, where $\mathbf{Z} = [0,1]^2$ is the set of vectors $\mathbf{z} = \langle z_1, z_2 \rangle$ in the objective space, with $z_1 = \tau_{R|C}$ and $z_2 = \tau_{C|R}$.

To compare different candidate partitions the algorithm implements a Pareto-dominance approach. The goal is to identify optimal partitions $\mathcal{P}_{opt} \subset \mathcal{P}$, where optimality consists in the fact that no solution in $\mathcal{P} \setminus \mathcal{P}_{opt}$ is superior to any solution of $\mathcal{P}_{opt}$ on every objective function. This set of solutions is known as Pareto optimal set or non-dominated set. Recently an extension of this algorithm to multiview clustering has been shown to converge in a finite number of steps to a Pareto-optimal solution [21]. These concepts are formally defined below.

Definition 1 (Pareto-dominance) An objective vector $\mathbf{z} \in \mathbf{Z}$ is said to *dominate* an objective vector $\mathbf{z}' \in \mathbf{Z}$ iff $z_1 \geq z'_1$ and $z_2 > z'_2$ or vice versa. This relation is denoted $\mathbf{z} \succ \mathbf{z}'$ hereafter.

Definition 2 (Non-dominated objective vector and Pareto optimal solution) An objective vector $\mathbf{z} \in \mathbf{Z}$ is said to be *non-dominated* iff there does not exist another objective vector $\mathbf{z}' \in \mathbf{Z}$ such that $\mathbf{z}' \succ \mathbf{z}$.

A solution $p \in \mathcal{P}$ is said to be *Pareto optimal* iff its mapping in the objective space ($\tau(p)$) results in a non-dominated vector.

Thus, the *τCoClust* co-clustering problem is to seek for a Pareto optimal solution of equation (2).

τCoClust is shown as Algorithm 1. This algorithm takes as input the original dataset D and the number of iterations N_{iter}. As a first step it initializes R and C at the discrete partitions respectively over rows and columns. As a second step the function *ContingencyTable* is applied over the dataset D using R and C. This function applies the equation (1) over D to obtain the initial contingency table w.r.t. the two given partitions. Then the algorithm alternates the optimization of the Goodman and Kruskal τ over both dimensions.

[30] proposes a heuristic with the aim of finding a local optimum of the two coefficients $\tau_{C|R}$ and $\tau_{R|C}$. Given two co-clusterings matrices, T and T' obtained from the same original data matrix D, the differential $\tau_{R|C}$ between T and T' is given by:

$$\Delta\tau_{R|C} = \tau_{R|C}(T) - \tau_{R|C}(T')$$

Analogously, $\Delta\tau_{C|R}$ is given for $\tau_{C|R}$. The algorithm that finds the partitions on rows or on columns and that optimizes $\tau_{R|C}$ and $\tau_{C|R}$ is unique. The algorithm is *optimizePartition* and it is shown as Algorithm 2. When the partition on the rows is computed, R is modified and C is fixed. When the partition on the columns is computed, it is the reverse. Thus, for sake of brevity, we introduce two parameters of the algorithm: U for the partition to be modified; and V for the fixed partition. When $U = R$, $V = C$ and vice versa.

Algorithm 2 adopts a stochastic optimization technique for $\tau_{U|V}$. Starting from a given set of clusters, it allows to obtain another set of clusters that optimizes the objective function $\tau_{U|V}$. As first step, the algorithm randomly chooses a cluster u_b from the set of the initial clusters U. Then it randomly picks an object o belonging to u_b. It tries to move o from the original cluster

u_b to another cluster u_e, including the possibility of formation of a new cluster with o.

At the end, among all the possible clusters, it chooses the cluster $u_{e_{min}}$ that minimizes the difference $\Delta\tau_{U|V}$ with the following constraints: $u_e \neq u_b$ and $u_e \in \{U \cup \emptyset\}$. The loop from line 4 to line 9 applies function $\Delta_{\tau_{U|V}}(o, u_b, u_e, V)$ that incorporates object o to cluster u_e and computes the improvement in the objective function $\tau_{U|V}$. In [30] the computation of $\Delta_{\tau_{U|V}}$ is performed in an efficient way without computing the complete contingency table T' but only considering the effect of incorporation of o to each u_e.

Finally, at line 11 it updates the contingency table T and the row/column cluster partition. To perform this step, it first moves the object o from the cluster u_b (and delete u_b if its only member was o) to the cluster $u_{e_{min}}$ (by creating it if it contains only o). Then it updates the contingency table T consequently.

Thanks to this strategy, the number of clusters may grow or decrease at each step, and their number only depends on the effective optimization of $\tau_{U|V}$. This makes this approach substantially different from the one described in [7] and in other state-of-the-art co-clustering approaches, where the number of co-clusters is fixed as a user-defined parameter.

Algorithm 1 $\tau CoClust(D, N_{iter})$

```
1: Initialize $R$, $C$ //initialize row and column partitioning
2: $T \leftarrow ContingencyTable(R, C, D)$ //initialize the contingency table
3: while ($t \leq N_{iter}$) do
4:     $optimizePartition(R, C, T)$ //refine row partitioning
5:     $optimizePartition(C, R, T)$ //refine column partitioning
6:     $t \leftarrow t + 1$
7: end while
8: return ($R, C$)
```

Algorithm 2 $optimizePartition(U, V, T)$

```
1: $min_{\Delta\tau_{U|V}} = 0$
2: Randomly choose a cluster $u_b \in U$
3: Randomly choose an object $o \in u_b$ projected on $V$
4: for all $u_e \in \{U \cup \emptyset\} s.t. u_b \neq u_e$ do {scan all other clusters including the empty one}
5:     if ($\Delta_{\tau_{U|V}}(o, u_b, u_e, V) < min_{\Delta\tau_{U|V}}$) then {only improvements of $\tau_{U|V}$ are allowed}
6:         $u_{min} = e$ {update selected cluster}
7:         $min_{\Delta\tau_{U|V}} = \Delta_{\tau_{U|V}}(o, u_b, u_e, V)$ {update minimum $\tau_{U|V}$}
8:     end if
9: end for
10: if $min_{\Delta\tau_{U|V}} \neq 0$ then
11:     $[U, T] \leftarrow Update(U, T, o, u_b, u_{e_{min}})$ {Update $U$ moving $o$ from $u_b$ to $u_{e_{min}}$ and modify
        $T$ consequently}
12: end if
```

3.3 Description of Algorithm *HiCC*

We can now present the details of our hierarchical co-clustering approach. The initialization procedure is presented in Algorithm 3 while the whole procedure of construction of the hierarchies (named *buildHier*) is presented in Algorithm 4. *HiCC* adopts a divisive strategy. At line 1, *HiCC* initializes the partitions by calling function $\tau CoClust$ (see Algorithm 1). Thus it builds the first level of both the hierarchies. Then it starts building the hierarchies by calling function *buildHier*. At line 1 *BuilHier* initializes the indices that represent the levels for the two hierachies. From line 3 to line 15 it starts building the level $k+1$ of the hierarchy on the rows. Then, from line 16 to line 28 it builds the level $l+1$ of the hierarchy on the columns. We explain here in detail only the construction of the hierarchy on the rows since the construction of the hierarchy on the columns is analogous. At line 4 each row cluster in the partition at level k of the hierarchy is considered. At line 5 each row cluster $r_{ki} \in R_k$, is split into a new set of row clusters R'_i using *RandomSplit* function. This function first sets the cardinality of R'_i randomly; then it randomly assigns each row in r_{ki} to a cluster of R'_i. Subsequently, it initializes a new contingency table (here denoted by T_{ki}) related to the portion of the contingency table relative to cluster r_{ki} and to the set C_l of column clusters (we keep all the column clusters found at current level l). Without changing columns partition, the algorithm optimizes $\tau_{R'_i|C_l}$ using the *optimizePartition* function. It returns a new and optimized R'_i. After all r_{ki} have been processed, the algorithm adds the new level of the row hierarchy to $\mathcal{R}$ (line 14). Column clusters are then processed in analogous way, using the row cluster assignment just returned. In this way the two hierarchies grow until the $TERMINATION$ condition is reached. The $TERMINATION$ condition is satisfied when all leaves of the two hierarchies contain only one element. Obviously, a cluster may be split into singletons at a higher level than other clusters. At the end, *HiCC* returns both the hierarchies over rows ($\mathcal{R}$) and columns ($\mathcal{C}$).

Algorithm 3 $HiCC(D, N_{iter})$

```
1: $\mathcal{R} \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$ {initialize co-cluster hierarchies}
2: $(R_1, C_1) \leftarrow \tau CoClust(D, N_{iter})$ {compute the first co-cluster hierarchy level}
3: $\mathcal{R} \leftarrow \mathcal{R} \cup R_1$, $\mathcal{C} \leftarrow \mathcal{C} \cup C_1$ {update the first co-cluster level}
4: $(\mathcal{R}, \mathcal{C}) \leftarrow buildHier(\mathcal{R}, \mathcal{C})$ {compute lower co-cluster hierarchy level}
5: return $(\mathcal{R}, \mathcal{C})$
```

3.4 Local convergence of $\tau CoClust$

As shown in [30,28], the local search strategy employed to update partitions, sometimes leads to some degradation of $\tau_{R|C}$ or $\tau_{C|R}$. This is due to the fact that an improvement on one partition may decrease the quality of the other

