Accelerating Relational Clustering Algorithms With Sparse Prototype Representation

Fabrice Rossi[(1)], Alexander Hasenfuß[(2)], and Barbara Hammer[(2)]
[(1)] Projet AxIS, INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, B.P. 105, 78153 Le Chesnay cedex, France
Fabrice.Rossi@inria.fr
[(2)] Clausthal University of Technology - Institute of Computer Science
Julius Albert Strasse 4, 38678 Clausthal-Zellerfeld - Germany
{hasenfuss,hammer}@in.tu-clausthal.de,

Keywords: relational data, pairwise data, dissimilarity data, software implementation

Abstract— In some application contexts, data are better described by a matrix of pairwise dissimilarities rather than by a vector representation. Clustering and topographic mapping algorithms have been adapted to this type of data, either via the generalized Median principle, or more recently with the so called relational approach, in which prototypes are represented by virtual linear combinations of the original observations. One drawback of those methods is their complexity, which scales as the square of the number of observations, mainly because they use dense prototype representations: each prototype is obtained as a virtual combination of all the elements of its cluster (at least). We propose in this paper to use a sparse representation of the prototypes to obtain relational algorithms with sub-quadratic complexity.

1 Introduction

In some application domains, data cannot be described by vectors, while in others, the standard Euclidean metric doesn't provide a meaningful way of comparing objects. Online handwriting recognition, for instance, implies to classify drawing of characters that vary in length and in execution speed: the non constant length prevents a direct vector representation and even in the particular case of identical dimensions, the non constant drawing speed imposes to use distance based on variants of some type of dynamic time warping to achieve good recognition rate [2]. Other examples can be found in, e.g., biology where proteins are considered as sequences of amino acids (see e.g., [13]).

A generic way to address this type of data is to use a measure of resemblance between the considered objects (a similarity or a dissimilarity) that can be expertly designed to handle complex non vector representation and/or to focus on some specific aspects of the data that cannot be captured by the Euclidean metric. When data are considered only via the outcome of all the pairwise comparison between the observations, they are called indifferently *pairwise data*, *(dis)similarity data* or *relational data*.

Numerous data analysis methods have been designed for relational data, especially for the specific problem of clustering. Hierarchical clustering methods for instance are naturally capable of handling dissimilarity data. Prototype based clustering has also been an early target for extension from the vector case to relational data, in particular with the Partitioning Around Medoids algorithm [10], which extends K-means like algorithm to relational data using the idea of the generalized median: as the center of mass of each cluster cannot be computed for relational data, it is replaced by its best approximation among the original data, by minimizing the sum of the dissimilarities between the candidate and all the elements of the cluster. More recently, this principle has been used to extend the Self Organizing Map [11, 1, 12] and Neural Gas [4] to relational data (see also [5, 6] for a quite different approach based on mean field annealing).

One drawback of the median based method is the quantization effect induced by the restriction on the prototype values. As they must be chosen in the original dataset, any sparsity in this set or any sampling problem will be negatively reflected in the classification result. In the particular case of the Median SOM this induces frequently some strong map folding as several distinct neurons share the same prototype value. While this particular problem can be avoided by enforcing uniqueness among prototype values [14], the general problem of the low quality of some prototypes cannot be addressed in the median framework.

A solution consists in using implicit "linear combination" of the original data. When the dissimilarity is the squared Euclidean distance, one can obtain the distance between a data point and any linear combination of the original data points using solely the dissimilarity matrix. While the formula is valid only for the squared Euclidean distance, it can be applied to other dissimilarities, as proposed in [9, 8] for K-means and some fuzzy variations of this algorithm. We show in [7] that it can also be used for the Self Organizing Map and for Neural Gas, both in there batch versions.

Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM 2007)
Published by the Neuroinformatics Group, Bielefeld University, Germany, ISBN 978-3-00-022473-7
All contributions to WSOM 2007 are available online at: http://biecoll.ub.uni-bielefeld.de

One limitation shared by median algorithms and relational ones is there cost. Careful implementations of median algorithms lead to minimal costs per iteration (epoch) of order $O(N^2)$ (for N observations), as shown in [3] for the Median SOM. We show in the present paper that this is also the case for the relational K-means. As the size of the dissimilarity matrix is $O(N^2)$ one might consider this cost per iteration to be acceptable. However, clustering and mapping algorithms are by essence exploratory methods: while the dissimilarity matrix is computed once and for all, exploratory methods have some user driven parameters such as the number of clusters. It is natural for the user to test several values of this number (and of other parameters if needed). Moreover, while batch methods tend to converge very quickly to a local minimum, its quality strongly depends on the initial random configuration. It is therefore very important to compare outcomes of several random starts. All in one, a per iteration cost of $O(N^2)$ can lead in practice to quite important running times that are incompatible with exploratory analysis of large datasets.

This motivates the present work: we propose a strategy to enforce sparse prototypes in the sense that each prototype is obtained as an implicit linear combination of a small number of original data points (called its support points). This transforms the cost per iteration from $O(N^2)$ to $O(NPK+KP^3)$, where P is the number of data points per prototype and K is the number of clusters. While this approach is used here for the relational K-means only, it could be applied in the same way to topographic relational mappings [7].

The rest of the paper is organized as follows. Section 2 recalls the relational K-means algorithm and shows how to obtain the $O(N^2)$ per iteration cost. Then Section 3 presents the proposed sparse strategy, which is experimentally studied in section 4.

2 Relational K-means

2.1 Relational data

We assume given N observations, $(\mathbf{x}_i)_{1\leq i\leq N}$ together with a dissimilarity matrix, $D=(d_{ij})_{1\leq i,j\leq N}$ that fulfills the standard hypotheses for dissimilarities: D is symmetric, has only positive values and a zero diagonal.

Relational clustering methods are based on the following remark. Let us assume that the observations belong to $\mathbb{R}^n$ and that d is the squared Euclidean distance. Then consider a normalized linear combination of the observations, i.e., $y=\sum_{i=1}^N \alpha_i\mathbf{x}_i$ with $\sum_{i=1}^N \alpha_i=1$. Then we have:

$$\|y-\mathbf{x}_i\|^2=(D\boldsymbol{\alpha})_i-\frac{1}{2}\boldsymbol{\alpha}^TD\boldsymbol{\alpha}. \tag{1}$$

For an arbitrary dissimilarity on N observations from an arbitrary input space, we introduce implicit "linear combination" as normalized vectors from $\mathbb{R}^N$ and we define an extended dissimilarity:

$$d(\boldsymbol{\alpha},\mathbf{x}_i)=(D\boldsymbol{\alpha})_i-\frac{1}{2}\boldsymbol{\alpha}^TD\boldsymbol{\alpha}, \tag{2}$$

for $\boldsymbol{\alpha}\in\mathbb{R}^N$ and $\sum_{i=1}^N\alpha_i=1$.

The relational K-means uses linear combination prototypes (denoted $\boldsymbol{\alpha}^{(1)},\ldots,\boldsymbol{\alpha}^{(K)}$ for K clusters) and optimizes the following within class variance:

$$\sum_{k=1}^K\sum_{\mathbf{x}_i\in C_k}d(\boldsymbol{\alpha}^{(k)},\mathbf{x}_i), \tag{3}$$

where C_k is the k-th cluster. As in the standard K-means, this is done by alternating between an assignment phase in which each observation is assigned to the cluster of its closest prototype (in term of the extended dissimilarity) and a representation phase in which prototypes are updated by minimizing the within class variance while holding the partition constant. For cluster k, the representation phase consists in solving

$$(R_k)\ \min_{\boldsymbol{\alpha}\in\mathbb{R}^N}\sum_{\mathbf{x}_i\in C_k}d(\boldsymbol{\alpha},\mathbf{x}_i), \quad \text{with}\ \sum_{i=1}^N\alpha_i=1. \tag{4}$$

Some simple calculations leads to the obvious solution $\boldsymbol{\alpha}^{(k)}=\frac{1}{|C_k|}(\delta_{k,1},\ldots,\delta_{k,N})$, where $\delta_{k,i}=1$ if and only if $\mathbf{x}_i\in C_k$. The prototype can therefore by considered as an implicit center of mass for its cluster.

