Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Scalable Parallel Co-Clustering Over Multiple Heterogeneous Data Types

Francesco Folino, Gianluigi Greco,
Antonella Guzzo, Luigi Pontieri

RT-ICAR-CS-09-10 **Ottobre 2009**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Scalable Parallel Co-Clustering Over Multiple Heterogeneous Data Types

Francesco Folino[1], Gianluigi Greco[2], Antonella Guzzo[3], Luigi Pontieri[1]

[1] ICAR-CNR, National Research Council, via P. Bucci 41C, 87036, Rende, CS, Italy
[2]Dept. of Mathematics, University of Calabria, Italy
[3]Dept. DEIS, University of Calabria, Italy
{ffolino, pontieri}@icar.cnr.it, ggreco@mat.unical.it,
guzzo@deis.unical.it

Abstract. *The bi-clustering, i.e., simultaneously clustering two types of objects based on their correlations, has been studied actively in the last few years, in virtue of its impact on several relevant applications, such as text mining, collaborative filtering, gene expression analysis. In particular, many research efforts were recently spent on extending such a problem towards higher-order scenarios, where more than two data types are to be clustered synergically, according to pairwise inter-type relations. Measuring co-clustering quality as a weighted combination of the distortions over input relations, a number of alternate-optimization methods were developed of late, which scale linearly with the size of data. This result is likely to be inadequate for large scale applications where massive volumes of data are involved, and high performance solutions would be desirable. However, to date, parallel clustering approaches have been investigated deeply only for the case of just one or two inter-related data types. In this paper, we face the more general (high-order) co-clustering problem by proposing a parallel implementation of an effective and state-of-the-art method, by leveraging a parallel computation infrastructure implementing popular Map-Reduce paradigm.*

1 Introduction

The problem of simultaneously clustering two heterogeneous types of objects based on their correlations (known as co-clustering or bi-clustering), was thoroughly studied in several fields of research such as text mining, collaborative filtering and gene expression analysis, where it is crucial to group objects together based on their similarity over a suitable subset of features. Many recent proposals tried to extend this problem towards higher-order scenarios, where more than two data types are to be clustered synergically [4, 10, 11, 2]. Most of these efforts focus on the case where the domains are correlated in a pairwise manner, and face an optimization problem where: *(i)* the quality of the clustering along each pair of correlated domains is assessed via some standard bi-clustering objective function (e.g., the loss in mutual information), and *(ii)* the quality of the "global" co-clustering is evaluated as a linear combination of these functions. Based on such a formulation, a number of similar alternate-optimization schemes were proposed, which allows to compute a solution efficiently, and scale nearly linearly with the size of data, under the realistic assumption that a constant number of optimization

steps are sufficient to reach a local optimum. Conversely, the method in [10] faces the problem of automatically weighting the different bi-clustering sub-tasks (i.e. one for each contingency table), based on their capability to agree with each other without diverging too much from their optimal solution. Notably, even this self-tuning algorithm ensures linear scaling with the size of data.

Yet, linear scalability still risks being inadequate for analyzing large collections of data, which are becoming more and more common, due to the high volumes and variety of data characterizing certain application contexts (e.g., world-wide web-based systems, or biological repositories). Hence, resorting to some parallel computation scheme seems to be the only solution that can truly satisfy such a need of high-performance (high-order) co-clustering tools.

However, so far, a number of works applied some parallel computation only for clustering single-/two- type data, some of which exploit the popular computing paradigm Map-Reduce (e.g., [8] and [12]). The only parallelized co-clustering approach dealing with more than two domains seems to be [3], which simultaneously clusters multiple pairwise-correlated data types, in a way that minimizes the loss of mutual information among all the domains, by iteratively refining the clusters for a pair of domains per time. However, a major limitation of this method is that it does not allow to appropriately weight the relevance/reliability of the pairwise relationships either manually (e.g., as in [4, 11, 2]) or automatically (as in [10]). In conclusion, the parallelization of high order co-clustering algorithms is still an open field of research, especially for the case of information-theoretic co-clustering.

Contribution and Organization. The main goal of this paper is just to go further along this research by devising a parallelized efficient solution to the high-order co-clustering problem. Specifically, we extend and refine the approach in [10] in order to compute a co-clustering solution in a parallel manner, on the top of a Map-Reduce infrastructure. To this end, we take advantage of the open-source system *Hadoop* [1], providing an implementation of core Map-Reduce functionalities. Moreover, we study the performances of our approach from both an analytical and empirical viewpoint.

The rest of the paper is organized as follows. In Section 2 we introduce the high-order co-clustering information-theoretic framework, and propose a meta-algorithm allowing to compute a locally-optimal solution via *alternate-optimization* steps. Section 3 shows a parallel implementation scheme, and studies its scalability from a theoretical viewpoint. An empirical performances' analysis is given in Section 4, based tests conducted against both real and synthesized data, with the Hadoop-based implementation of the approach. Finally, Section 5 summarizes our work and traces a few lines of further research.