Algorithm 4 $buildHier(\mathcal{R}, \mathcal{C})$

```
k ← 1, l ← 1
while ($TERMINATION$) do
    $R_{k+1} \leftarrow \emptyset$ {initialize next row hierarchy level}
    for all $r_{ki} \in R_k$ do
        $R'_i \leftarrow RandomSplit(r_{ki})$ {split current cluster level into random sub-clusters}
        $t \leftarrow 0$
        $T_{ki} \leftarrow ContingencyTable(R'_i, C_l, D)$ {compute the related contingency table}
        while ($t \leq N_{iter}$) do
            $optimizePartition(R'_i, C_l, T_{ki})$ {refine row partitioning}
            $t \leftarrow t + 1$
        end while
        $R_{k+1} \leftarrow R_{k+1} \cup R'_i$ {add the sub-hierarchy to the new level}
    end for
    $\mathcal{R} \leftarrow \mathcal{R} \cup R_{k+1}$ {add the new level to the global hierarchy}
    $k \leftarrow k + 1$
    $C_{l+1} \leftarrow \emptyset$ {initialize next column hierarchy level}
    for all $c_{lj} \in C_l$ do
        $C'_j \leftarrow RandomSplit(c_{lj})$ {split current cluster level into random sub-clusters}
        $t \leftarrow 0$
        $T_{lj} \leftarrow ContingencyTable(R_k, C'_j, D)$ {compute the related contingency table}
        while ($t \leq N_{iter}$) do
            $optimizePartition(C'_j, R_k, T_{lj})$ {refine column partitioning}
            $t \leftarrow t + 1$
        end while
        $C_{l+1} \leftarrow C_{l+1} \cup C'_j$ {add the sub-hierarchy to the new level}
    end for
    $\mathcal{C} \leftarrow \mathcal{C} \cup C_{l+1}$ {add the new level to the global hierarchy}
    $l \leftarrow l + 1$
end while
return ($\mathcal{R}, \mathcal{C}$)
```

one. However, in [21] it was shown that the solution converges to a Pareto-local optimal solution in a finite number of steps. Considering iterated local search algorithms for multi-objective optimization requires to extend the usual definitions of local optimum to that context [27]. Let $\mathcal{N} : \mathcal{P} \rightarrow 2^{\mathcal{P}}$ be a neighborhood structure that associates to every solution $p \in \mathcal{P}$ a set of neighboring solutions $\mathcal{N}(p) \subseteq \mathcal{P}$. A Pareto local optimum with respect to $\mathcal{N}$ is defined as follows.

Definition 3 (Pareto local optimum) A solution $p \in \mathcal{P}$ is a Pareto local optimum with respect to the set of solutions $\mathcal{N}(p)$ that constitutes the neighborhood of p, if and only if there is no $q \in \mathcal{N}(p)$ such that $\tau(q) \succ \tau(p)$ (see equation (2)).

Let $p = R \cup C$ be an element of $\mathcal{P}$. In algorithm $\tau CoClust$ (see Algorithm 1), the neighborhood $\mathcal{N}$ of p is defined as the set of all candidate solutions determined by the movement of the element x from the cluster r_b to the cluster r_e in R, or by the movement of the element y from the cluster c_b to the cluster c_e in C. As proved in [21], the following theorem holds:

Theorem 1 *The iterated local search algorithm τCoClust terminates in a finite number of steps and outputs a Pareto local optimum with respect to $\mathcal{N}(p)$.*

The complete proof (omitted here) is presented in [21].

In our hierarchical approach (see Algorithm 4), instead of updating a cluster at a time, alternating the dimensions, a whole partition of one of the dimensions is updated, a single cluster at a time, while keeping the other partition fixed. Each update is accepted provided that the objective function increases. Thus, the algorithm ensures that the objective function increases at each iteration.

3.5 Complexity Discussion

HiCC complexity is influenced by three factors. The first one is the number of iterations. It influences the convergence of the algorithm. A deep study of the number of iterations is presented in [28]. The second factor is the number of row/column clusters in *optimizePartition* function. This number of clusters influences the swap step which tries to optimize the internal objective function moving one object from a cluster identified by b to another cluster identified by e. The number of possible different clusters that e can identify influences the speed of the algorithm, but this number varies during the optimization. The third factor is the depth of the two hierarchies, in particular the deeper one. This value influences the number of times the main loop of Algorithm 4 is repeated. If we denote by N the number of iterations, by $\widetilde{c}$ the mean number of row/column clusters inside the optimization procedure, and by v the mean branching factor, we observe that the complexity of a split of r_{ki} and c_{lj} is equal as average to $O(N \times \widetilde{c})$. Each split is performed for each node of the two hierarchies except for the bottom level. The number of nodes in a tree with branching factor v is $\sum_{i=0}^{levels} v^i$ where *levels* is the number of levels of the tree. We can expand this summation and we obtain $\frac{1-v^{1+levels}}{1-v}$. From the previous consideration we estimate the complexity of our approach $O\left(N \times \widetilde{c} \times \frac{1-v^{levels}}{1-v}\right)$. The worst case is verified when any split is binary and at each level $\widetilde{c} = n$, where n is the number of rows/columns. In this case, the complexity is $O(N \times (n-1) \times n)$, $(n-1)$ being the number of internal nodes in the hierarchy. In conclusion, in the worst case, the overall complexity is $O(Nn^2)$. In the experimental section, we show that our algorithm often splits into more than two clusters, thus reducing the number of internal nodes. Moreover, assumption $\widetilde{c} = n$ is very pessimistic. In general our algorithm runs in linear time with the number of iterations, and in subquadratic time with the number of objects/features. Notice also that in this work, the number of iterations is constant, but it could be adapted to the size of the set of objects, and then reduced at each split.

4 Incremental hierarchical co-clustering

In this section we tackle the problem of dealing with evolving collections of data, i.e., datasets that grow with time. As an example, consider a dataset of documents published by a newsgroup and described by the occurrence of a set of words. A newsgroup is dynamic and it is updated frequently. A pre-computed model which takes into account only the first portion of posts (documents) may become obsolete after that a certain amount of new documents have been added to the database. Thus, the hierarchical co-clustering model needs to be updated in order to take into account the new data. This operation may be performed by re-launching the hierarchical co-clustering process on the whole updated dataset. However, this may require a high computational time, and make the analyst's experience frustrating, since it has to wait for the new co-clustering to be computed.

To overcome this problem, we introduce an incremental version of *HiCC* algorithm that starts from a previously computed hierarchical co-clustering and updates it. From the results of this paper, we will see that in the revised version of *HiCC*, that we will name $HiCC_{incr}$, less computational resources are needed than in the traditional one, when the algorithm is re-launched on the complete dataset. As shown by the experiments in Section 5, computing only the hierarchical structure (step at line 4 of Algorithm 3) on the whole dataset, having already initialized the initial partitions, requires significantly less time than computing also the initial (first level) splitting (that corresponds to the execution of $\tau CoClust$ at line 2).

As a consequence, we focus our incremental optimization on this first step, and let unvaried the remaining part of the algorithm. This is also motivated by the fact that our algorithm is divisive: variations of the first co-clustering level may make inconsistent the existent hierarchy.

Another important consideration concerns the way our algorithm handles the new incoming objects. New objects may be processed one-by-one or in packets. The choice depends mainly on the application and the data, and on how an obsolete hierarchical co-clustering model is tolerated by the analyst's needs.

4.1 Incremental version of $\tau CoClust$

In the remainder of this section, we consider a pre-computed hierarchical co-clustering on a first portion of the dataset, named D and an updated portion Δ_D of arbitrary size. We assume that D is built over the object-set X and the feature-set Y, and Δ_D is built over the object-set Δ_X and the feature-set Δ_Y. In the applications we consider here, $X \cap \Delta_X = \emptyset$, while $Y \cap \Delta_Y$ is not empty. For instance, our framework can be applied to document repositories that are periodically fed by adding new documents, while the existing ones are not updated. However, new objects may contain new features as well as features that were already present in the initial portion of the dataset. For instance, part of the terms contained in new documents may appear also in old ones, but they may probably contain a certain number of new words. We think that our assumption is quite general and allows us to include many existing application schemas, typical of online evolving data repositories.

We formulate our incremental approach as follows. Starting from a previous co-clustering schema (R, C) over Δ_D we generate a new result (Δ_R, Δ_C) that incorporates the information carried by Δ_D. The overall incremental hierarchical co-clustering approach is described by Algorithm 6. It takes the original dataset D, the updating dataset Δ_D and the first level of the co-clustering over D (given by R and C) as input. It provides a hierarchical co-clustering $(\mathcal{R}, \mathcal{C})$ on $D \cup \Delta_D$. The algorithm is similar to the non-incremental version (see Algorithm 4), except that, instead of partitioning the whole dataset from scratch, it employs the pre-computed co-clustering (R, C), by calling $\tau CoClust_{incr}$ (Algorithm 5). Algorithm 5 describes the procedure that extends $\tau CoClust$ (i.e. Algorithm 1) to the incremental setting.

The new procedure takes as input the two dataset portions D and Δ_D, the previous co-clustering results over D, i.e., (R, C), and a number of iterations. At line 1 of Algorithm 5, the row set Δ_X of Δ_D is initialized by assigning each row to a singleton cluster. Instead, columns in $\{\Delta_Y \cap Y\}$ are assigned to the corresponding cluster of C, while each column $\{\Delta_Y \setminus Y\}$ is assigned to a singleton cluster. Finally, columns in $\{Y \setminus \Delta_Y\}$ are zero-filled and assigned to the corresponding cluster of C. We call the resulting partitioning Δ_R and Δ_C. The overall partitioning over $D \cup \Delta_D$ is now $R = R \cup \Delta_R$ and $C = C \cup \Delta_R$ (lines 2 and 3). Given the so computed R and C, the algorithm initializes the contingency table T over $D = D \cup \Delta_D$ (line 5). In other words, after these preliminary steps, the contingency table T contains as many row clusters as $R \cup \Delta_R$, and as many column clusters as $C \cup \Delta_C$. Finally (lines 6-10) the algorithm alternatively optimizes R and C, following the updating criteria given in Section 3, and described by function *optimizePartition* (see Algorithm 2). At the end, the original R and C partitions have been rewritten and constitute the final optimized output (line 11).

