An Architecture to Efficiently Learn Co-Similarities from Multi-View Datasets

Gilles Bisson and Clément Grimal

Université Joseph Fourier / Grenoble 1 / CNRS,
Laboratoire LIG - Bâtiment CE4, 38610 Gières FRANCE
{gilles.bisson,clement.grimal}@imag.fr

Abstract. In this paper, we introduce the MVSim architecture which is able to cluster multi-view datasets (i.e. datasets containing several objects linked together by different relations), by using several instances of a co-similarity algorithm. We show that this framework provides better results than existing approaches, while reducing both time and space complexities thanks to an efficient parallelization of the computations. This approach allows to split large datasets into a set of smaller ones.

Keywords: Multiview Learning, Similarity Learning, Co-clustering.

1 Introduction

Co-clustering methods allow to efficiently capture high-order similarities between objects described by rows and columns of a data matrix. However, in complex domains as *social network analysis*, many objects and relations exist, such as: users/users, users/documents, documents/tags, ... all of them providing a different *view* on the dataset that can be expressed as a collection of matrices. By separately processing these matrices, we get a huge loss of information.

Therefore, *multi-view clustering* task is an interesting challenge *wrt* classical clustering. Since the seminal work of [2], introducing semi-supervised learning, many extensions to the clustering methods have been proposed to deal with such multi-view data. For example, [5] and [1] respectively describe an extension of k-means (MVKM) and of EM algorithms; the framework of spectral clustering has also been investigated, for instance in [7] the similarities computed in one view are used to constrain the similarities computed in the other views through the eigenvectors of the Laplacian matrix. It is worth noting that multi-view clustering can also be tackled by *consensus clustering* methods which aim at combining the results of multiple clusterings [8]. Similarly, some works aim at combining multiple similarity matrices to perform a given learning task [9], [3], the idea being to build clusters from multiple similarity matrices computed along different views. The present work is an extension of an existing algorithm, named χ-Sim [6], which obtained good results on the co-clustering task.

The rest of the paper is structured as follows. In Sect. 2, after introducing some notations, we provide a rapid insight about the χ-Sim method and then,

we present and analyze the MVSim architecture allowing to adapt the previous algorithm to the multi-view context. In Sect. 3, we explain how it is possible to use this architecture to efficiently compute co-similarities on large databases by splitting a data matrix into smaller ones. Finally, in both sections (2 and 3), we provide some experimental results in order to evaluate our proposals.

2 Dealing with multi-view databases

2.1 Notations

Here, we use the classical notations: matrices (in capital letters) and vectors (in small letters) are in bold; variables are in italic.

Type of objects: let N be the number of types of objects in the dataset. $\forall i \in 1..N$, T_i is the type of object i (i.e. users, documents, words, etc.). For the sake of simplicity, we assume that each T_i has the same set of n_i instances across the collection of matrices.

Relation matrices: let M be the number of matrices in the dataset. Then $\mathbf{R}_{i,j}$ is the relation matrix describing connections between instances of T_i and T_j, of size $n_i \times n_j$. Each element $(\mathbf{R}_{ij})_{ab}$ expresses the link 'intensity' between the a^{th} instance of T_i and the b^{th} instance of T_j. For example, in a [documents/terms] matrix it can be expressed as the frequency of the b^{th} term in the a^{th} document.

Similarity matrices: we can thus consider N similarity matrices $\mathbf{S}_1 \dots \mathbf{S}_N$. Then $\mathbf{S}_i$ (of size $n_i \times n_i$) is the square and symmetrical matrix that contains the similarities between all the pairs of instances of T_i. The values of the similarity measures must be in $[0, 1]$, the value 1 expressing a full similarity.

2.2 Algorithm χ-Sim

The basic component of our approach is the χ-Sim co-similarity measure [6]. The main idea behind χ-Sim is to make use of the duality between object (e.g. documents and words): each one being a descriptor of the other. This is achieved by simultaneously calculating similarities between documents on the basis of the similarities between their words, and similarities between words on the basis of the similarities between the documents in which they appear. Once the similarity matrices have been generated they can be used by any clustering algorithm (for example k-means) to organize documents and/or words into clusters. However, due to the interleaved way these similarities have been computed, the resulting clusters are similar to those obtained with a genuine co-clustering algorithm.