2.2 Naive algorithm

The standard formulation of the relational K-means is then the following one:

```
(1) initialize randomly the clusters
(2) compute the $\boldsymbol{\alpha}^{(k)}=\frac{1}{|C_k|}(\delta_{k,1},\ldots,\delta_{k,N})$
(3) for all $i\in\{1,\ldots,N\}$
    (a) for all $k\in\{1,\ldots,K\}$, compute $d(\boldsymbol{\alpha}^{(k)},\mathbf{x}_i)=\left(D\boldsymbol{\alpha}^{(k)}\right)_i-\frac{1}{2}{\boldsymbol{\alpha}^{(k)}}^TD\boldsymbol{\alpha}^{(k)}$
    (b) assign $\mathbf{x}_i$ to $C_k$ for which $d(\boldsymbol{\alpha}^{(k)},\mathbf{x}_i)$ is minimal
(4) update the $\boldsymbol{\alpha}^{(k)}$
(5) return to 3 if the algorithm has not converged
```

A naive implementation of this algorithm can lead to rather large running times, especially if the natural sparseness of the $\boldsymbol{\alpha}^{(k)}$ is not used. For an arbitrary vector $\boldsymbol{\alpha}$ (with $\sum_i\alpha_i=1$), computing $(D\boldsymbol{\alpha})_i$ is a $O(N)$ operation, while $\boldsymbol{\alpha}^TD\boldsymbol{\alpha}$ is a $O(N^2)$ operation. During the assignment phase of the algorithm, $\frac{1}{2}{\boldsymbol{\alpha}^{(k)}}^TD\boldsymbol{\alpha}^{(k)}$ has to be calculated for each cluster leading to a $O(KN^2)$ total cost. Moreover, $\left(D\boldsymbol{\alpha}^{(k)}\right)_i$ must be computed for each pair (k,i), leading again to a $O(KN^2)$ total cost. Other operations have a negligible relative cost: this naive implementation of the relational K-means has therefore a complexity of $O(KN^2)$.

2.3 Sparse Prototypes

Fortunately, the $\boldsymbol{\alpha}^{(k)}$ are not dense vectors: there is only $|C_k|$ nonzero terms in $\boldsymbol{\alpha}^{(k)}$. The total number of nonzero coefficients is therefore N. This can be used to reduce the complexity to $O(N^2)$.

The first aspect consists in using only nonzero terms in the calculation of $\left(D\boldsymbol{\alpha}^{(k)}\right)_i = \frac{1}{|C_k|}\sum_{j\in C_k} d_{ij}$, leading to a cost of $O(|C_k|)$ for one pair (k,i) and therefore to a total cost of $O(N^2)$, as $\sum_{k=1}^{K}|C_k| = N$. Applying this rule to the calculation of ${\boldsymbol{\alpha}^{(k)}}^T D\boldsymbol{\alpha}^{(k)}$ would lead to a cost of $O(|C_k|^2)$. However, assignment is based on the calculation of $\left(D\boldsymbol{\alpha}^{(k)}\right)_i$ for all pairs (k,i), including those for which $i \in C_k$. As ${\boldsymbol{\alpha}^{(k)}}^T D\boldsymbol{\alpha}^{(k)} = \frac{1}{|C_k|}\sum_{i\in C_k}\left(D\boldsymbol{\alpha}^{(k)}\right)_i$, the additional cost to obtain this value based on the $\left(D\boldsymbol{\alpha}^{(k)}\right)_i$ is only $O(|C_k|)$, i.e., $O(N)$ for all clusters, rather than $O(\sum_{k=1}^{K}|C_k|^2)$.