Algorithm 5 $\tau CoClust_{incr}(D, \Delta_D, R, C, N_{iter})$

```
1: Initialize $\Delta_R$ and $\Delta_C$
2: $R \leftarrow R \cup \Delta_R$ {set the new row partitioning}
3: $C \leftarrow C \cup \Delta_C$ {set the new column partitioning}
4: $D \leftarrow D \cup \Delta_D$ {concatenate the two dataset portions}
5: $T \leftarrow ContingencyTable(R, C, D)$ {compute the new overall contingency table}
6: while $(t \leq N_{iter})$ do
7:     $optimizePartition(R, C, T)$ {refine row partitioning}
8:     $optimizePartition(C, R, T)$ {refine column partitioning}
9:     $t \leftarrow t + 1$
10: end while
11: return $(R, C)$
```

Algorithm 6 $HiCC_{incr}(D, \Delta_D, R, C, N_{iter})$

```
$\mathcal{R} \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$ {initialize co-cluster hierarchies}
$(R_1, C_1) \leftarrow \tau CoClust_{incr}(D, \Delta_D, R, C, N_{iter})$ {compute the first co-cluster hierarchy level}
$\mathcal{R} \leftarrow \mathcal{R} \cup R_1$, $\mathcal{C} \leftarrow \mathcal{C} \cup C_1$ {update the first co-cluster level}
$(\mathcal{R}, \mathcal{C}) \leftarrow buildHier(\mathcal{R}, \mathcal{C})$ {compute lower co-cluster hierarchy level}
return $(\mathcal{R}, \mathcal{C})$
```

In the next section, we provide a comprehensive experimental analysis that shows that the incremental version provides good hierarchical co-clustering results, but requires significantly less computational time.

5 Experimental validation

In this section we report on several experiments performed on real, high-dimensional, multi-class datasets. We compare our approach with Information-Theoretic Co-Clustering (*ITCC*) [7], a well-known co-clustering algorithm which minimizes the loss in mutual information. To evaluate our results, we use several objective performance parameters that measure the quality of clustering. Besides the precision of the hierarchical co-clustering we analyze also the hierarchies returned by our approach for both rows and columns. This analysis allows to emphasize the utility of the hierarchical structure w.r.t. standard flat co-clustering approaches. Furthermore, we provide an exhaustive set of experiments to point out the differences in computational time and quality of the results between the off-line approach and the incremental one. All experiments are performed on a PC with a 2.6GHz Opteron processor, 4GB RAM, running Linux.

The section is organized as follows. We first introduce the datasets and the quality indices we adopted in our experimental study. Then, we show and analyze some examples of co-cluster hierarchies that *HiCC* is able to provide and compare them with the partitions found by *ITCC*. The stability of *HiCC* is then assessed both in terms of average depth of the hierarchies and their content. Finally, the incremental version of our algorithm is evaluated by considering it in both flat and hierarchical co-clustering settings. The stability of the results is studied in this case as well.

5.1 Datasets for the evaluation of *HiCC* and its results

To evaluate our results we use some of the datasets described in [10]. In particular we use:

- **oh0**, **oh15**: two samples from OHSUMED dataset. OHSUMED is a clinically-oriented MEDLINE subset of abstracts or titles from 270 medical journals over five-year period (1987-1991).

Dataset	n. instances	n. features	n. of classes
oh0	1003	3182	10
oh15	913	3100	10
tr11	414	6429	9
tr21	336	7902	6
re0	1504	2886	13
re1	1657	3758	25

Table 1 Datasets characteristics

- **tr11**, **tr21**: two samples from TREC dataset. These data come from the Text REtrieval Conference archive.
- **re0**, **re1**: two samples from Reuters-21578 dataset. This dataset is widely used as test collection for text categorization research.

All datasets have more than five classes, which usually is a hard context for text categorization. The characteristics of datasets are shown in Table 1.

5.2 External Evaluation Measures

We evaluate the algorithm performance using two external validation indices. We denote by $\mathbf{C} = \{C_1 \ldots C_J\}$ the partition built by the clustering algorithm on objects at a particular level, and by $\mathbf{P} = \{P_1 \ldots P_I\}$ the partition inferred by the original classification. J and I are respectively the number of clusters ($|\mathbf{C}|$) and the number of classes ($|\mathbf{P}|$). We denote by n the total number of objects.

The first index is the *Normalized Mutual Information* (NMI). NMI provides an information that is impartial with respect to the number of clusters [33]. It measures how clustering results share the information with the true class assignment. NMI is computed as the average mutual information between every pair of clusters and classes:

$$\mathbf{NMI} = \frac{\sum_{i=1}^{I} \sum_{j=1}^{J} x_{ij} \log \frac{n x_{ij}}{x_i x_j}}{\sqrt{\sum_{i=1}^{I} x_i \log \frac{x_i}{n} \sum_{j=1}^{J} x_j \log \frac{x_j}{n}}}$$

where x_{ij} is the cardinality of the set of objects that occur both in cluster C_j and in class P_i; x_j is the number of objects in cluster C_j; x_i is the number of objects in class P_i. Its values range between 0 and 1.

The second measure is the adjusted Rand index [19]. Let a be the number of object pairs belonging to the same cluster in $\mathbf{C}$ and to the same class in $\mathbf{P}$. This metric captures the deviation of a from its expected value corresponding to the hypothetic value of a obtained when $\mathbf{C}$ and $\mathbf{P}$ are two random, independent partitions. The expected value of a denoted by $E[a]$ is computed as follows:

$$E[a] = \frac{\pi(C) \cdot \pi(P)}{n(n-1)/2}$$

where $\pi(C)$ and $\pi(P)$ denote respectively the number of object pairs from the same clusters in **C** and from the same classes in **P**. The maximum value for a is known to be:

$$max(a) = \frac{1}{2}\left(\pi(C) + \pi(P)\right)$$

The agreement between **C** and **P** can be estimated by the adjusted rand index as follows:

$$AR(\mathbf{C}, \mathbf{P}) = \frac{a - E[a]}{max(a) - E[a]}$$

Notice that this index can take negative values, and when $AR(\mathbf{C}, \mathbf{P}) = 1$, we have identical partitions.

To obtain a quality measure of *HiCC*, for each level i of the hierarchy on the rows we select the corresponding level of the hierarchy on the columns. These levels define a pair of partitions: the first partition comes from the row cluster hierarchy, and the second one from the columns cluster hierarchy. On the pair of partitions from level i, an evaluation function EF_i is computed on the basis of Goodman-Kruskal τ_S [14], which is a symmetrical version of τ [13] whose aim is to quantify the agreement between the two partitions. In order to compute an overall measure for the co-clustering we compute the following weighted mean:

$$Goodness = \frac{\sum_{i=1} \alpha_i \cdot EF_i}{\sum_{i=1} \alpha_i} \tag{3}$$

where α_i is the weight associated to the i-th level of the hierarchy, and allows to specify the significance assigned to the i-th level w.r.t. to the other levels of the hierarchy. Of course, EF_i might refer to any other validation index presented beforehand.

(3) is a general formula for the evaluation of the goodness of a run of our method. In this work we set $\alpha_i = 1/i$. Indeed, we give a heavy weight to the top level and the lowest weight to the last level. This choice is motivated by observation that in a hierarchical solution the clusters on a level depend on the clusters at previous level. If we start with a good clustering, then next levels are more likely to produce good results too.

5.3 Inspecting hierarchies

In this section we analyze in a qualitative way two hierarchies built by our algorithm. In particular we analyze the row hierarchy for **oh0** data and the column hierarchy from **re1** data. We chose these two datasets because **oh0** has an understandable class description and **re1** uses a vocabulary (feature space) with common words. In Table 2 we report the first three levels of the row hierarchy produced by *HiCC*. To obtain this hierarchy we assigned at each cluster a label. We chose the label of the majority class in the cluster. Each column in Table 2 represents a level of the hierarchy and each cell of the table represents a single cluster. We notice that the produced hierarchy is

1st level	2nd level	3rd level
Aluminum	Aluminum	Aluminum
		Aluminum
	Aluminum	Aluminum
		Aluminum
Enzyme-Activation	Enzyme-Activation	Enzyme-Activation
		Enzyme-Activation
	Cell-Movement	Cell-Movement
		Leucine
Staphylococcal-Infections	Uremia	Uremia
		Staphylococcal-Infections
	Staphylococcal-Infections	Staphylococcal-Infections
		Memory

Table 2 Row hierarchy of **oh15**

meaningful to a domain expert. In fact, the first branch is homogenous, since each node contains a majority of elements coming from the *Aluminum* class. Cluster *Enzyme-Activation* at first level is split into two clusters: the first one has the same label, the second is *Cell-movement*, a topic more related to *Enzyme-Activation* than to the other cluster labelled *Staphylococcal-Infections*. Then *Cell-movement* is split into two clusters. One of these is *Leucine*, which is a fundamental amino acid of the protein cells. In the last branch of the hierarchy we notice that *Uremia* cluster is a child of *Staphylococcal-Infections* (a renal failure with bacterial infection as one of its causes).

In Table 3 we report instead the first two levels of the column hierarchy produced by *HiCC*. In order to assign a label to the clusters, we computed for each cluster the mutual information between each word and the cluster; then we ranked the set of words for each cluster, as described in [32]. Finally, we selected the 10-top words for each cluster and use them in Table 3 to describe the clusters. In this way we identified the words whose meaning was almost exclusively connected to each cluster. Again, we can notice that the generated column hierarchy is meaningful. At the first level of the hierarchy each of the six clusters is about a well distinct topic: the first one is about *coffee and cocoa*. The second one is on *agriculture*. The third and the forth ones are about *gulf war* and *oil economy*. Finally the fifth and the sixth clusters are about *syndicate and work* and *Aegean politics*. The second level introduces a further specialization of the top-level clusters. For instance, the first cluster is split into *coffee and cocoa production* and *agricultural economics*. The fourth one is split into *oil industry* and *finance*.