We selected this approach for two reasons. First, it simultaneously builds similarity matrices $\mathbf{S}_i$ and $\mathbf{S}_j$, rather than set of clusters, between rows and columns of a data matrix $\mathbf{R}_{i,j}$. This is useful in the multi-view context to combine easily the set of similarity matrices computed from the different matrices of the dataset. Second, in this algorithm, the similarity matrices of each type of objects T_i can be initialized, allowing us to inject some *a priori* knowledge about the data. In this way, it becomes possible to iteratively transfer the similarities computed from one view to the others.

More formally, the inputs of χ-Sim are: a data matrix $\mathbf{R}_{i,j}$ describing T_i and T_j relationship, an initialization of the two matrices $\mathbf{S}_i$ and $\mathbf{S}_j$ (i.e. set by default to the identity matrix $\mathbf{I}$), and the outputs are the two modified similarity matrices, denoted $\mathbf{S}_i^{(i,j)}$ and $\mathbf{S}_j^{(i,j)}$ computed by χ-Sim to capture high order co-occurrences between rows and columns of $\mathbf{R}_{i,j}$.

2.3 The MVSim Architecture

In this paper, we want to compute simultaneously the co-similarity matrix $\mathbf{S}_i$ for each of N different kinds of objects T_i described by the M relation matrices $\mathbf{R}_{i,j}$ of the dataset. The ground idea is to create a learning network isomorphic to these dataset structure (see Fig. 1 and Fig. 2 to have an example of network). At first, an instance of χ-Sim, denoted χ-Sim$_{i,j}$, is associated to each matrix $\mathbf{R}_{i,j}$. It computes the similarity matrices $\mathbf{S}_i^{(i,j)}$ and $\mathbf{S}_j^{(i,j)}$. However, for a given type of object T_i, as each instance χ-Sim$_{i,.}$ produces its own similarity matrix, we thus get a set of output similarity matrices $\{\mathbf{S}_i^{(i,1)}, \mathbf{S}_i^{(i,2)}, ...\}$ the cardinal of which being equal to the number of relation matrices related to T_i. Therefore, we need to introduce an *aggregation function*, denoted Σ_i, to compute a consensus similarity matrix merging all of the $\{\mathbf{S}_i^{(i,1)}, \mathbf{S}_i^{(i,2)}, ...\}$ with the current matrix $\mathbf{S}_i$. In turn, these resulting consensus matrices are connected to the inputs of all the χ-Sim$_{i,.}$ instances, thus *creating feedback loops* allowing the system to spread the knowledge provided by each $\mathbf{R}_{i,j}$ within the network.

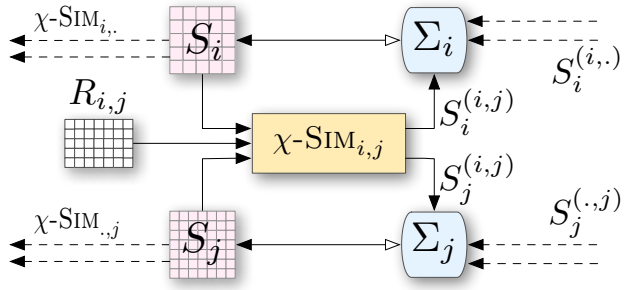

Fig. 1. Generic component of the architecture to deal with one $\mathbf{R}_{i,j}$ matrix.

The system runs iteratively: χ-Sim$_{i,j}$ instances are fired in parallel (Alg. 1), then the similarity matrices $\mathbf{S}_i$ are updated through Σ_i aggregation functions. Without detailing (see [6]), the meaning of an iteration t is the same as in χ-Sim: it takes into account the $order - t$ paths of the bipartite graph expressed by each matrix $\mathbf{R}_{i,j}$. Of course, functions Σ_i must be defined in order to ensure that $\mathbf{S}_i$ are converging along iterations and to take into account that confidence of the information provided by each iteration decrease according to length of the paths. More formally, let $\lambda \in [0, 1[$ be a damping parameter, let F a merging function (here achieved by computing the element-wise average of all matrices)