2 Formal Framework

We next describe an information-theoretic framework, originally introduced in [10], for co-clustering heterogeneous objects belonging to an arbitrary number of domains, forming a star structure of interrelationships.

Let $D_{Y^1} = \{y_1^1, \ldots, y_{n_1}^1\}, \ldots, D_{Y^N} = \{y_1^N, \ldots, y_{n_N}^N\}$, $D_X = \{x_1, \ldots, x_m\}$ be $N+1$ domains (i.e. sets of values), and let $Y^1, \ldots, Y^N$,X be discrete random variables

taking values in $D_{Y^1}, \ldots, D_{Y^n}$,D_X, resp. Each "auxiliary" domain D_{Y^i}, with $1 \leq i \leq N$, is only linked to the "central" one D_X, according to a star structure. The correlations between elements from D_X and D_{Y^i} are stored in a contingency matrix P_i, which can be modeled as a joint probability distribution $p_i(X, Y^i)$ between the random variables X and Y^i — $p_i(x, y^i_j)$ is the probability that X and Y^i take the values $x \in D_X$ *and* $y^i_j \in D_{Y^i}$, respectively.

Assume that the values in D_X are to be clustered into k clusters, say $\widehat{D}_X = \{\widehat{x}_1, \widehat{x}_2, ..., \widehat{x}_k\}$, and those in D_{Y^i} are to be clustered in l_i clusters, say $\widehat{D}_{Y^i} = \{\widehat{y}^i_1, \widehat{y}^i_2, ..., \widehat{y}^i_{l_i}\}$, for each $1 \leq i \leq N$. Then, a *high-order co-clustering* for $Y^1, ..., Y^N$ w.r.t. X is a tuple $C = \langle C_X, C_{Y^1}, ..., C_{Y^N} \rangle$, such that $C_X : D_X \mapsto \widehat{D}_X$, and $C_Y : D_{Y^i} \mapsto \widehat{D}_{Y^i}$, for $i = 1 \ldots N$.

Like in [6], for any domain, lower-case letters denote its elements, and upper-case ones denote the random variable ranging over them, with clusters and clustered random variables denoted by hatted letters.

The quality of a co-clustering solution can be measured as the loss in mutual information that occurs when the variables X, $Y^1, ..., Y^N$ are replaced with their respective clustered versions $\widehat{X}, \widehat{Y^1}, ..., \widehat{Y^N}$. In particular, the loss on the i-th contingency matrix is denoted by $\Delta I_i(C_X, C_{Y^i}) = I(X; Y^i) - I(\widehat{X}; \widehat{Y^i})$ (shortly, ΔI_i). As stated in [6], this loss can be expressed as a dissimilarity between the original joint distribution $p_i(\cdot, \cdot)$ and a function $q_i(\cdot, \cdot)$ that approximates it. More specifically, $\Delta I_i = \mathcal{D}(p_i(X, Y^i) || q_i(X, Y^i))$, where $\mathcal{D}(\cdot||\cdot)$ denotes the well-known Kullback-Leibler (KL) divergence, and $q_i(X, Y^i)$ is a function of the form:

$$q_i(X, Y^i) = p_i(\widehat{X}, \widehat{Y}^i) \cdot p_i(X|\widehat{X}) \cdot p_i(Y|\widehat{Y}^i) \tag{1}$$

which preserves all marginals of p_i, and is univocally determined by the co-clustering at hand.

When $N = 1$, the bi-clustering problem turns into searching for the function q_i that is the most similar to p_i, according to $\mathcal{D}$. This problem can be solved via the alternate minimization scheme of [6], where, at each iteration, each element $x \in D_X$ (resp., $y \in D_{Y^i}$) is greedily assigned to the cluster $\widehat{x}^*$ (resp. $\widehat{y}^*$) that minimizes the point-wise loss measure $\delta_i(x, \widehat{x})$ (resp., $\delta_i(y, \widehat{y})$) below:

$$\delta_i(x, \widehat{x}) = \mathcal{D}(\ p_i(Y^i|x)\ ||\ q_i(Y^i|\widehat{x})\ ) \tag{2}$$

$$\delta_i(y, \widehat{y}) = \mathcal{D}(\ p_i(X|y)\ ||\ q_i(X|\widehat{y})\ ) \tag{3}$$

This method is guaranteed to monotonically decrease the pairwise information loss, since the loss function ΔI_i can be computed by summing up the above point-wise losses over all elements of either D_X or D_{Y^i}, and that these losses are all independent of each others, i.e.: $\Delta I_i = \sum_{y \in D_{Y^i}} \delta_i(y, C_{Y^i}(y)) = \sum_{x \in D_X} \delta_i(x, C_X(x))$.