5.4 *HiCC* vs Partitional Co-Clustering

Here we evaluate *HiCC* performance w.r.t *ITCC* [7]. *HiCC* is non-deterministic like other well-known clustering algorithms such as K-means or *ITCC* itself. At each run we can obtain similar, but not equal, hierarchies. For this reason we run *HiCC* 50 times over each dataset. We set the number of iterations for the first level equal to 10 times the maximum between the number of instances and the number of attributes. Instead from the second levels of the hierarchy

1st level	**2nd level**
coffee, buffer, cocoa, deleg, consum, ico, quota, stock, produc, icco	coffee, buffer, deleg, cocoa, consum, ico, icco, council, pact, bag
	quota, produc, stock, meet, brazil, intern, talk, agreem, negoti, organ
tonne, wheate, sugar, grain, agricultur, usda, crop, corn, ec, soybean	mln, vexport, market, plant, offer, import, trade, trader, sale, week
	tonne, wheate, sugar, grain, crop, usda, corn, agricultur, ec, soybean
ship, gulf, iran, missil, tanker, iranian, attack, ferri, vessel, load	ship, gulf, vessel, water, load, wait, militari, hit, war, ice
	iran, missil, tanker, iranian, attack, ferri, escort, kuwait, warship, iraq
oil, barrel, opec, compani, gold, bpd, crude, ga, energi, ounce	oil, barrel, gold, compani, opec, bpd, crude, ga, ounce, reserv
	dlr, price, product, tax, industri, dollar, contract, pct, increas, dai
strike, seamen, union, port, worker, miner, employ, disput, labour, leader	union, port, miner, disput, pai, leader, spokesman, sector, cargo, protest
	strike, seamen, worker, employ, labour, redund, marin, rotterdam, janeiro, fnv
aegean, greece, greek, turkish, turkei, papandr, athen, nato, right, row	aegean, greece, greek, turkei, turkish, papandr, athen, nato, row, ankara
	right, readi, territori

Table 3 Column hierarchy of **re1**

(where the optimization is independent for each dimension) we run the process until these two conditions are satisfied: a) the total number of iterations is more than 50,000 and b) the number of iterations which end with no changes in the cluster structure is smaller than the number of objects/attributes. These two conditions are standard criteria adopted in partitional clustering algorithms. From the set of row/column co-hierarchies obtained in the different runs, we choose the ones that better optimize an internal evaluation function.

To compare each level of our hierarchies with *ITCC* results, we need to fix a number of row/column clusters to set *ITCC* parameters. We recall that *ITCC* is flat and does not produce hierarchies. For this reason we plan our experiments in the following way. Since *HiCC* is not deterministic, each run may produce partitions of different cardinality at each level. For this reason, we need to select one specific run of *HiCC*. Using *Goodness* function with $\tau_{\mathcal{S}}$ as evaluation function, we choose for each dataset the hierarchy with the highest Goodness value. This hierarchy is a representative solution whose selection is not biased by the external index (based on the classes) that will be used for the final comparison evaluation.

From this hierarchy, we obtain a set of pairs (#numberRowCluster, #numberColCluster), where each pair specifies the number of clusters in a level of the hierarchy. For each of these combinations, we run *ITCC* 50 times with (#numberRowCluster,#numberColCluster) as parameters, and average for each level the obtained results.

In Table 4 we show the experimental results. To obtain a single index value for each dataset we compute the previously proposed *Goodness* having as evaluation function (EF_i) each of the two external validation indices. These two indices are computed between the partition on the objects given by the

	HiCC		**ITCC**	
Dataset	**NMI**	**Adj. Rand Index**	**NMI**	**Adj. Rand Index**
oh0	**0.5301**	**0.2275**	0.4535	0.213
oh15	**0.379**	0.1317	0.3531	**0.1503**
tr11	**0.4077**	0.1438	0.4038	**0.1464**
tr21	0.1787	**0.0305**	**0.2027**	0.0157
re0	0.2568	0.0415	**0.3065**	**0.0803**
re1	**0.4565**	**0.1544**	0.3922	0.1047

Table 4 Comparison between *ITCC* and *HiCC* with the same number of clusters for each level

		HiCC		**ITCC**	
RowClust	ColClust	NMI	Adj. Rand	NMI	Adj. Rand
6	6	**0.4149**	**0.2691**	0.2755 ± 0.0403	0.1683 ± 0.0489
13	12	**0.3771**	**0.2488**	0.3607 ± 0.0297	0.1897 ± 0.0306
28	29	0.3779	**0.1955**	**0.4345** ± 0.0177	0.167 ± 0.0158
58	81	0.4032	**0.1511**	**0.4433** ± 0.011	0.1144 ± 0.0077
121	184	**0.4481**	**0.092**	0.3213 ± 0.0107	0.0262 ± 0.004
250	441	**0.4846**	**0.0504**	0.3176 ± 0.0036	0.0034 ± 0.0005
470	855	**0.5236**	**0.0275**	0.4013 ± 0.0026	0.0017 ± 0.0002

Table 5 Complete view of performance for the top 8 levels of **re1**

clusters and the partition on the objects given by the classes. From the results we can see that our approach is competitive w.r.t. *ITCC*. Notice however that, for a given number of desired co-clusters, *ITCC* tries to optimize globally its objective function, and each time it starts from scratch. Thus, one would expect that *ITCC* provides better results for each pair of number of clusters.

On the contrary, *HiCC* solution at level i in the hierarchy is constrained by clusters found at level $i-1$ in the same hierarchy. Thus one would expect that this behaviour would propagate the errors level by level. Instead, we can notice that *ITCC* is not more accurate than our algorithm. This phenomenon has been recently observed in [6], where a hierarchical model selection is used to help improving results in prediction tasks. To clarify this point we report the complete behavior of the two algorithms in an example. In Table 5 we report the value of the two indices for each level of the hierarchy obtained by *HiCC* for **re1**. For the sake of brevity, we can only show here one example, but in all the other experiments we observe the same trend here described. In the same table we show also the values obtained by *ITCC* using the same number of clusters (#numberRowCluster,#numberColCluster) discovered by *HiCC*. We also report the standard deviation for *ITCC*, since for each pair of cluster numbers, we run it 50 times. We can see that *HiCC* outperforms *ITCC*, especially at the higher levels (first, second and third) of the row hierarchy. We notice also that NMI index always increases monotonically in *HiCC* but not in *ITCC*. This experiment shows that, when we explore deeper levels of the hierarchies of *HiCC*, the confusion inside each cluster decreases.

Dataset	**Goodness at 1st level**	**Goodness until the 5th level**	**Goodness for all the levels**
oh0	0.2676 ± 0.0065	0.1824 ± 0.0047	0.1394 ± 0.0034
oh15	0.2783 ± 0.0012	0.1829 ± 0.001	0.1371 ± 0.0022
tr11	0.2301 ± 0.0005	0.1621 ± 0.0006	0.1227 ± 0.0015
tr21	0.3005 ± 0.0	0.1893 ± 0.0028	0.1314 ± 0.0027
re0	0.2409 ± 0.0149	0.1163 ± 0.0044	0.1142 ± 0.0063
re1	0.1728 ± 0.0081	0.1163 ± 0.0044	0.09 ± 0.0029

Table 6 Average Goodness on the basis of τ_S

Dataset	**Row Hier. Depth**	**Col. Hier. Depth**
oh0	14.9±1.02	17.6±1.28
oh15	15.08±0.93	18.48±1.49
tr11	14.5±1.41	21.38±1.56
tr21	16.23±1.43	27.96±2.43
re0	17.04±1.13	19.98±2.08
re1	15.92±1.01	18.2±1.46

Table 7 Mean depth of hierarchies

5.5 Stability of the induced Hierarchies

The intrinsic nature of *HiCC* is non-deterministic. As such, two instances of the algorithm processing the same dataset may provide two different results. Here, we measure the stability of our approach, in terms of quality and depth of the hierarchies, and their structure. In Table 6 we report the average Goodness of τ_S for each dataset, and the related standard deviation, computed over one, five and all hierarchy levels respectively. We observe that standard deviation is very low w.r.t. the average Goodness. From this empirical evaluation we can conclude that the quality of the hierarchical co-clustering is quite stable. In Table 7 we show the mean depth of the row hierarchy and of the column hierarchy for each dataset. We observe that the standard deviation is low. This points out that our algorithm is stable also from this point of view. Notice that *HiCC* generates hierarchies which are not deep, if the number of levels is compared with the cardinality of the object and attribute sets. Shorter hierarchies are preferable to hierarchies obtained only by binary splits, since they allow a compact representation of the data and they improve the exploration of the results because are easy to browse from a user point of view.

In order to evaluate the stability of the produced hierarchies from a different viewpoint, we adapted the strategy of summarizing and indexing hierarchies presented in [34, 24] namely, by Concept Propagation/Concept Vector (CP/CV). CP/CV assumes that each cluster node of the hierarchy is projected into a concept space and represented by a vector with as many dimensions as the cardinality of the concept space. In our context the concept space is given by the class labels. The CP/CV approach combines the concept space representation with the structure of the hierarchy. It adopts a process that propagates several times the information from the parent nodes to their children and vice versa, and takes into account the tree structure together with its content. The

	Cosine Distance	
Dataset	Avg.	std. dev.
oh0	0.0063	± 0.0124
re0	0.0023	± 0.0038
re1	0.0207	± 0.0142
tr21	0.0049	± 0.0164
oh15	0.0334	± 0.0328
tr11	0.0001	± 0.0001

Table 8 Average and Std. Dev. for Cosine distance to evaluate the stability of the induced hierarchies

authors suggest that, after a sufficient number of propagation steps, a good summary of the whole concept hierarchy is obtained in the root vector. They show that using only the root node vector, as a candidate summary, they are able to perform a high quality indexing of the hierarchy.

We adopt this technique to summarize the content of the hierarchies and quantify their stability in terms of the class concept space. We represent each node of the hierarchy by a vector having as many components as the number of classes. Each vector contains the distribution of the objects classes within the related node. Thanks to the CP/CV method we obtain for each result of *HiCC* one vector summarizing the whole hierarchy. To quantify the stability of our approach, we employ a simple strategy: we compute the distance between any pair of summary vectors obtained by different runs of *HiCC*. Then we compute the average and the standard deviation of these distances. Here, we use the cosine distance:

$$cosDist(X, Y) = 1 - \frac{\sum_i X_i \cdot Y_i}{||X|| \cdot ||Y||}$$

where X and Y are two vectors and $|| \cdot ||$ is the Euclidean norm of a vector.

For each dataset, we run the algorithm 50 times. The average and standard deviation of the cosine distances are computed over these 50 instances. We report the results in Table 8.

We observe that the average distance is very low. Even though the standard deviations are as high as the average, the order of magnitude is still very low. This means that the stochastic optimization process employed by our approach does not affect the stability of the final result.