combining the matrices $\{\mathbf{S}_i^{(i,1)}, \mathbf{S}_i^{(i,2)}, ...\}$ and let $\mathbf{S}_i^{[t-1]}$ be the previously computed similarity matrix of instances of T_i. The formula used is as follows:

$$\Sigma_i = (1 - \lambda^t)\, \mathbf{S}_i^{[t-1]} + \lambda^t\, F\left(\mathbf{S}_i^{(i,1)}, \mathbf{S}_i^{(i,2)}, \dots\right) \tag{1}$$

As the F function is bounded and the damping factor λ^t is exponentially decreasing with t, this formula ensures the convergence of the sequence composed of the successive similarity matrices computed by the Σ_i functions. Experimentally, with $\lambda = 0.5$ convergence is obtained after about 6 iterations.

Algorithm 1 The MVSim algorithm

Require: A collection of relation matrices $\{\mathbf{R}_{i,j}\}$
  Let $\{\mathbf{S}_i\}$ the similarity matrices with $\mathbf{S}_i \leftarrow \mathbf{I}$
  for $t = 1 \rightarrow Convergence$ **do**
    Execute every χ-Sim$_{i,j}$
    Update every $\mathbf{S}_i$ with Σ_i using Eq. (1).
  end for

The complexity of MVSim architecture is obviously related to the one of the χ-Sim algorithm. Let us consider a matrix $\mathbf{R}_{i,j}$ of size n by m with $m > n$, as this algorithm consists in multiplying three matrices (see [6]), the complexity to compute a similarity matrix of size m^2 (columns) equals $\mathcal{O}(nm^2)$. In the MVSim framework, as each instance of χ-Sim$_{i,j}$ can run on an independent core, the method can easily be parallelized, thus keeping the global complexity unchanged (considering the number of iterations as a constant factor). Finally, the complexity of the Σ_i functions can be ignored since it equals $\mathcal{O}(m^2)$. As we will see in Sect. 3, MVSim can also be useful to turn a large problem into a collection of simpler ones, thus reducing further the overall complexity.

2.4 Experiments

We selected datasets with labeled clusters and then we evaluated the correlation between the clusters learned with MVSim and those already known using the classical Micro-Averaged Precision (MAP) [4]. We used eight datasets[1]. The first dataset is extracted from the *IMDb* website. It contains three types of objects: movies, actors and keywords and two relation matrices: the [movies/actors] matrix and the [movies/keywords] matrix. The six next databases concern "Web data" and are all constructed following the same structure with two types of objects (documents and words) and four relation matrices. More precisely, we used the Cora and CiteSeer dataset [5] and four datasets describing the pages of universities (WebKB), classified in five classes. Finally, we built a multi-view dataset from the Reuters RCV1/RCV2 collection following the methodology of [7]: we used the [documents/words] matrices in english and their traductions in french, german, italian and spanish, to get a total of 5 views.

[1] Dataset repository and details: `http://membres-lig.imag.fr/grimal/data.html`

With these eight benchmarks, we compared MVSim with: **Cosine**, **LSA**, **CTK** [10] and **χ-Sim** [6] that are five classical similarity or co-similarity measures; **ITCC** [4] a well-known co-clustering system; **MVSC** [7] a multi-view algorithm. Finally, we ran two basic versions of MVSim without iteration (no feedback loop nor damping factor), to verify that our results are significantly better than those obtain by simply averaging the similarity matrices computed from each $\mathbf{R}_{i,j}$; we tested two similarity measures : cosine (Merge Cosine) and χ-Sim (Merge χ-Sim). For the similarity based systems: Cosine, LSA, SNOS, CTK and χ-Sim, the final clusters were generated by an *Agglomerative Hierarchical Clustering* method using Ward's linkage. Then, we cut the resulting tree at the level corresponding to the number of expected classes.

Table 1. Results of the experiments expressed with the Micro-averaged precision.