Figure 1 shows a meta-algorithm, named $HOCC$, that extends the alternate minimization method of [6] to an arbitrary number of star-structured domains. Given the domains D_X, D_{Y^1},..., D_{Y^N}, the cluster sets $\widehat{D}_X$, $\widehat{D}_{Y^1}$,..., $\widehat{D}_{Y^N}$ (specifying the number of clusters required for each dimension), and the joint probability distributions $p_1(X, Y^1), ..., p_N(X, Y^N)$, an initial co-clustering is computed, and refined iteratively,

Input: Domains D_X (central), D_{Y^1},...,D_{Y^N}, cluster sets $\widehat{D}_X$, $\widehat{D}_{Y^1}$,...,$\widehat{D}_{Y^N}$, and joint distributions $p_1(X, Y^1), ..., p_N(X, Y^N)$;
Output: A co-clustering for X, Y^1,...,Y^N;

1. Set an arbitrary initial co-clustering $\langle C_X, C_{Y^1}, ..., C_{Y^N} \rangle$, and all global variables;
2. Compute q_i's parameters, for $i = 1..N$;
3. $stop$ = **false**;
4. **do**
   // Compute C'_{Y^i} based on $q_i(X|\widehat{Y})$), for $i = 1..N$
5. **for each** Y^i and $y \in D_{Y^i}$ **do**
6. $C'_{Y^i}(y) = \arg\min_{\widehat{y} \in \widehat{D}_{Y^i}} \delta_i(y, \widehat{y})$;
7. Update q_i's parameters, for $i = 1..N$, and global variables;
8. **for each** $x \in D_X$ **do**
9. Compute $C'_X(x)$, based on $q_i(Y^i|\widehat{X})$), for $i = 1..N$;
10. Update q_i's parameters, for $i = 1..N$ and globals variables;
11. $stop$ = `testExitCondition();`
12. **if not** $stop$ **then** set $C_X = C'_X$, $C_{Y^i} = C'_{Y^i}$, for $i = 1..N$;
13. **until** $stop$;
14. **return** $\langle C_X, C_{Y^1}, ..., C_{Y^N} \rangle$;

Fig. 1. Meta-Algorithm $HOCC$.

in an alternate manner. At each repetition of the main loop, first a new clustering C'_{Y^i} is computed for each auxiliary domain Y^i (step 6), based on its associated distribution q_i (which depends on the previous clusterings $C_X, C_{Y^i}, .., C_{Y^N}$). Since each auxiliary dimension Y^i is independent of the others, any of its elements $y \in D_{Y^i}$ can be greedily assigned to the cluster $\widehat{y}$ that minimizes the score $\delta_i(y, \widehat{y})$, i.e. $C'_{Y^i}(y) = \arg\min_{\widehat{y} \in \widehat{D}_{Y^i}} \delta_i(y, \widehat{y})$. Notably, these scores are fully specified by the current distribution q_i. Modifying the clusters of each auxiliary domain D_{Y^i} now requires to update the parameters of distribution q_i. Subsequently, the algorithm updates the clustering C'_X over the central domain D_X, based on all q_i. Again, the parameters of all distributions q_i are updated accordingly.

The computation scheme sketched above can accommodate a number of specific information-theoretic algorithms, including those (namely `LC-HOCC` and `AD-HOCC`) proposed in [10], which both deal with the problem of setting the importance of the joint probability matrices p_i in a different way. This can be accomplished by customizing the generic steps 9 and 11, and by suitably defining and handling a number of auxiliary global variables, as discussed next.

2.1 Two concrete algorithms

LC-HOCC. Algorithm `LC-HOCC` [10] is essentially addressed to minimizing a linear combination of the pairwise information losses ΔI_i, for $1 \leq i \leq N$. Similarly to [4, 11, 2], N real numbers $\beta_1, ..., \beta_N$, s.t. $\sum_{i=1}^{N} \beta_i = 1$, are associated with the joint

distributions p_1, .., p_N, respectively. Then, a (locally) optimal co-clustering is looked for, which minimizes the quantity $\sum_{i=1}^{N} \beta_i \cdot \Delta I_i$.

Updating X's clusters (step 9). In this case, the best cluster for any element $x \in D_X$ is simply chosen as follows: $C'_X(x) = \arg\min_{\widehat{x} \in \widehat{D}_X} \sum_{i=1}^{N} \beta_i \cdot \delta_i(x, \widehat{x})$. Note that this will never increase the (global) objective function $\sum_{i=1}^{N} \beta_i \cdot \Delta I_i$.

Exit condition (step 11). Since the updating of clusters in `LC-HOCC` ensures the monotonicity of the global objective function $\sum_{i=1}^{N} \beta_i \cdot \Delta I_i$, it is enough to check the convergence around local optima. Notice that no auxiliary global variable is needed, since the convergence test can be done by simply verifying whether at least one element really moved from a cluster to another.