We obtained the following $O(N^2)$ algorithm:

(1) initialize randomly the clusters

(2) compute and store $\left(D\boldsymbol{\alpha}^{(k)}\right)_i$ for all k and all i (memory cost: $O(NK)$; computation cost: $O(N^2)$)

(3) compute and store $-\frac{1}{2}{\boldsymbol{\alpha}^{(k)}}^T D\boldsymbol{\alpha}^{(k)}$ for all k (memory cost: $O(K)$; computation cost: $O(N)$)

(4) for all $i \in \{1,\ldots,N\}$ (computation cost of the whole loop: $O(NK)$):

  (a) for all $k \in \{1,\ldots,K\}$, compute $d(\boldsymbol{\alpha}^{(k)}, \mathbf{x}_i)$ (in constant time)

  (b) assign $\mathbf{x}_i$ to its closest prototype

(5) return to 2 if the algorithm has not converged

3 Enforcing sparser prototypes

3.1 Impact of sparsity on the running time

As shown in the previous section, the natural sparsity of the prototypes leads to a $O(N^2)$ algorithm. While this is acceptable for reasonable values of N, this quadratic behavior reduces the scalability of the algorithm. Fortunately, sparser prototypes reduce further the complexity.

The computation cost of $(D\boldsymbol{\alpha})_i$ is indeed proportional to the number of nonzero terms in $\boldsymbol{\alpha}$. If we assume to have only P nonzero terms for each cluster prototype, computing $\left(D\boldsymbol{\alpha}^{(k)}\right)_i$ for all pairs (k,i) costs only $O(NPK)$. Moreover, ${\boldsymbol{\alpha}^{(k)}}^T D\boldsymbol{\alpha}^{(k)}$ can be computed based on the quantities needed for data assignment in $O(P)$ for each cluster.

Enforcing sparse prototypes can therefore in theory lead to a $O(NPK)$ algorithm.

3.2 Center of mass approximation

However, the optimal prototypes obtained by the relational K-means algorithm have $O(|C_k|)$ nonzero term. We have therefore to rely on an approximation method based on the following remark. In $\mathbb{R}^n$, n points in general positions are sufficient to generate the whole space as the linear combinations of these points. This means that while the center of mass of cluster C_k is conveniently represented by $\boldsymbol{\alpha}^{(k)} = \frac{1}{|C_k|}(\delta_{k,1},\ldots,\delta_{k,N})$, one might also choose a limited number of points in C_k and find coefficients for those points that also represent the center of mass.

Let us more precisely consider $J_k = \{j_{k,1},\ldots,j_{k,P}\}$ such that $\mathbf{x}_{j_{k,p}} \in C_k$ for all p. We are looking for a representation of the center of mass of C_k as a linear combination of the $\mathbf{x}_{j_{k,p}}$ (called the *support points* for this cluster), that is we want to solve the following constrained optimization problem:

$$
\begin{aligned}
(P_k)\quad & \min_{\boldsymbol{\alpha}\in\mathbb{R}^N} \sum_{i\in C_k} d(\boldsymbol{\alpha},\mathbf{x}_i), \\
& \text{with} \sum_{i=1}^{N}\alpha_i = 1;\ \forall j \notin J_k,\ \alpha_j = 0.
\end{aligned}
\tag{5}
$$

Let us denote $s_{k,j} = \sum_{i\in C_k} d_{ij}$, $D_J = (d_{uv})_{u\in J,\ v\in J}$ and $\mathbf{s}_{k,J} = \left(s_{k,j_{k,1}},\ldots,s_{k,j_{k,P}}\right)^T$. Then problem (P_k) can be rewritten:

$$
\begin{aligned}
(P_k)\quad & \min_{\boldsymbol{\beta}\in\mathbb{R}^P} {\mathbf{s}_{k,J}}^T\boldsymbol{\beta} - \frac{|C_k|}{2}\boldsymbol{\beta}^T D_J\boldsymbol{\beta}, \\
& \text{with} \sum_{i=1}^{P}\beta_i = 1.
\end{aligned}
\tag{6}
$$