Datasets	Best monoview		Merge Cosine	Merge χ-Sim	MVSC	MVSim
IMDb	0,332	CTK	0,191	0,233	0,296	**0,347**
Cora	0,502	χ-Sim	0,394	0,393	0,528	**0,697**
CiteSeer	0,608	χ-Sim	0,405	0,560	0,578	**0,635**
Cornell	0,631	χ-Sim	0,364	0,569	0,519	**0,708**
Texas	**0,722**	χ-Sim	0,497	0,642	0,591	0,647
Washington	0,652	LSA	0,470	0,635	0,605	**0,709**
Wisconsin	0,675	χ-Sim	0,600	0,536	0,551	**0,706**
ReutersEN	**0,601**	LSA	0,35	0,420	0,510	0,509

Although we tested single-view algorithms on all the views of the eight datasets, we just report in Table 1 the result obtained by the best method along with its name. MVSim obtains the best results in all the datasets but two. With Reuters, MVSC is slightly better but LSA is a clear winner with a single view (english version) and with Texas (best: χ-Sim and LSA) MVSim is ranked at the third position. We are still investing the reasons why our algorithm fails on this last dataset since it is very close to Cornell, Washington and Wisconsin and the two data matrices *content* and *in/out* do not seem to contain contradictory information. It is worth noticing that none of the two consensus approaches (Merge Cosine and Merge χ-Sim) obtain good results, emphasizing the fundamental role played by the feedback loop of our architecture.

3 Parallelization and splitting approaches

Until now, we considered multi-views clustering as a way to combine knowledge coming from different sources of data. However, at the same time, the MVSim architecture can also be used to reduce the algorithmic complexity of a problem by splitting a data matrix $\mathbf{R}$ into a collection of smaller ones, each submatrix becoming a component of our network and processed as a separate view. This can be done either on one dimension of $\mathbf{R}$ (Sect. 3.1) or both dimensions (Sect. 3.2).

3.1 One dimensional splitting

Let us consider a dataset with one [documents/words] matrix of size n by m in which we just want to cluster the documents. If the number of words is huge with respect to the number of documents, we can divide the problem into a collection of h submatrices of size n by m/h (Fig. 2). Thus, by using a distributed version of MVSim on h cores, we will gain both in time and space complexity: indeed, the time complexity decreases from $\mathcal{O}(nm^2+n^2m)$ to $\mathcal{O}(1/h^2(nm^2)+1/h(n^2m))$ leading to an overall gain of $1/h^2$ when $n < m$. In the same way, the memory needed to store the similarity matrices between words will decrease by a $1/h$ factor (but not $1/h^2$ since we have now h similarity matrices to compute).

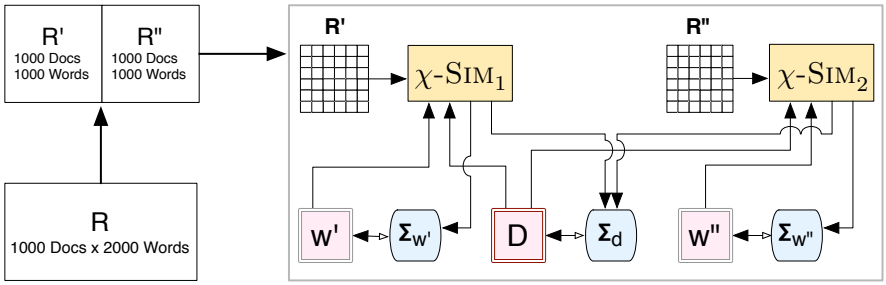

Fig. 2. Example of a [documents/words] matrix $\mathbf{R}$ splitted vertically into two submatrices $\mathbf{R}'$ and $\mathbf{R}$" and the corresponding MVSim network. Here, the goal is to learn the co-similarity matrix $\mathbf{D}$ between documents, the two other matrices $\mathbf{W}'$ and $\mathbf{W}$" being only used during the learning process.

Of course, when using this splitting approach, we lost some information. In the example of Fig. 2, we don't compute the co-similarities between all pairs of words but only between the words occurring in $\mathbf{R}'$ or those occurring in $\mathbf{R}$"; there are no "inter-matrices" comparisons. However, our assumption is that, thanks to the feedback loops of the MVSim network and to the presence of the common co-similarity matrix $\mathbf{D}$, we will be able to alleviate this problem.