AD-HOCC. Rather than using an a priori weighting scheme, this method tries to find an "agreement" among the different (and potentially discordant) loss functions. Essentially, sub-optimal values for some information losses are tolerated, provided that a bound α on the distance from their respective optimal values is ensured. Specifically, α is an estimated lower bound for all the quantities $\Delta I_i^* / \Delta I_i$, where ΔI_i is the loss produced by the the solution of D_{Y^i}, while ΔI_i^* is the minimal loss, which would be obtained when co-clustering D_X and D_{Y^i} only.

Updating X's clusters (step 9). The clustering C_X of the central domain must be updated in a way that guarantees a maximal value of the approximation bound α discussed above. First, for each element $x \in D_X$ all point-to-cluster losses δ_i over each domain D_{Y^i} are normalized w.r.t. the best possible assignment of x in that domain (i.e., $\min_{\widehat{x}' \in \widehat{D}_X} \delta(x, \widehat{x}')$). Eventually, x is assigned to the cluster guaranteeing the maximal approximation bound $\alpha(x, \widehat{x})$, i.e.,: $C'_X(x) = \arg\max_{\widehat{x} \in \widehat{D}_X} \alpha(x, \widehat{x})$.

Exit condition (step 11). Since the updating of X's clusters does not enjoy any monotonicity property with regard to the information loss functions, some specific termination criterion must be defined. A possible strategy consists in iterating as long as the current agreement can be improved. To this end, a global approximation bound α is computed after each updating of X's cluster, based on the point-wise scores $\alpha(x, \widehat{x})$, as follows: $\alpha = \min_{x \in D_X} \max_{\widehat{x} \in \widehat{D}_X} \alpha(x, \widehat{x})$. The exit condition mainly amounts to verify that the new bound is higher than the one computed in the previous iteration of the main loop — two further global variables α and α' are then needed in order to make such a comparison. By the way, to reduce the risk of converging into poorly accurate solutions, a tolerance factor can be introduced. Further details on this issue can be found in[10].

3 Parallel Implementation

In order to devise a parallel version of meta-algorithm $HOCC$, we observe that three main kinds of operations recur in each iteration of the main loop: *(a)* updating the distributions q_i, for $i = 1..N$, *(b)* re-clustering all objects y of any auxiliary domain D_{Y^i}, based on q_i and on the (column) distribution vectors $p_i(X|y)$, and *(c)* re-clustering all objects x of central domain D_X, based on q_i and on the (row) distribution vectors $p_i(Y|x)$.

Since any $x \in D_X$ (resp., $y \in D_{Y^i}$) can be clustered independently of all other elements in D_X (resp., in D_{Y^j}, for $j = 1..N$), steps *(b)* and *(c)* can proceed in parallel over row/ column elements, if row-wise column-wise distribution vectors are assigned to different processors.

On the other hand, for any $i = 1..N$, $x \in D_X$ and $y \in D_{Y^i}$, $q_i(x, y)$ can be reconstructed based on: *(i)* the information about the clusters $\widehat{x}$ and $\widehat{y}$ to which x and y, respectively, were assigned in the previous iteration (i.e. $\widehat{x} = C_X(x)$ and $\widehat{y} = C_{Y^i}(y)$), and their associated *(ii)* mutual joint probability $p_i(\widehat{x}, \widehat{y})$ and *(iii)* marginals $p_i(\widehat{x})$ and $p_i(\widehat{y})$. Notably, the two latter pieces of information can be derived from the cluster-wise joint distribution $p_i(\widehat{X}, \widehat{Y})$) —with $\widehat{X}$ and $\widehat{Y}$ ranging on D_X and D_{Y^i}, respectively. Moreover, for any pair of clusters $\widehat{x}_r \in \widehat{D}_X$ and $\widehat{y}_c \in \widehat{D}_{Y^i}$, the probability $p_i(\widehat{x}_r, \widehat{y}_c)$ can be computed by summing up all of its associated element-wise joint probabilities. i.e. $p_i(\widehat{x}_r, \widehat{y}_c) = \sum_{\{(x,y)|C_X(x)=\widehat{x}_r, C_{Y^i}(y)=\widehat{y}_c\}} p_i(x, y)$.

Thus, each processor can locally compute the new clusters for all the elements of D_X,$D_{Y^1}, \ldots, D_{Y^N}$ that were assigned to it, as well as their partial contribution to the corresponding cluster-wise joint distributions $p_1(\widehat{X}, \widehat{Y}), \ldots, p_N(\widehat{X}, \widehat{Y})$. The overall clustering functions $C_X, C_{Y^1}, \ldots, C_{Y^N}$ and the overall matrices $p_1(\widehat{X}, \widehat{Y}), \ldots, p_N(\widehat{X}, \widehat{Y})$ are obtained in a subsequent synchronization step, were the partial contributions provided by all processors are merged together.