The corresponding Lagrangian is

$$
L_k(\boldsymbol{\beta},\lambda) = {\mathbf{s}_{k,J}}^T\boldsymbol{\beta} - \frac{|C_k|}{2}\boldsymbol{\beta}^T D_J\boldsymbol{\beta} + \lambda\left(\sum_{j=1}^{P}\beta_j - 1\right),
\tag{7}
$$

whose gradient is given by

$$
\nabla L_k = \mathbf{s}_{k,J} - |C_k| D_J\boldsymbol{\beta} + \lambda\mathbf{1},
\tag{8}
$$

where $\mathbf{1} = (1,\ldots,1)^T \in \mathbb{R}^P$. Combining $\nabla L_k = 0$ and $\sum_{j=1}^{P}\beta_i = 1$, we obtain the following linear system:

$$
\begin{pmatrix} |C_k| D_J & -\mathbf{1} \\ \mathbf{1}^T & 0 \end{pmatrix}
\begin{pmatrix} \boldsymbol{\beta} \\ \lambda \end{pmatrix}
=
\begin{pmatrix} \mathbf{s}_{k,J} \\ 1 \end{pmatrix}.
\tag{9}
$$

Solving this system gives a local minimum of the problem. It should be noted that in the particular case of the squared Euclidean distance, if P is larger than the dimension of the data space, then the linear system will likely be underdetermined. It is therefore advisable to look for a least square solution to the problem (via, e.g., a singular value decomposition).

3.3 Sparse relational K-means

Using the sparse representation proposed above, we obtain the following algorithm:

(1) initialize randomly the clusters

(2) select P random points in each cluster (see Section 3.4 for details)

(3) compute a solution $\boldsymbol{\beta}_k$ to (P_k) for each cluster:

   (a) compute $\mathbf{s}_{k,J}$ (computational cost: $O(|C_k|P)$)

   (b) solve Equation 9 (computational cost: $O(P^3)$ with singular value decomposition)

(4) compute and store $(D\boldsymbol{\beta}_k)_i$ for all k and all i (computational cost: $O(NPK)$)

(5) compute and store $-\frac{1}{2}\boldsymbol{\beta}_k{}^T D\boldsymbol{\beta}_k$ for all k (computational cost: $O(PK)$)

(6) for all $i \in \{1, \ldots, N\}$ (computation cost of the whole loop: $O(NK)$)

   (a) for all $k \in \{1, \ldots, K\}$, compute $d(\boldsymbol{\alpha}^{(k)}, \mathbf{x}_i)$

   (b) assign $\mathbf{x}_i$ to its closest prototype

(7) return to 2 if the algorithm has not converged

The total cost is therefore $O(NPK + KP^3)$. As a consequence, this approach will reduce the running time if $PK \ll N$ and $P^2 \ll N$.

3.4 Choosing support points

Computing the optimal support points for a cluster is a combinatorial problem. It is therefore mandatory to rely on heuristic selection. In the case of the squared Euclidean distance, the center of mass of a cluster can obviously be recovered exactly with d points in general position, where d is the dimension of the data space. In practice, selecting randomly d points from the cluster leads to very good results.

The same strategy can be applied to general dissimilarities, with some additional care. In general, the sparse prototype won't coincide with the real center of mass. As a consequence, each random subset of P points will produce a different estimate of the center of mass. If the subset changes at each iteration, the algorithm might never converge. We propose therefore to select randomly P points from each cluster and then to track assignments so as to maintain the support points as constant as possible. When a support point is removed from one cluster, another support point is randomly selected in the new cluster. As will be shown in section 4 this strategy leads to very satisfactory results.