5.6 Evaluation of $\tau CoClust_{incr}$

In this section we evaluate the behavior of $\tau CoClust_{incr}$ (see Algorithm 5 in Section 4). Before introducing this experimental study, we further motivate the incremental version providing a brief report on the time performances of *HiCC*. In Table 9 we show the average time (with standard deviation) for the two components of the algorithm. We observe that the step which requires most of the computational time is the first one ($\tau CoClust$). In general the time employed by the first step is one order of magnitude greater than the

	τCoClust		**buildHier**	
Dataset	**Avg**	**std. dev.**	**Avg**	**std. dev.**
oh0	386.382	±329.36	24.77	±9.05
oh15	301.05	±206.53	28.54	±13.65
tr11	309.97	±126.04	103.16	±59.72
tr21	266.51	±111.03	137.50	±40.32
re0	1018.68	±637.39	63.32	±33.43
re1	1405.40	±765.80	39.83	±20.18

Table 9 Original Time Performance for each algorithmic component

time employed by the second step (*buildHier*). This observation has motivated the incremental version presented in this work (see Section 4 for the details). Now we will show that the use of the incremental approach helps to decrease the computational time spent by $\tau CoClust$. This improvement allows to speed up the whole procedure and it allows the employment of *HiCC* in an incremental/on-line scenario.

To this purpose, using all the datasets described beforehand, we simulate a simple incremental scenario. We divide each dataset into two blocks. The first block is supplied to the $\tau CoClust$ to obtain a first bi-partition. $\tau CoClust_{incr}$ starts from the previous result and updates the bi-partition using the second block. The goal of this first group of experiments is to prove that the results provided by $\tau CoClust_{incr}$ are not very dissimilar from those provided by the off-line version (see Algorithm 1 in Section 3). We let vary the size of the first block from 5% to 90% of the entire dataset (with a step of 5%). To evaluate the results we set up two types of comparison: a) using the partition induced by the class variable as reference; b) using the partition computed by the off-line version as reference. We average the results over 50 runs of both the algorithms. Here, we use the same sets of features for the two blocks. If in the first block some features are not represented, they simply contains zeros for all the rows. This choice does not influence the final results, since columns containing only zeros have no impact on the computation of the objective function.

In Figure 3 we plot the time performances of the incremental version. The X axis represents, in percentage, the size of the first block. We compute the sum of the time spent for processing the first block plus the time spent for the second block.

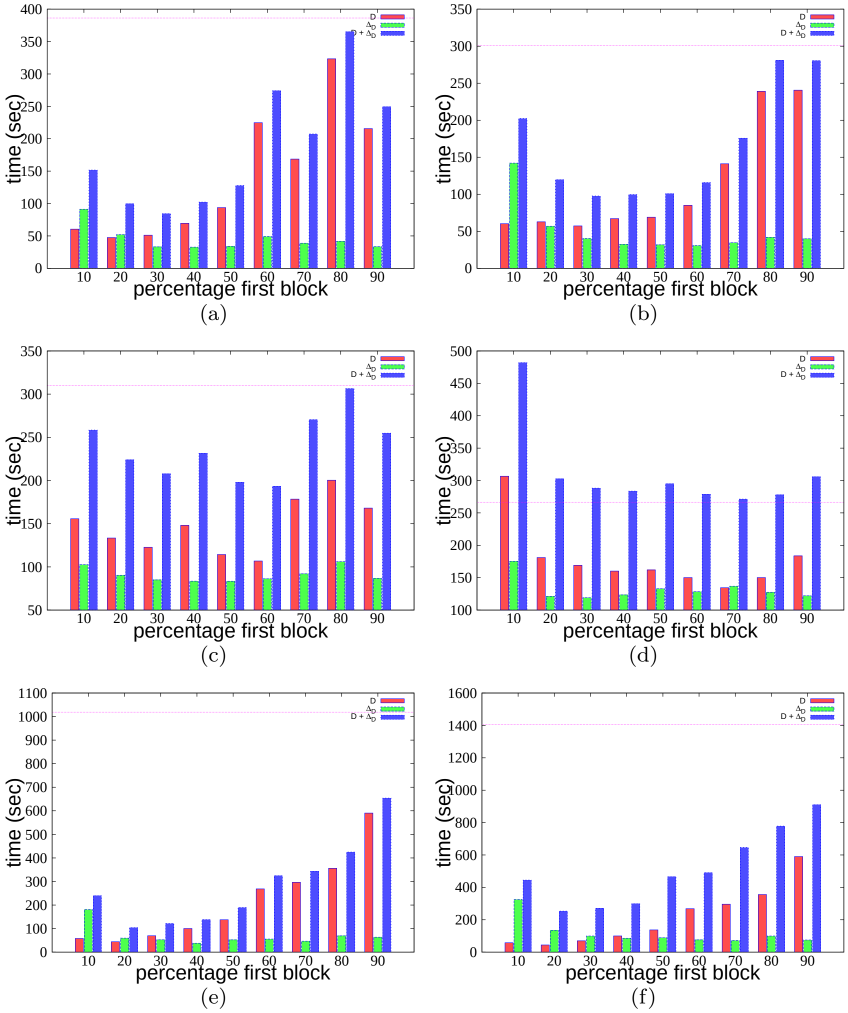
The various vertical segments (box plots) represent the average time spent to process the different blocks. The horizontal line represents the average time spent by the off-line version of the algorithm. We observe that, in general the time required by the incremental version is always smaller than the time spent by the off-line version. Notice that all the curves share a similar general trend. For very small sizes of the initial block, the time employed by $\tau CoClust_{incr}$ is similar to that taken by $\tau CoClust$. The rationale is that the initial sample is not that representative of the entire dataset and thus the arrival of a new big block arises the whole re-computation of the co-clustering. Similarly, high sizes of the initial block require almost the same time needed by the computation of the entire dataset. The computational time of the two versions is, again, similar. In the middle, for

small block sizes (20% to 40%) the incremental algorithm employs from 2 times up to 10 times less than the off-line algorithm to complete co-clustering. The only exception is **tr21**. This may be partly due to the fact that this dataset contains very few documents (about 300), but a huge number of features (about 8000).

We consider now the quality of the results provided by $\tau CoClust_{incr}$ and compare them with the results provided by $\tau CoClust$. The first comparison takes into consideration the NMI and ARI indexes computed w.r.t. the class variable (see Figures 4 and 5). We observe that the results are always close to those provided by the off-line version (the horizontal line in the plots). With 20% to 30% of initial block size, the performance are already reasonably good and the computational time is kept at its lowest level (see Figure 3). However these experiments only show that the agreement between the discovered partitions and the class partitions are maintained.

To evaluate the distortion introduced by the incremental partitioning, we measure the NMI and ARI indexes w.r.t. the original partitioning. In Figure 6 and 7, we plotted the average values of the two indexes together with the standard deviation. The average values are computed by considering each of the 50 runs of the incremental algorithm with each of the 50 runs of the off-line version. The horizontal line represents the average of the index values computed by the off-line version only. First, notice that all the indexes are significantly high. In general, the NMI and ARI computed for the incremental results are close to the average results which consider only the off-line version. This confirms that, the distortion introduced by the incremental version is comparable to the amount of natural unstability of the algorithm, due to its stochastic optimization approach. These empirical evidences confirm that this strategy can be adopted also to manage standard datasets, since it enables to speed up the entire co-clustering process.

Finally, we analyze the incremental algorithm from another point of view. Due to the stochastic nature of our optimization solution, we cannot assure that similar clusters belonging to two different blocks will always merge in the final partition. This may lead to uncorrect interpretations of the results as a negative side-effect. In practice, however, this condition never happens. In Table 10, we report the number of clusters found by our algorithm given different dimensions of the initial block. In almost all the datasets, these values are stable and are not correlated to the percentage of the block size. We may notice significant variations only for very low percentages (10%), which may be explained by the poor representativity of the initial sample. In **re1** the variation (still low), seems stronger than in the other datasets, but it is within the typical tolerance range for this particular dataset. As a side observation, we might notice that the standard deviation of the values is in general quite



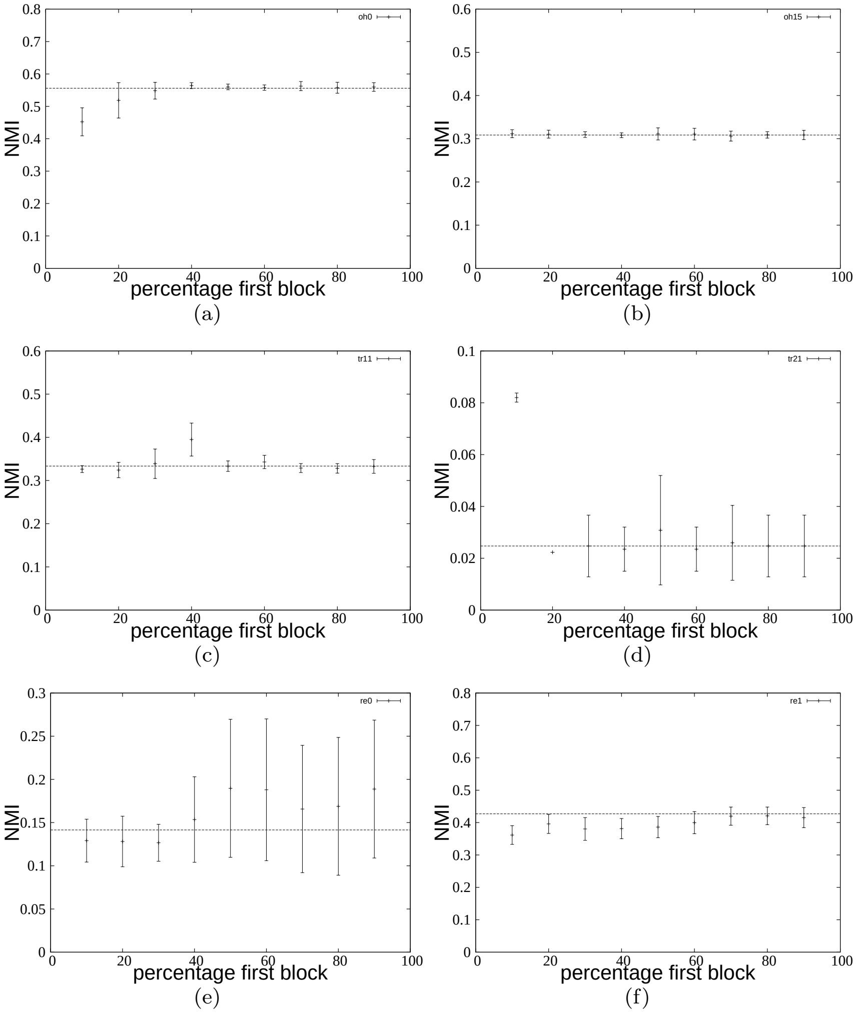
(a) (b) (c) (d) (e) (f)

Fig. 3 Time performance w.r.t. the percentage of the first block for (a) oh0 (b) oh15 (c) tr11 (d) tr21 (e) re0 (f) re1.

low: this means that, given a dataset, our approach converges always to very similar solutions, in spite of its stochasticity.