Complexity analysis. Let us study the computational costs of the meta-algorithm in Figure 1. For the sake of simplicity, some upper bounds are assumed to be known for the different parameters characterizing the size of input data — which consisting of $N + 1$ domains (i.e., D_X,D_{Y^1},..,D_{Y^N}) and N pairwise correlation matrices P_1,..., P_N (each of them encoding a joint probability distribution). Specifically, we assume that: *(i)* at most $\overline{n}$ elements appear in each domain, *(ii)* at most $\overline{k}$ clusters are to be found over each domain, and *(iii)* each contingency matrix contains at most $\overline{m}$ non-zero cells.

Let H be the number of nodes used in the computation, where each node $h = 1..H$ is provided with the (column-wise) correlation data corresponding to a fraction $1/H$ of the elements in $\bigcup_{i=1..N} D_{Y^i}$, and with all the (row-wise) correlation data corresponding to a fraction $1/H$ of the elements in D_X. Moreover, we assume that input data are distributed in a way that approximatively the same number of non-zero elements ($2N \times \overline{m}/H$) are given to each node.

Then, it can be shown that computing each iteration of the main loop in the $HOCC$ over H nodes, whatever of the two concrete implementation (i.e., `LC-HOCC` or `AD-HOCC`) is used, has the following cost (ignoring communication overhead and assuming that $\overline{m} \geq \overline{n}$):

$$O\left(\frac{N \times \overline{m} \times \overline{k}}{H} + H \times N \times \overline{k}^2\right)$$

where the term on the left accounts for parallel tasks, while the one on the right side concerns synchronized steps of the elaboration. Clearly, the overall computation will depend on the overall number of iterations. This latter, however, tends to stay under a couple of dozens, in most practical cases, as a large number of studies in the literature evidenced for different information-theoretic co-clustering methods. This result evidences the benefits of parallelization, and ensures a (theoretically) quasi-linear speed-

up, under the hypothesis that $H << \sqrt{\frac{\overline{m}}{\overline{k}}}$ — in fact this usually holds, as $\overline{m}$ is far bigger then the number of clusters one is expected to search for (i.e. $\overline{m} \geq \overline{n} >> \overline{k}$).

3.1 Implementation with Map-Reduce

Map-reduce [5] is quite an intuitive computation model over large clusters of machines, which somewhat takes inspiration from analogous primitives available in functional languages such as Lisp. As a major advantage provided to programmers, it allows to concentrate on key computational steps at a logical level, without considering low-level complicate issues, concerning, e.g., fault-tolerance, data distribution and load balancing. In particular, in our implementation we exploited the open-source *Hadoop*[1] framework, which offers basic Map-Reduce's functionalities, and provides an efficient distributed storage system, named $HDFS$, for storing input data and shared results.

Roughly, a Map-Reduce computation typically proceeds along the following phases: (i) *Split*, where the data are partitioned among different nodes; (ii) *Map*, where all input records are read concurrently, on different machines, and mapped to multiple (intermediate) key-value pairs; (iii) *Partition and Shuffle*, where the key-value pairs produced by the Map phase are locally grouped and sorted according to their keys; and (iv) *Reduce*, where each reducer node takes in input all the key-value pairs associated with the same intermediate key, and combine them into a new key-value pair.

We pinpoint that our implementation ensures that, for any element x of the central domain X, all of its joint probabilities (along all matrices p_i) are kept together in the split phase and sent to the same mapper. This choice clearly aims at reducing the amount of data exchanged between all the nodes. To the same end, the output of each mapper is forwarded to a *combiner* running on the same node, which applies the reduce procedure to local data.

We next illustrate how Map-Reduce paradigm was used to implement the co-clustering scheme in Figure 1, only focusing on the different pairs of Map and Reduce tasks that we defined. For the sake of conciseness, for each of such tasks we indicate structure of its input/output records, and assume that the first field (emphasized in bold letters) is the key of the record itself.

A. Data Preprocessing. In this phase all correlation data are processed in order to associate each element of any domain (i.e., D_X, D_{Y^1}, ...D_{Y^N}) with its non-zero joint probabilities, and with its marginal probability. The following records are obtained eventually: $\langle(\mathbf{x}, \mathbf{i}), \{(y, p_i(x, y))|y \in D_{Y^i}, p_i(x, y) > 0\}, p_i(x)\rangle$ for each $x \in D_X$. $\langle(\mathbf{y}, \mathbf{i}), \{(x, p_i(x, y))|x \in D_X, p_i(x, y) > 0\}, p_i(y)\rangle$ for $i = 1..N$ and for all $y \in D_{Y^i}$. This can be done in a parallel way, via a trivial map-reduce computation. Details are omitted for space lack.