3.2 Two dimensional splitting

Space complexity of a distance (or similarity) matrix $\mathcal{O}(N^2)$ is a strong limit to the number of instance that a learning algorithm can process. For instance, a similarity matrix between one millions of instances needs *terabytes* of storage. Here, we propose to use the MVSim architecture to deal with this problem.

Let us consider one [documents/Words] square matrix $\mathbf{R}$ of size n by n. If we assume we have an access to a cluster of computers having h^2 nodes (or cores) the idea is to split $\mathbf{R}$ into h^2 submatrices of size n/h and then to use a distributed MVSim architecture to learn the similarity matrices with these submatrices (Fig. 3). In this way, the time complexity decreases from $\mathcal{O}(n^3)$ for the full matrix to $\mathcal{O}(n/h)^3$ thus leading to an strong overall gain of $1/h^3$. In this

approach, each Σ_i functions has to merge h partial similarity matrix (line or column of the network), however this cost remains negligible as long as $n > h^2$.

Concerning the space complexity, during the learning step, as we need to compute a similarity matrix for all the submatrices, the overall memory consumption is the same as with a classical approach $\mathcal{O}(n^2)$ but it is shared on the h^2 nodes. However, the two output matrices only use $\mathcal{O}(n^2/h)$ of memory, thus leading to an gain of $1/h$. Indeed, the learned similarity matrices correspond to the "diagonal" of the general co-similarity matrices [documents/documents] and [words/words]. As we evoked in section 3.1, documents (resp. words) of one submatrix are only compared with documents of the same submatrix.

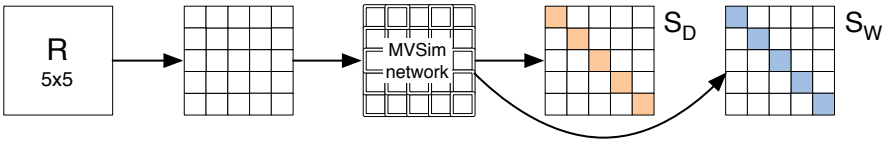

Fig. 3. Assuming with have a cluster of h^2 cores. We split a [documents/words] matrix $\mathbf{R}$ into a collection of h^2 submatrices of size n/h and we create the corresponding MVSim network to deal with them in parallel. The output is a collection of co-similarity matrices (colored elements) that are a subset of the two general co-similarity matrices between documents and words: $\mathbf{S_D}$ and $\mathbf{S_W}$.

Of course, this is a problem since with the learned matrices we are not able to know the similarity between each pair of documents or each pair of words. Fortunately, we can use the following trick in order to evaluate the missing co-similarities. Let us consider two documents $\mathbf{d_i} = (\mathbf{R})_{i:}$ and $\mathbf{d_j} = (\mathbf{R})_{j:}$ of the data matrice $\mathbf{R}$ that was not in the same submatrix when we learned the co-similarity matrices. We compute their co-similarity in the following way :

$$CoSim(\mathbf{d_i}, \mathbf{d_j}) = \mathbf{d_i} \times \mathbf{S_W} \times \mathbf{d_j}^T \tag{2}$$

On the one hand, this value is an approximation of the value we will get by using directly the χ-Sim method (especially if h is large), but on the other hand this splitting approach allows to compute co-similarity values on larger datasets.

3.3 Experiments

To evaluate the divisive approaches introduced in the two previous sections, we used the classical NG20 collection consisting of approximately 20,000 newsgroup articles collected from 20 different Usenet groups[2]. Here, our goal is to cluster the articles (documents) and to explore the behavior of MVSim when varying the number of "splits" (i.e. of submatrices). From this collection, we selected the 10 newsgroups[3] having the largest number of documents as being our target

[2] `http://people.csail.mit.edu/jrennie/20Newsgroups/`

[3] Namely: comp.graphics, misc.forsale, rec.autos, rec.sport.baseball, rec.sport.hockey, sci.crypt, sci.med, sci.space, soc.religion.christian, talk.politics.mideast.

clusters. Next, we randomly built 10 folds of four subsets containing a different number of documents from 400 documents (40 per newsgroup), up to 6400 documents (320 per newsgroup) with the intermediate values 800, 1600 and 3200. Secondly, we selected a subset of 4000 words from the whole collection by using the k-medoids algorithm in order to get the "best" (most representative) words.