5.7 Evaluation of $HiCC_{incr}$

We showed that the results of the co-clustering algorithm on an initial portion of the data, can be used as the initial splitting function for an incremental approach. This strategy can be more generally be applied to an incremental framework without loss of quality. Now we focus on the overall hierarchical



(a)

(b)

(c)

(d)

(e)

(f)

Fig. 4 Normalized Mutual Information w.r.t. the percentage of the first block (using class labels) for (a) oh0 (b) oh15 (c) tr11 (d) tr21 (e) re0 (f) re1.

co-clustering procedure, and show that the resulting hierarchies are close to those obtained by the off-line version. To quantify the quality of the produced hierarchies we use the method of CP/CV (described in Section 5.5). Here, we compare the hierarchies generated by $HiCC_{incr}$ with the hierarchies generated by *HiCC*. To perform this analysis, we split the original dataset into multiple blocks (we let vary the number of blocks, from 5 to 20 per dataset).

Each dataset is handled as follows: 1) the first block is processed by *HiCC*; 2) the following block is processed by $HiCC_{incr}$ using the result on the first block as starting point; 3) the process is iterated incrementally for each of the remaining blocks. To obtain statistically relevant results we launch each algo-

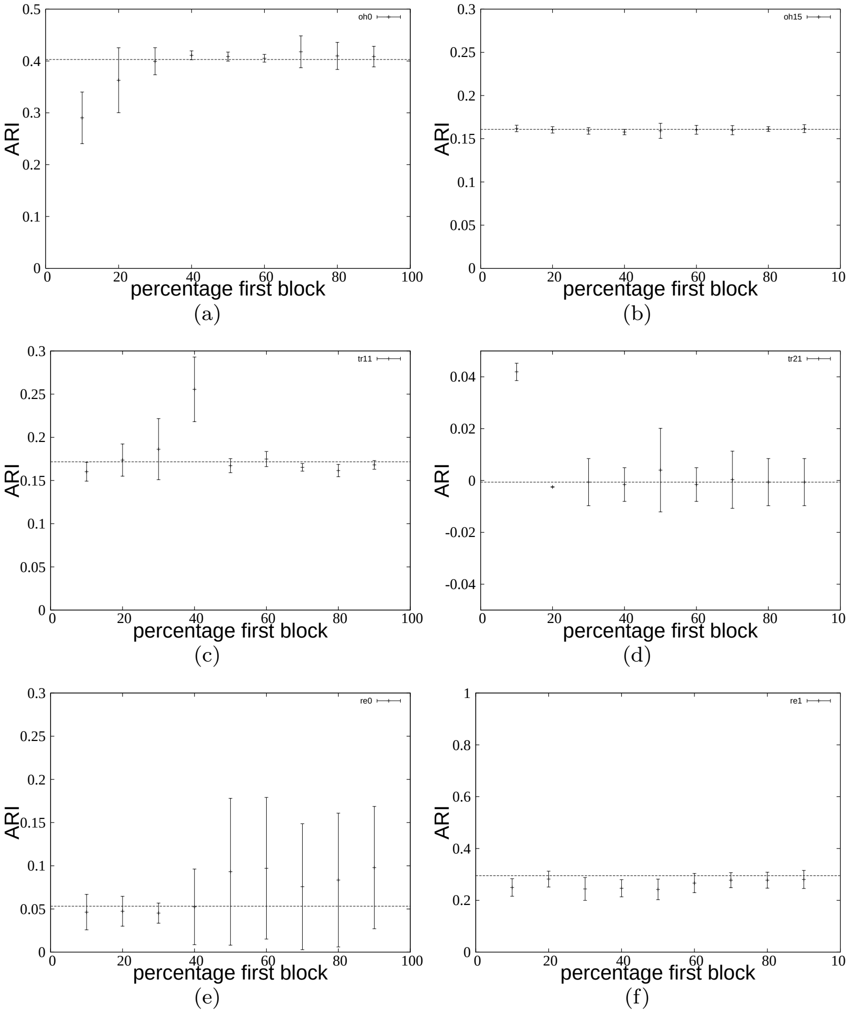
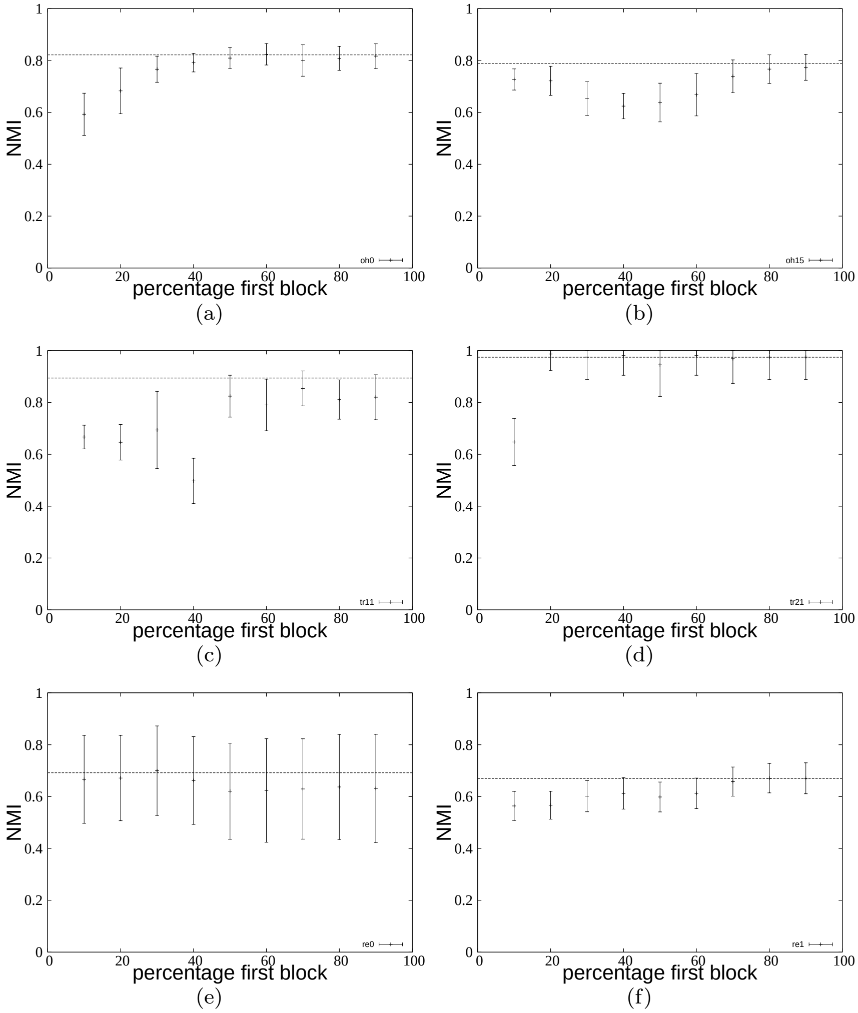


Fig. 5 Adjusted Rand Index w.r.t. the percentage of the first block (using class labels) for (a) oh0 (b) oh15 (c) tr11 (d) tr21 (e) re0 (f) re1.

rithm 50 times. Afterwards we compute the cosine distance between the root vectors representing the hierarchies obtained by *HiCC* and the root vectors of the hierarchies obtained from $HiCC_{incr}$. In Figure 8 we report the average and the standard deviation of the cosine distance over the 50 runs. We can notice that all the distances are below 0.1. However, in two cases, they are really very low (below 0.01; we recall here that the cosine distance takes values between 0 and 1). This means that the incremental version of our algorithm does not change substantially the returned hierarchies.



(a) (b) (c) (d) (e) (f)

Fig. 6 Normalized Mutual Information w.r.t. the percentage of the first block (using off-line results as reference) for (a) oh0 (b) oh15 (c) tr11 (d) tr21 (e) re0 (f) re1.