B. Initializing co-clusters and distributions q_i. Via a simple map-reduce computation, we randomly set an initial co-clustering, and calculate its associated low-order statistics $p_i(\widehat{X}, \widehat{Y})$, for $i = 1..N$. Two kinds of mappers are used, for processing row-wise records (of the form $\langle(\mathbf{x}, \mathbf{i}), \{p_i(x, y)\}\rangle$) and column-wise records (of the form $\langle(\mathbf{y}, \mathbf{i}), \{p_i(x, y)\}\rangle$), respectively.

Map. Each mapper computes the new cluster for each row (resp., column) element $x \in D_X$ (resp. $y \in D_Y^i$) it receives in input, and produces a new record where the key is the ID of the cluster chosen. This intermediate key is associated with both a singleton set containing the element's ID (i.e. $\{x\}$ or $\{y\}$, resp.) and a set encoding all of its non-zero probabilities. Specifically, for any element $y \in D_{Y^i}$ (resp. $x \in D_X$), the mapper returns an intermediate record $\langle(\mathbf{i}, \widehat{\mathbf{y}}), \{y\}, \{(\widehat{x}, p_i(\widehat{x}|y)) \mid p_i(\widehat{x}|y) > 0\}\rangle$ (resp., $\langle\widehat{\mathbf{x}}, \{x\}, \{(i, \widehat{y}, p_i(\widehat{y}|x)) \mid p_i(\widehat{y}|x) > 0\}\rangle$) — where $p_i(\widehat{x}|y)$=$\sum_{x \in D_X} p_i(x|y)$ and $p_i(\widehat{y}|x)$=$\sum_{y \in D_{Y^i}} p_i(x|y)$.

Reduce. The reducers combine the records above by computing the union of the elements' ID associated with the same cluster, and by summing up the corresponding probabilities in a cluster-by-cluster way (for obtaining the sufficient statistics $p_i(\widehat{X}, \widehat{Y})$ for each q_i). As a final result a record $\langle\widehat{\mathbf{x}}, \{x \in D_X \mid C_X(x) = \widehat{x}\}, \{(\widehat{y}, p_i(\widehat{x}, \widehat{y})) \mid \widehat{y} \in \widehat{D}_{Y^i}, p_i(\widehat{x}, \widehat{y}) > 0\}\rangle$, is generated for each cluster $\widehat{x} \in \widehat{D}_X$. Similarly, a record $\langle(\mathbf{i}, \widehat{\mathbf{y}}), \{y \in D_{Y^i} \mid C_{Y^i}(y) = \widehat{y}\}, \{(\widehat{x}, p_i(\widehat{x}, \widehat{y})) \mid \widehat{x} \in \widehat{D}_X, p_i(\widehat{x}, \widehat{y}) > 0\}\rangle$, is produced for each cluster $\widehat{y} \in \widehat{D}_{Y^i}$ and for $i = 1..N$.

C. Updating the clustering of the auxiliary domains. In this phase, for any domain D_{Y^i} a new clustering C'_{Y^i} must be computed by assigning each object $y \in D_{Y^i}$ to its "closest" cluster in D_{Y^i}, according to the scores δ in Equation 3. Notice that this implies comparing each non-zero joint probability associated with y (in the column distribution $p_i(X|y)$) with the corresponding element in $q_i(X|y)$. As observed earlier, this latter distribution can be estimated "on-the-fly" based on the previous clusterings C_X and C_{Y^i} and on the associated statistics $p_i(\widehat{X}, \widehat{Y})$. These are indeed the only partial results that must be shared by the nodes involved in the co-clustering computation.

Map. Each mapper receives a record corresponding to an $y \in D_Y^i$ and encoding its column distribution $p_i(X|y)$, and locally decides which cluster $\widehat{y}$ it must be assigned to. A new record is then generated that associates the selected cluster $\widehat{y}$ (key) with two kinds of information: the element y and all of its contributions to the cluster-wise probabilities $p_i(\widehat{X}, \widehat{Y})$ — i.e. a record of the form $\langle(\mathbf{i}, \widehat{\mathbf{y}}), \{y\}, \{(\widehat{x}, p_i(\widehat{x}|y)) \mid p_i(\widehat{x}|y) > 0\}\rangle$.

Reduce. Reducers combine the above intermediate records, based on cluster IDs, by merging the associated (singleton) membership vectors and by aggregating the probabilities into the cluster-wise statistics $p_i(\widehat{X}, \widehat{Y})$. Since this computation is identical to that performed in the initialization of the clusters, we omit further details.

D. Updating the clustering of the central domain. From a conceptual viewpoint, this task is pretty similar to the clustering of an auxiliary domain, as long as concerns the flow of data and the results. Actually, the main difference lies in the fact that each element x being re-clustered is now associated with N probability vectors $p_i(Y|x)$ (i.e. the x-th row in matrix P_i) for $i = 1..N$.