4 Experimental validation

4.1 Setup

The proposed algorithms have been implemented in Java and tested on a Athlon 64 3000+ Processor, under a 64 bits Linux operating system (with the 1.5 version of the Java virtual machine from Sun). Timing is done with a native library that leverages the posix high resolution timer API. To limit the impact of Just In Time compilation, the timing is done as follows. First, the dissimilarity matrix is loaded and the chosen algorithm is run to completion once. Then, in the same virtual machine, the algorithm is run again ten times. The reported figure is the median value of the ten last runs, while the first one is discarded. Even for very short running times, the combination of this strategy and of the native high resolution timer results in very small relative variations between each run and therefore to reliable estimates of the running time of the algorithms. We report the average time spend per epoch in each situation.

4.2 Memory layout

Our main targets are datasets for which the dissimilarity matrix fits into the main memory of a recent personal computer, i.e. that occupies roughly a maximum of one gigabyte of memory. With single precision floating point values (4 bytes per value), this gives a maximum of approximately 23 000 observations. However, this implies storing only the upper (or lower) half of the dissimilarity matrix and induces therefore memory locality problems.

As pointed out in section 2.3, relational K-means operates by using sums of the form $\sum_{j \in C_k} d_{ij}$. If one stores the full dissimilarity matrix, the dissimilarities from $\mathbf{x}_i$ to any other data point are close to one another in the main memory. For instance, 23 000 such dissimilarities occupy 96 Kilo bytes of memory. As a consequence, the processor can load the corresponding memory block in its fast cache[1] and then operate in this cache to compute $\sum_{j \in C_k} d_{ij}$ (on the test hardware, the cache is 512 kilobytes large, while the main memory measures 3 gigabytes).

On the contrary, storing the upper half of the dissimilarity matrix implies to spread in the whole memory the dissimilarities needed to compute sums of the form $\sum_{j \in C_k} d_{ij}$. As a consequence, the cache is far less efficient in this calculations, leading, in theory, to an order of magnitude slower algorithms. In practice, there is a complex balance between the amount of memory used and the memory access pattern, which means that in some situations, using half dissimilarity matrix might reduce the running time. However, this is very unlikely for the standard relational K-means: this algorithm must indeed access to all the dissimilarities at each epoch and is strongly impaired by non locality.

To illustrate this point, we give a simple example with a moderate size dataset. We consider $N = 5\,000$ points randomly chosen in the unit square (in $\mathbb{R}^2$) with the squared Euclidean distance and we measure the running time of the relational K-means for $K = 20$ clusters. Results are reported in the following table:

[1] This a very simplified description of the processor caching strategy, but additional details are not needed to understand the problem of memory locality.

Relational K-means

Storage method	mean time per epoch
Full matrix	94.6 ms
Upper half	394 ms

In practice, one has to wait around one minute[2] to get a clustering result (the best out of 10 random initial states) in the first case against around four minutes in the second case. As this difference in running time increases very quickly with the size of the dataset, the half storage leads quickly to very high computation load. For instance, with $N = 15\ 000$ and $K = 10$, the mean time per epoch is more than 10 seconds, leading to running times of the hour order that are completely incompatible with exploratory analysis. In practice the standard relational K-means must be used with full dissimilarity matrices and is therefore limited to roughly 15 000 observations for one gigabyte of main memory.

The sparse approach proposed in this paper is far less sensitive to memory locality, mainly because it scans only NPK dissimilarity values for each epoch. When PK is small, this reduces a lot the effects of non locality. In the same experimental setting and with $P = 3$, we obtain the following timing:

Sparse Relational K-means

Storage method	mean time per epoch
Full matrix	5 ms
Upper half	4.8 ms

In practice, results are obtained in 15 seconds for the full storage and in 14 seconds for the half storage. The sparse approach can therefore be used with the half storage solution and scales up to 23 000 observations.