6 Related work

One of the earliest co-clustering formulations was introduced by Hartigan [16]. This algorithm begins with the entire data in a single block and then at each stage finds the row or column split of every block into two pieces, choosing the one that produces the largest reduction in the total within block variance. The splitting is continued till the reduction of within block variance due to further splitting is less than a given threshold. This approach is clearly hierarchical, but it does not build any cluster hierarchy. Moreover, it does not optimize any global objective function. Kluger et al. [25] propose a spectral co-clustering

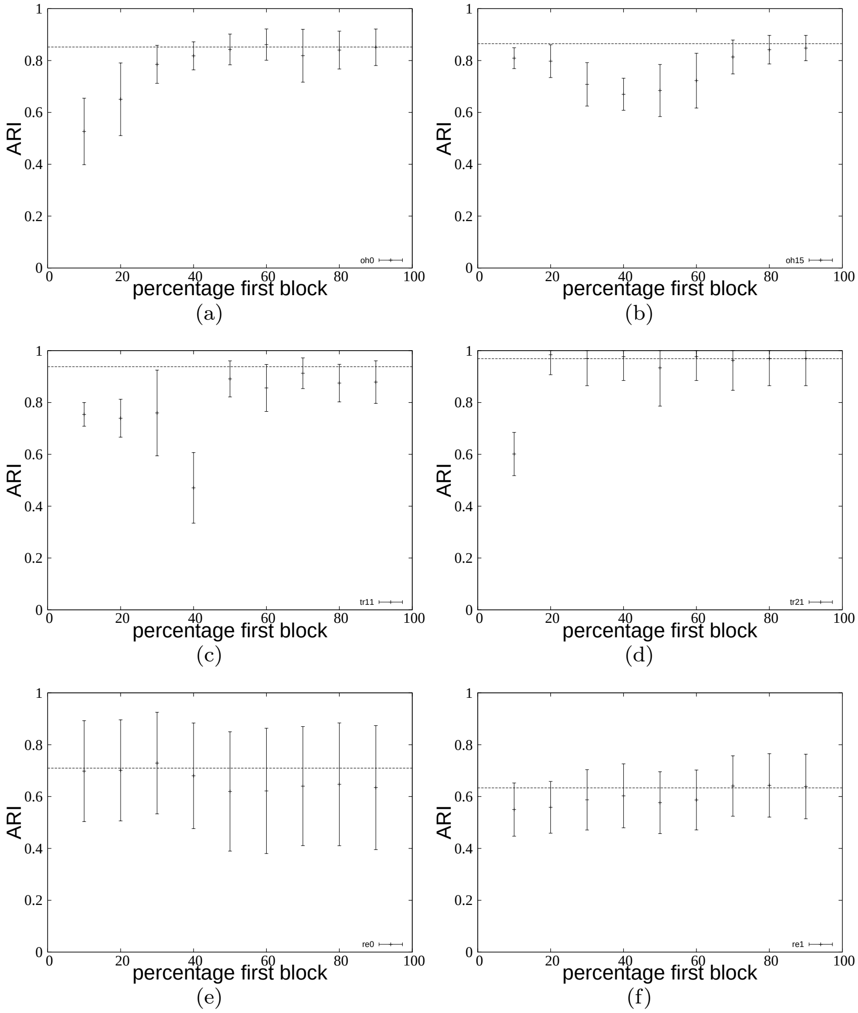


Fig. 7 Adjusted Rand Index w.r.t. the percentage of the first block (using off-line results as reference) for (a) oh0 (b) oh15 (c) tr11 (d) tr21 (e) re0 (f) re1.

method. First, they perform an adequate normalization of the data set to accentuate co-clusters if they exist. Then, they consider that the correlation between two columns is better estimated by the expression level mean of each column w.r.t. a partition of the rows. The bipartition is computed by the algebraic eigenvalue decomposition of the normalized matrix. Their algorithm critically depends on the normalization procedure.

Dhillon et al. [7] and Robardet et al. [30] have considered the two searched partitions as discrete random variables whose association must be maximized. Different measures can be used. Whereas *ITCC* [7] uses the loss in mutual information, $\tau CoClust$ [30] uses Goodman-Kruskal's τ coefficient to evaluate

size(D)	oh0	oh15	tr11	tr21	re0	re1
10%	3.1±0.3	2.8±0.4	2.0±0.0	3.0±0.0	2.0±0.2	5.5±1.0
20%	3.7±0.5	2.2±0.4	2.0±0.0	2.0±0.0	2.2±0.4	6.1±1.0
30%	4.0±0.2	2.7±0.5	2.0±0.0	2.0±0.2	2.1±0.3	6.6±1.4
40%	4.0±0.0	2.6±0.5	2.0±0.0	2.0±0.1	2.5±0.5	5.8±1.2
50%	4.0±0.2	2.7±0.5	2.0±0.0	2.1±0.3	2.7±0.5	6.1±1.3
60%	4.0±0.2	2.5±0.6	2.0±0.0	2.0±0.1	2.5±0.5	5.8±1.3
70%	4.2±0.4	2.1±0.2	2.0±0.0	2.1±0.2	2.3±0.5	6.6±0.9
80%	4.1±0.4	2.1±0.2	2.0±0.0	2.0±0.2	2.3±0.5	7.4±1.0
90%	4.1±0.3	2.6±0.5	2.0±0.0	2.0±0.2	2.5±0.6	7.8±1.2
100%	4.1±0.3	2.4±0.5	2.0±0.0	2.0±0.2	2.2±0.5	7.9±1.1

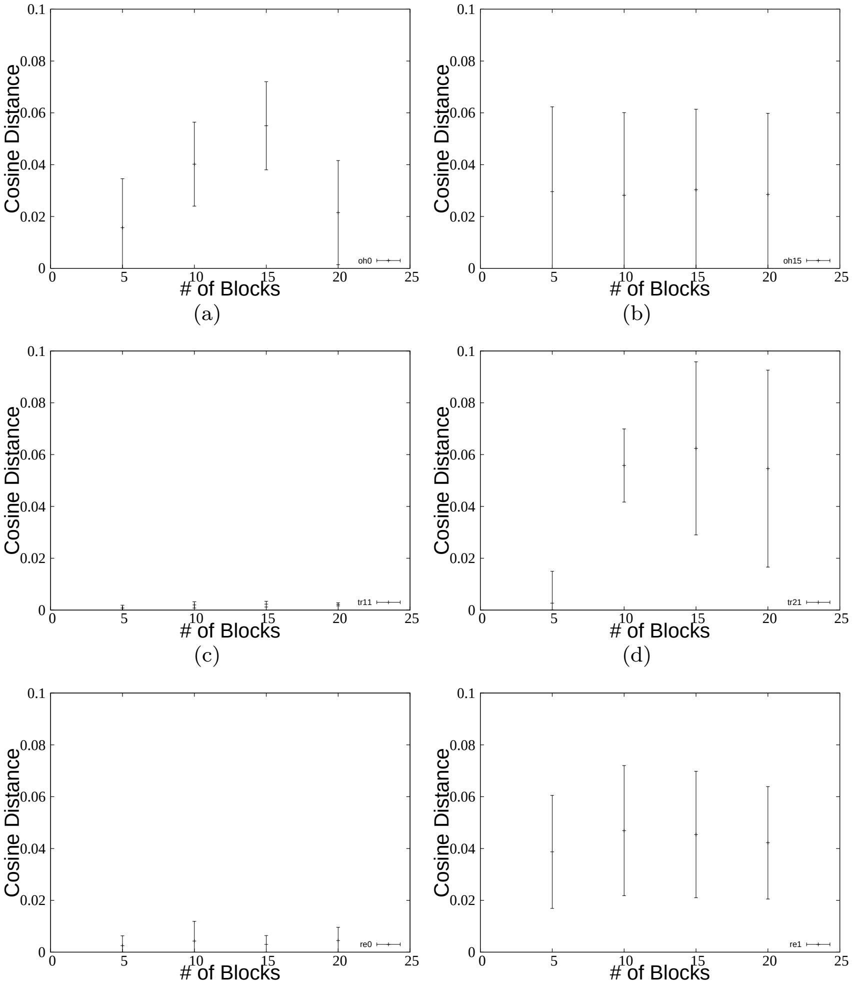
Table 10 Mean number of clusters for various initial block sizes

the link strength between the two variables. In both algorithms, a local optimization method is used to optimize the measure by alternatively changing a partition when the other one is fixed. The main difference between these two approaches is that the τ measure is independent of the number of co-clusters and thus *τCoClust* can automatically determine the number of co-clusters.

Another co-clustering formulation was presented in [4]. Authors propose two different residue measures, and introduce their co-clustering algorithm which optimizes the sum-squared residues function. Contrary to the above mentioned approaches, the data addressed by this work is not limited to co-occurrences data. However, both residue measures require the number of desired clusters to be specified as a parameter. In [3] the authors propose a fully automatic cross-association framework. The proposed algorithm consists in optimizing an entropy-based objective function. Like *τCoClust*, their approach is parameter free and it determines automatically the number of row clusters and column clusters. However, it is strongly limited by the fact that it can manage only binary matrices, while our proposed technique is designed to deal with both binary and counting/frequency data. In this paper, among other innovations, we have also considered a possible extension of *τCoClust* to generate hierarchies.

Recently, Banerjee et al. have proposed in [2] a co-clustering setting based on matrix approximation. The approximation error is measured using a large class of loss functions called Bregman divergences. They introduce a meta-algorithm whose special cases include the algorithms from [7] and [4]. Another recent and significant theoretical result has been presented in [1]. The authors show that the co-clustering problem is NP-hard, and propose a constant-factor approximation algorithm for any norm-based objective function.

[6] presents SCOAL, a different co-clustering approach aiming at learning a regression model dedicated to collaborative filtering applications. In this work a hierarchical clustering is associated to the co-clustering method. However hierarchies are only used to guide the model selection step. In particular, the improved version of SCOAL (M-SCOAL) uses a model selection technique,



(a) (b) (c) (d) (e) (f)

Fig. 8 Cosine distance to assess hierarchies varying the number of incremental blocks (a) oh0 (b) oh15 (c) tr11 (d) tr21 (e) re0 (f) re1.

based on a divisive hierarchical clustering, to choose an adequate number of clusters for rows and columns. In our work, instead, we force the algorithm to produce the whole hierarchy, until each leaf cluster is a singleton. Our final goal, in fact, is to provide a complete taxonomy that can be browsed by any expert. Any common model selection approaches [12,35] can be applied as post-processing to cut or reduce the depth of the hierarchy for visualization purposes. Another significant difference between the two approaches is that our hierarchy is n-ary, while M-SCOAL always splits each cluster into two sub-clusters.

To the best of our knowledge, our approach is the first one that performs a simultaneous hierarchical co-clustering on both dimensions, and that returns two coupled hierarchies. However, in recent literature, several approaches have been proposed that could be related to our work, even though they do not produce the same type of results. In [18] a hierarchical co-clustering for queries and URLs of a search engine log is introduced. This method first constructs a bipartite graph for queries and visited URLs, and then all queries and related URLs are projected in a reduced dimensional space by applying singular value decomposition. Finally, all connected components are iteratively clustered using *k-means* for constructing a hierarchical categorization. In [5], the authors propose a hierarchical, model-based co-clustering framework that views a binary dataset as a joint probability distribution over row and column variables. Their approach starts by clustering tuples in a dataset, where each cluster is characterized by a different probability distribution. Then, the conditional distribution of attributes over tuples is exploited to discover natural co-clusters in the data. This method does not construct any coupled hierarchy; moreover, co-clusters are identified in a separate step, only after the set of tuples has been partitioned. In [17], the proposed method constructs two hierarchies on gene expression data, but they are not generated simultaneously. In our approach, levels of the two hierarchies are alternately generated, so that each level of both hierarchies identifies a strongly related set of co-clusters of the matrix.