Map. Still assuming that all the joint probabilities of any element $x \in X$ are kept together, each mapper can choose the new clusters $C'_{Y^i}(x)$ locally, by computing all the scores $\delta_i(x, \widehat{x})$, for each possible cluster $\widehat{x}$ (cf. Equation 2) – based on the clusterings $C_X, C'_{Y^1}, .., C'_{Y^N}$ and global statistics $p_i(\widehat{X}, \widehat{Y})$ computed in the previous map-reduce task. These score are indeed the only information needed to make the decision independently of the politics adopted for weighting the pairwise relationships (e.g.,AD−HOCC

or LC−HOCC). As for the updating of auxiliary domains' clusters, the selected cluster $\widehat{x}$ is treated as a new key, and is associated with both x and its non-zero contributions to clustered distributions $p_i(\widehat{X}, \widehat{Y})$, for $i = 1..N$. Eventually, for each any element x a record is produced of the form $\langle \widehat{\mathbf{x}}, \{x\}, \{(i, \widehat{y}, p_i(\widehat{y}|x)) \mid p_i(\widehat{y}|x) > 0\}\rangle$.

Reduce. Again, reducers combine the above intermediate records by merging their associated singleton membership vectors and by aggregating the probabilities into the cluster-wise statistics $p_i(\widehat{X}, \widehat{Y})$. This computation is identical to the one carried out in the initialization phase, and hence we do not detail it any more.

All of the map-reduce tasks described above are coordinated by a single driver task, which keeps on iterating the parallel computation phases C and D until the termination condition is satisfied.

4 Experiments

The experimental activity was conducted over both synthetic and real data. Tests on real-world data were aimed at assessing the accuracy of the co-clustering solutions found, while those on synthetic data were devoted to support scalability analyses, and to verify the actual benefits of parallelization. All running code was implemented in Java by using the Sun JDK version 1.6.0_17-b04.

In all tests, the number of required clusters over each domain was set equal to the number of clusters expected for that domain. Moreover, in order to reduce statistical bias, 20 runs were performed against every dataset, then reporting average measures.

4.1 Effectiveness Tests on Real Data

For testing the effectiveness of our implementation, we reproduced some of the experiments in [10], using data extracted from popular 20-*Newsgroup* corpus. This corpus gathers a number of news documents, from 20 different newsgroups. In particular, we built two different datasets, named $TM1$ and $TM2$, where a subset of documents was taken from certain newsgroups. The newsgroups themselves can be considered as categories for their associated documents, and can be grouped in classes based on their topics. In more detail, dataset $TM1$ consists of documents of five categories (i.e. newsgroups) organized in two classes concerning sport and politics, respectively. $TM2$ collects instead documents of five categories (i.e. newsgroups), organized in three groups, related to computers, autos/motorcycles and electronic devices, respectively.

Effectiveness was evaluated based on the correspondence between the original groups of documents and the ones discovered by co-clustering the documents together with the other kinds of objects they are correlated with (i.e., the words contained in them and the group/category they came from). To this purpose, we resorted two classical accuracy measures: *(i)* the *micro-averaged precision* P and *(ii)* the *normalized mutual information (NMI)*.

The co-clustering approach was tested with both the AD−HOCC LC−HOCC strategies, using this latter with different values of the weights β_c and $\beta_t = 1 - \beta_c$ associated with the document-by-terms and the document-by-terms matrices, respectively. Each

		P		NMI	
Methods	β_c	*TM1*	*TM2*	*TM1*	*TM2*
LC−HOCC	0.9	0.850	0.718	0.484	0.593
	0.7	0.838	0.649	0.444	0.476
	0.5	0.813	0.673	0.362	0.479
	0.3	0.832	0.657	0.422	0.471
	0.1	0.831	0.656	0.417	0.449
AD−HOCC	N.A.	0.887	0.749	0.606	0.602

Table 1. Results on real datasets.

run was performed on the 200 biggest documents, by considering 2000 terms filtered according to a common stop-words list and to their relevance, based on their mutual information over the whole corpus of documents. For any document belonging to h categories, its correlation with each of them was set to $1/h$ in the document-by-category matrix and the document-by-term matrix were initialized with $k\text{-}means$.

The results obtained, shown in Table 1, are perfectly in line with those of the sequential versions of the algorithms [10]. In particular, it is worth noticing that AD−HOCC always produces better results (in terms of both precision and NMI) than LC−HOCC, whatever value of β_c is used.