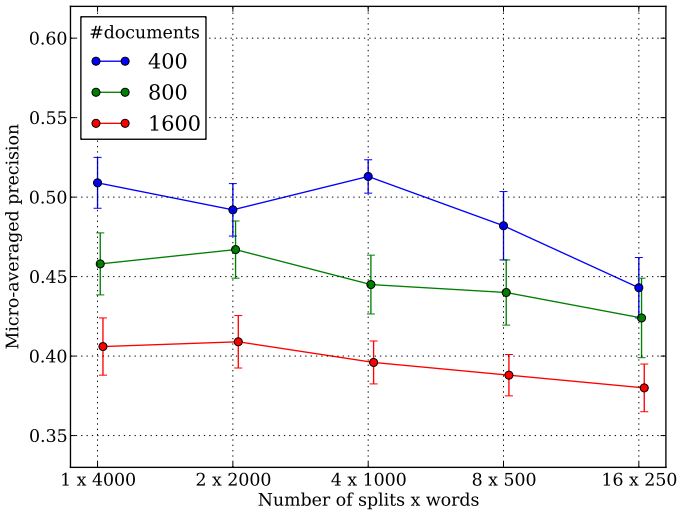

Fig. 4. Mean and standard deviation of the micro-averaged precision (over 10 folds) for various number of splits using the *one dimensional splitting*.

To evaluate one dimensional splitting, we tested the MVSim architecture with 1 split (full matrix) containing 4,000 words, then 2 random splits of 2,000 words, etc. until 16 random splits of 250 words. For each run, the number of χ-Sim$_{i,j}$ instances in the MVSim network equals the number of splits.

Fig. 4 shows the mean micro-averaged precision over 10 folds, obtained with the tested conditions. Obviously, the quality of the clustering tends to decrease when the number of splits increases, but if we compare, for instance, the micro-averaged precision obtained for 8 splits of 500 words with the best value (obtained with one matrix of 4000 words), the precision is only 2-3% lower (on average) for these three experiments. Nevertheless, to get this result, the computation time of the *similarity matrices between words* was divided by an impressive 64 factor[4] ($1/splits^2$) and the memory footprint is 8 ($1/splits$) times smaller. The observed loss of performance is due to the fact that by splitting the set of features (words), one prevent the system to compute the similarities between features across the different splits; furthermore, the relative performance drop we observe with 16 splits is a direct consequence of the fact that the number of words in each matrix becomes *too small* with respect to the number of documents.

[4] Of course, this is a theoretical result, on a multicore computer the speed gain could be smaller due to the limit of the bus bandwidth between the cores and the memory.

To conclude, there is a clear trade-off between the quality of the clustering and both running time and memory usage. But when computation time and memory needs to be reduced, the MVSim architecture provides a very efficient solution to speed-up computations with a minimal loss in clustering accuracy.

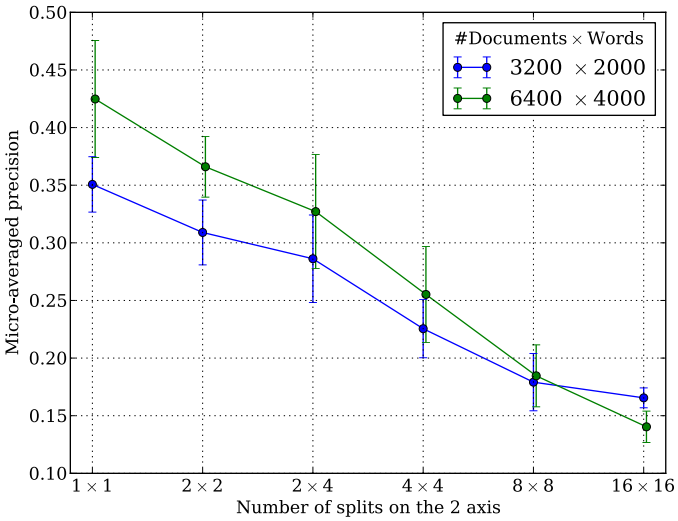

Fig. 5. Mean and standard deviation of the micro-averaged precision (over 10 folds) for various number of splits using the *two dimensional splitting*.