4.3 Running time

We first validate complexity models. The cost per epoch depends only on the considered sizes (number of observations, of clusters and of support points) and not on the actual values contained in the dissimilarity matrix (even if in practice, the exact time needed to solve Equation 9 might depend on the involved values). We decided therefore to mainly validate the cost models on simple Euclidean data. As explained in the previous section, we limit ourselves to a maximum of 15 000 observations with a full representation of the dissimilarity matrix. Data are generated in the unit square in $\mathbb{R}^2$ and compared with the squared Euclidean distance. We first use a fixed number of $K = 10$ clusters to illustrate the quadratic behavior of the relational K-means. The following table gives the mean time per epoch as a function of N:

Relational K-means

N	3 000	5 000	7 500	10 000	12 500	15 000
mean time (ms)	32.2	88.9	198	366	592	873

A quadratic model of the form time= $\alpha N^2 + \beta N + \gamma$ fits correctly the data (mean absolute relative error: 1.6%).

On the same data and with $P = 3$, the sparse relational K-means has a roughly linear behavior and a much smaller running time:

Sparse Relational K-means

N	3 000	5 000	7 500	10 000	12 500	15 000
mean time (ms)	5	9.4	14.2	19.5	25.4	31

A linear model of the form time= $\alpha N + \beta$ fits the observations correctly (mean absolute relative error: 2.1%).

We then study the influence of P on the running time of the sparse relational K-means. In theory, the rapid growth of $O(KP^3)$ limits drastically the value of P for small values of N: with $N = 3\ 000$ and $K = 10$, for instance, P cannot exceed 7 without dominating the complexity. In practice however, the behavior of the algorithm is more complex than expected as shown in the following table, which reports running time for $N = 3\ 000$ and $K = 10$:

Sparse Relational K-means

P	3	4	6	10	20	50
mean time (ms)	5	5.5	6.2	7.8	14.1	53.4

As P remains small compared to N, solving Equation 9 takes less relative time than expected which leads to better scalability than expected in terms of P. We obtained similar figures for other values of N. For instance with $N = 15\ 000$, the mean running time per iteration is 495 ms with $K = 10$ and $P = 100$, still below the value obtained by the standard relational K-means.

We finally study the effect on K on the running time. In theory, the standard relational K-means should be unaffected, whereas the sparse version should behave linearly with K (for a fixed value of P). In practice, the running time of the standard algorithm increases with K: this an expected consequence of the additional bookkeeping induced by a larger number of clusters. The following table gives an example with $N = 7\ 500$:

Relational K-means

K	10	20	40	80	160
mean time (ms)	198	209	222	246	301

For the sparse version, the behavior is roughly linear as shown on the following table for the same dataset (and $P = 3$):

Sparse Relational K-means

K	10	20	40	80	160
mean time (ms)	14.2	21.2	37.4	72.1	137

In summary, while the dependency on N is quite clear (quadratic for the standard algorithm and linear for the sparse one), the cost model is not accurate enough to predict reliably the expected running time for a given configuration (i.e., when taking into account both the number of clusters and the number of support points), even if small values of P always lead to short running times.

[2] including virtual machine start up and data loading

4.4 Clustering quality

Another important question is obviously the quality of the clustering obtained via the sparse approach. To investigate it, with first reused similar artificial data as the ones used in the previous section, but in higher dimension. The following table compares for instance the clustering results for $N = 5\ 000$ points randomly chosen in $[0, 1]^{50}$ clustered in $K = 50$ clusters. Reported figures are the percentage of running time used by the sparse algorithm and the percentage of increase in the final error, as a function of P (the running time corresponds to 10 random configurations, keeping the best error):

Sparse Relational K-means

P	2	5	10	15	20
running time	18%	28%	64%	94.4%	118%
error increase	3.2%	2.6%	1.7%	1.1%	0.7%

Similar results have been obtained with different parameters, except that, for smaller number of clusters, as shown in the previous section, the running time remains lower for higher values of P. An interesting aspect is that despite a lower value of K for a fixed P corresponds to a sparser representation, the quality of the obtained clusters doesn't decrease. For instance for the same dataset with $K = 20$ and $P = 2$, the error increases only by 2.1% for 9% of the running time.