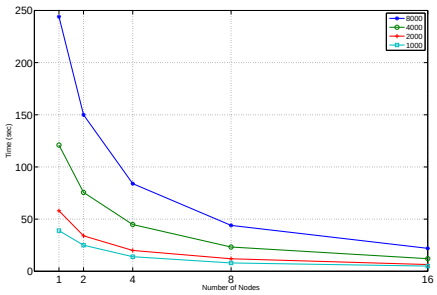
4.2 Scalability tests on synthesized data

Scalability tests were performed on a cluster of 16 nodes, with a locally attached 1TB SATA hard drive. Each node is a dual-core processor (Intel Xeon E5520 2,26GHz) with 2GB DDR3 1333MHz RAM and running Linux CentOS 5.3 with kernel 2.6.18.

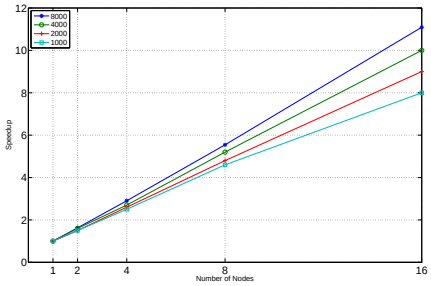
Synthetic datasets were produced with the Java generator described in [10], by varying the following parameters: *(i)* the number $N > 1$ of auxiliary domains, *(ii)* the sizes m, $n_1, ..., n_N$ of the domains D_X, $D_{Y^1}, ..., D_{Y^N}$, *(iii)* the number of required clusters along each dimension.

The improvement gained through parallelization can be appreciated in Figure 2. There the time spent for co-clustering several datasets is reported, when varying the size of the central domain (up to 2000 objects) and the number of processor nodes, while fixing the number of auxiliary domains to 16 (i.e. $N = 16$) and the size of each of them to 2500 (i.e. $n_i = 2500$ for $i = 1..N$).

In particular, Figure 2.(a) evidences the gain in the execution time that the parallelized version has w.r.t. the sequential one. The same behavior is shown in Figure 2.(b), in terms of relative speed-up, which is almost linear, even though with a slope lower than 1. This is mainly due to communication costs and to other kinds of overhead of the runtime Map-Reduce environment used in the tests. We finally notice that the speed-up slope gets better when higher volumes of data (namely, higher amounts of elements in the central domain) are given in input. In fact, increasing the number of elements that are to be clustered only impacts on parallelized tasks and not on the synchronized aggregation of their partial results.



(a)



(b)

Fig. 2. Computation time when varying the number of processing nodes and the size of central domain.

5 Conclusion

This work attempts to face the problem of efficiently co-clustering an arbitrary number of inter-related domains, which has received little attention in the community of parallel Data Mining — which mainly concentrated on the case of just one or two data types. Specifically, we illustrated a parallel implementation of two a state-of-the-art methods, based on a Map-Reduce infrastructure. Theoretical and empirical analyses confirmed both the scalability of the approach and its capability to achieve the same accuracy as the sequential version of the methods. As future work, we plan to extend the approach to arbitrary inter-relationships structure (going beyond the star-shaped one considered in this paper), yet providing a parallelized implementation based on Map-Reduce.

References

1. http://hadoop.apache.org/.
2. Arindam Banerjee, Sugato Basu, and Srujana Merugu. Multi-way clustering on relation graphs. In *Proceedings of the 7th SIAM International Conference on Data Mining (SDM 2007)*, pages 145–156. SIAM, 2007.
3. R. Bekkerman and M. Scholz. Data weaving: scaling up the state-of-the-art in data clustering. In *Proc. of the 17th ACM conference on Information and knowledge management*, pages 1083–1092, 2008.

4. Ron Bekkerman, Ran El-Yaniv, and Andrew McCallum. Multi-way distributional clustering via pairwise interactions. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 41–48, New York, NY, USA, 2005. ACM Press.
5. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.
6. I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proc. 9th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining (KDD03)*, pages 89–98, New York, NY, USA, 2003. ACM Press.
7. I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 245–260, 2000.
8. C.T. Chu et al. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.
9. T. George and S. Merugu . A scalable collaborative filtering framework based on co-clustering. In *Proc of the Fifth IEEE International Conference on Data Mining*, pages 625–628, 2005.
10. Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Co-clustering multiple heterogeneous domains: Linear combinations and agreements. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints).
11. Bo Long, Zhongfei (Mark) Zhang, Xiaoyun Wú, and Philip S. Yu. Spectral clustering for multi-type relational data. In *Proceedings of the 23rd International Conference on Machine learning (ICML 2006)*, pages 585–592, New York, NY, USA, 2006. ACM Press.
12. S. Papadimitriou and J. Sun. Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *Proc. of the 2008 Eighth IEEE International Conference on Data Mining*, pages 512–521, 2008.
13. J. Zhou and A. Khokhar. Parrescue: Scalable parallel algorithm and implementation for biclustering over large distributed datasets. In *Proc. of the 26th IEEE International Conference on Distributed Computing Systems*, pages 21–, 2006.