Incremental clustering is an old and well studied problem in data mining [8]. However, only few works address the problem of incremental co-clustering. In [11] the authors extend the co-clustering approach described in [2] in a straightforward way: they add the new object to a temporary cluster and then update the clustering statistics by assigning the new objects to existing clusters. The assumption here is that the number of co-clusters is always fixed, while our approach can possibly determine a different number of clusters. This work has been recently improved by [23] in which the authors suggest to estimate the clusters of the new incoming objects as soon as they arrive. They also propose an ensemble method for combining multiple local co-clustering results (with different number of clusters). Using this setting, however, the difference in term of computational time is not that marked between the off-line and the on-line method. Both works have been applied to the area of collaborative filtering methods for recommender systems, where the matrix encodes users that rate some items. In this case, the incremental version may be also useful when existing users add new ratings for existing or new items. However, none of these two methods deals with hierarchies. An attempt of hierarchical clustering for text documents is [31], where the authors extends COBWEB [9], to deal with probability distribution of word occurrences in text documents. The authors first show that COBWEB is not suitable for this kind of data, and propose a variant which takes into account the word occurrence distribution. This algorithm, however, does not build hierarchies over the feature space.

7 Conclusion

Quality of flat clustering solutions in high-dimensional data results often degraded. In this paper we have proposed both a hierarchical co-clustering approach and its extension to an incremental setting. *HiCC* is a novel hierarchical algorithm, which builds two coupled hierarchies, one on the objects and one on features thus providing insights on both them. Hierarchies are high quality. We have validated them by objectives evaluation measures like NMI and Adjusted Rand Index on many high-dimensional datasets. In addition, *HiCC* has other benefits: it is parameter-less; it does not require a pre-specified number of clusters; it produces compact hierarchies because it makes $n-$ary splits, with n automatically determined. We empirically demonstrate that the concept hierarchies produced by our approach are stable w.r.t. the space given by the original class labels. As a second important result we have extended *HiCC* to an incremental setting. We have shown that the incremental variant produces good hierarchical co-clusterings, similar to those provided by the off-line approach, but requires significantly less computational time. This observation opens the way to a novel usage of our algorithms, in an incremental setting, in which the data are partitioned into several blocks and they are incrementally processed. Here, we conducted many experiments on text data. However our algorithm could be applied to other kind of data as well. For instance, in gene expression data analysis, biologists usually employ hierarchical clustering techniques for exploring data in both senses, but they usually cluster genes and samples separately, often leading to uncorrelated results. In the future, we will investigate new applications of our hierarchical co-clustering techniques in life science domains, such as genomics, proteomics and phylogenetics. Furthermore, we will study alternative measures that can be applied directly on data other than co-occurrence tables.

Acknowledgements We thank Céline Robardet for her advice on the Pareto dominance criterium for the local convergence of the algorithms.

References

1. Anagnostopoulos, A., Dasgupta, A., Kumar, R.: Approximation algorithms for co-clustering. In: Proceedings of PODS 2008, pp. 201–210. ACM Press, Vancouver, BC, Canada (2008)
2. Banerjee, A., Dhillon, I., Ghosh, J., Merugu, S., Modha, D.: A generalized maximum entropy approach to bregman co-clustering and matrix approximation. Journal of Machine Learning Research **8**, 1919–1986 (2007)
3. Chakrabarti, D., Papadimitriou, S., Modha, D.S., Faloutsos, C.: Fully automatic cross-associations. In: Proc. ACM SIGKDD 2004, pp. 79–88. ACM Press, Seattle, USA (2004)
4. Cho, H., Dhillon, I.S., Guan, Y., Sra, S.: Minimum sum-squared residue co-clustering of gene expression data. In: Proceedings of SIAM SDM 2004. Lake Buena Vista, USA (2004)
5. Costa, G., Manco, G., Ortale, R.: A hierarchical model-based approach to co-clustering high-dimensional data. In: Proceedings of ACM SAC 2008, pp. 886–890. ACM Press, Fortaleza, Ceara, Brazil (2008)

6. Deodhar, M., Ghosh, J.: SCOAL: A framework for simultaneous co-clustering and learning from complex data. Trans. Knowl. Discov. Data **4**(3), 11:1–11:31 (2010)
7. Dhillon, I.S., Mallela, S., Modha, D.S.: Information-theoretic co-clustering. In: Proceedings of ACM SIGKDD 2003, pp. 89–98. ACM Press, Washington, USA (2003)
8. Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X.: Incremental clustering for mining in a data warehousing environment. In: Proceedings of VLDB 1998, pp. 323–333. Morgan Kaufmann, New York City, New York, USA (1998)
9. Fisher, D.: Knowledge acquisition via incremental conceptual clustering. Machine Learning **2**(2), 139–172 (1987)
10. Forman, G.: An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research **3**, 1289–1305 (2003)
11. George, T., Merugu, S.: A scalable collaborative filtering framework based on co-clustering. In: Proceedings of ICDM 2005, pp. 625–628. IEEE Computer Society, Houston, Texas, USA (2005)
12. Gong, J., Oard, D.: Selecting hierarchical clustering cut points for web person-name disambiguation. In: Proceedings of ACM SIGIR 2009, pp. 778–779. ACM Press, Boston, MA, USA (2009)
13. Goodman, L.A., Kruskal, W.H.: Measures of association for cross classification. Journal of the American Statistical Association **49**, 732–764 (1954)
14. Goodman, L.A., Kruskal, W.H.: Measure of association for cross classification ii: further discussion and references. Journal of the American Statistical Association **54**, 123–163 (1959)
15. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann (2000)
16. Hartigan, J.A.: Direct clustering of a data matrix. Journal of the American Statistical Association **67**(337), 123–129 (1972)
17. Heard, N.A., Holmes, C.C., Stephens, D.A., Hand, D.J., Dimopoulos, G.: Bayesian coclustering of anopheles gene expression time series: Study of immune defense response to multiple experimental challenges. Proc. Natl. Acad. Sci. **102**(47), 16,939–16,944 (2005)
18. Hosseini, M., Abolhassani, H.: Hierarchical co-clustering for web queries and selected urls. In: Proceedings of WISE 2007, *LNCS*, vol. 4831, pp. 653–662. Springer, Nancy, France (2007)
19. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification **2**(1), 193–218 (1985)
20. Ienco, D., Pensa, R., Meo, R.: Parameter-free hierarchical co-clustering by n-ary splits. In: Proceedings of ECML PKDD 2009, Part I, *Lecture Notes in Computer Science*, vol. 5781, pp. 580–595. Springer, Bled, Slovenia (2009)
21. Ienco, D., Robardet, C., Pensa, R., Meo, R.: Parameter-less co-clustering for star-structured heterogeneous data. Data Mining and Knowledge Discovery pp. 1–38 (2012). Published online
22. Jaszkiewicz, A.: Genetic local search for multi-objective combinatorial optimization. European Journal of Operational Research **137**(1), 50–71 (2002)
23. Khoshneshin, M., Street, W.: Incremental collaborative filtering via evolutionary co-clustering. In: Proceedings of ACM RecSys 2010, pp. 325–328. ACM Press, Barcelona, Spain (2010)
24. Kim, J., Candan, K.: CP/CV: concept similarity mining without frequency information from domain describing taxonomies. In: Proceedings of ACM CIKM 2006, pp. 483–492. ACM Press, Arlington, Virginia, USA (2006)
25. Kluger, Y., Basri, R., Chang, J., Gerstein, M.: Spectral biclustering of microarray data: coclustering genes and conditions. Genome Research **13**, 703–716 (2003)
26. Kriegel, H.P., Kröger, P., Zimek, A.: Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Trans. Knowl. Disc. Data **3**(1), 1:1–1:58 (2009)
27. Paquete, L., Stützle, T.: A study of stochastic local search algorithms for the biobjective qap with correlated flow matrices. European Journal of Operational Research **169**(3), 943 – 959 (2006)
28. Robardet, C.: Contribution à la classification non supervisée: proposition d'une methode de bi-partitionnement. Ph.D. thesis, Université Claude Bernard - Lyon 1 (2002)

29. Robardet, C., Feschet, F.: Comparison of three objective functions for conceptual clustering. In: Proceedings of PKDD 2001, *LNCS*, vol. 2168, pp. 399–410. Springer, Freiburg, Germany (2001)
30. Robardet, C., Feschet, F.: Efficient local search in conceptual clustering. In: Proceedings of DS 2001, *LNCS*, vol. 2226, pp. 323–335. Springer, Washington, DC, USA (2001)
31. Sahoo, N., Callan, J., Krishnan, R., Duncan, G., Padman, R.: Incremental hierarchical clustering of text documents. In: Proceedings of ACM CIKM 2006, pp. 357–366. ACM Press, Arlington, Virginia, USA (2006)
32. Slonim, N., Tishby, N.: Document clustering using word clusters via the information bottleneck method. In: Proceedings of ACM SIGIR 2000, pp. 208–215. ACM Press, Athens, Greece (2000)
33. Strehl, A., Ghosh, J.: Cluster ensembles — a knowledge reuse framework for combining multiple partitions. Journal of Machine Learning Research **3**, 583–617 (2002)
34. V.S.Cherukuri, Candan, K.: Propagation-vectors for trees (pvt): concise yet effective summaries for hierarchical data and trees. In: Proceeding of LSDS-IR 2008, pp. 3–10. ACM Press, Napa Valley, California, USA (2008)
35. Zhang, Y., Li, T.: Extending consensus clustering to explore multiple clustering views. In: Proceedings of SIAM SDM 2011, pp. 920–931 (2011)