As a final validation, we have tested the proposed approach on an artificial string dataset. $N = 10\ 000$ random strings of length uniformly distributed between $[5, 15]$ and compared via the string edit distance. We report the same figure as the previous ones on the following table, for $K = 50$ clusters:

Sparse Relational K-means

P	2	5	10	15	20
running time	7.1%	11.8%	17.9%	27.3%	68.4%
error increase	5.2%	3.8%	2.4%	1.6%	1.1%

Results are very satisfactory and compatible with those obtained with the squared Euclidean distance.

5 Conclusion

We have proposed a sparse version of the relational K-means algorithm with a complexity of $O(NKP + KP^3)$, where P corresponds to the number of support points in each cluster, a parameter under user control. This complexity is a major improvement over the cost of the standard relational K-means algorithm which scales as $O(N^2)$. We have also shown on simulated data that the running times of the proposed algorithm were much smaller than the ones of the standard algorithm, while the obtained clusters were only marginally worse than best ones, even with very small values for P. Future works include validation on real world data as well as extension to relational topographic mapping algorithms [7].

References

[1] C. Ambroise and G. Govaert. Analyzing dissimilarity matrices via Kohonen maps. In *Proceedings of 5th Conference of the International Federation of Classification Societies (IFCS 1996)*, volume 2, pages 96–99, Kobe (Japan), March 1996.

[2] C. Bahlmann and H. Burkhardt. The writer independent online handwriting recognition system *frog on hand* and cluster generative statistical dynamic time warping. *IEEE Trans. Pattern Anal. and Mach. Intell.*, 26(3):299–310, Mar. 2004.

[3] B. Conan-Guez, F. Rossi, and A. El Golli. Fast algorithm and implementation of dissimilarity self-organizing maps. *Neural Networks*, 19(6–7):855–863, July–August 2006.

[4] M. Cottrell, B. Hammer, A. Hasenfuß, and T. Villmann. Batch and median neural gas. *Neural Networks*, 19(6–7):762–771, July–August 2006.

[5] T. Graepel, M. Burger, and K. Obermayer. Self-organizing maps: Generalizations and new optimization techniques. *Neurocomputing*, 21:173–190, November 1998.

[6] T. Graepel and K. Obermayer. A stochastic self-organizing map for proximity data. *Neural Computation*, 11(1):139–155, 1999.

[7] B. Hammer, A. Hasenfuss, and M. Rossi, Fabrice annd Strickert. Topographic processing of relational data. In *Proceedings of the 6th Workshop on Self-Organizing Maps (WSOM 07)*, Bielefeld (Germany), September 2007.

[8] R. J. Hathaway and J. C. Bezdek. Nerf c-means: Non-euclidean relational fuzzy clustering. *Pattern Recognition*, 27(3):429–437, March 1994.

[9] R. J. Hathaway, J. W. Davenport, and J. C. Bezdek. Relational duals of the c-means clustering algorithms. *Pattern Recognition*, 22(2):205–212, 1989.

[10] L. Kaufman and P. J. Rousseeuw. Clustering by means of medoids. In Y. Dodge, editor, *Statistical Data Analysis Based on the L1-Norm and Related Methods*, pages 405–416. North-Holland, 1987.

[11] T. Kohonen. Self-organizing maps of symbol strings. Technical report A42, Laboratory of computer and information science, Helsinki University of technology, Finland, 1996.

[12] T. Kohonen and P. J. Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21:19–30, 1998.

[13] T. Kohonen and P. J. Somervuo. How to make large self-organizing maps for nonvectorial data. *Neural Networks*, 15(8):945–952, 2002.

[14] F. Rossi. Model collisions in the dissimilarity SOM. In *Proceedings of XVth European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 25–30, Bruges (Belgium), April 2007.