To evaluate two dimensional splitting, we tested the MVSim architecture with two matrices [documents/words] of size $[3,200 \times 2,000]$ and $[6,400 \times 4,000]$. In both cases we tested several kinds of random splits: initial matrix, 2×2, 2×4, 4×4, 8×8 and 16×16 splits. For each run, the number of χ-Sim$_{i,j}$ instances in the MVSim network equals the total number of submatrices. Fig. 5 shows the mean micro-averaged precision over 10 folds, obtained with the different tested conditions. Here, the quality of the clustering decreases more rapidly than in the previous experiment when the number of splits increases. However, for the $[6,400 \times 4,000]$ dataset and 2×2 splits, we observe that the result if slightly better than for the $[3,200 \times 2,000]$ dataset, although the two experiments are using the same amount of time, thanks to the parallelization. Nevertheless, it is clear that our second divisive strategy needs to be improved.

4 Conclusion

In this paper, we proposed the MVSim architecture to deal with the problem of learning co-similarities from a collection of matrices describing interrelated types of objects. This architecture provides some interesting properties both in terms of convergence and scalability and it allows a straightforward and efficient

parallelization. The experiments demonstrate that this architecture outperforms both single-view and multi-view approaches on several benchmarks.

For future works, many directions seem compelling to explore such as generalizing our architecture to work with *clustering ensembles* by considering, in the network, a data-flow of clusters rather than similarities. In the two divisive approaches more sophisticated splitting strategies will also be investigated such as using a fast clustering method (e.g k-means) in order to create more coherent submatrices. Another interesting perspective would be to adapt MVSim to supervised learning by modifying the aggregation function.

Acknowledgments. This work is partially supported by the French ANR project FRAGRANCES under grant 2008-CORD 00801.

References

1. Bickel, S., Scheffer, T.: Multi-view clustering. In: 4th IEEE International Conference on Data Mining. pp. 19–26. Brighton, UK (2004)
2. Blum, A., Mitchell, T.: Combining labeled and unlabeled data with co-training. In: 11th annual conference on Computational Learning Theory. pp. 92–100. ACM (1998)
3. de Carvalho, F., Lechevallier, Y., de Melo, F.M.: Partitioning hard clustering algorithms based on multiple dissimilarity matrices. Pattern Recognition 45, 447 – 464 (2012)
4. Dhillon, I.S., Mallela, S., Modha, D.S.: Information-theoretic co-clustering. In: 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 89–98. Washington, DC, USA (2003)
5. Drost, I., Bickel, S., Scheffer, T.: Discovering communities in linked data by multi-view clustering. In: 29th Annual Conference of the German Classification Society, Studies in Classification, Data Analysis, and Knowledge Organization. pp. 342–349. Magdeburg, Germany. Springer (2005)
6. Hussain, F., Grimal, C., Bisson, G.: An improved co-similarity measure for document clustering. In: 9th International Conference on Machine Learning and Applications. pp. 190–197. Washington DC, USA (2010)
7. Kumar, A., Daume III, H.: A co-training approach for multi-view spectral clustering. In: 28th International Conference on Machine Learning. pp. 393–400. Bellevue, Washington, USA (2011)
8. Li, T., Ding, C.: Weighted consensus clustering. In: 8th SIAM International Conference on Data Mining. pp. 798–809. Atlanta, USA (2008)
9. Tang, W., Lu, Z., Dhillon, I.S.: Clustering with multiple graphs. In: 9th IEEE International Conference on Data Mining. pp. 1016–1021. Miami, Florida, USA (2009)
10. Yen, L., Fouss, F., Decaestecker, C., Francq, P., Saerens, M.: Graph nodes clustering with the sigmoid commute-time kernel: A comparative study. Data & Knowledge Engineering 68(3), 338–361 (2009)