Genetic Algorithm for Multidimensional Scaling over Mixed and Incomplete Data

P. Tecuanhuehue-Vera, Jesús Ariel Carrasco-Ochoa, and José Fco. Martínez-Trinidad

National Institute for Astrophysics, Optics and Electronics
Luis Enrique Erro No. 1, Sta. Maria Tonantzintla, Puebla, Mexico
{pedrish,ariel,fmartine}@ccc.inaoep.mx

Abstract. Multidimensional scaling maps a set of *n*-dimensional objects into a lower-dimension space, usually the Euclidean plane, preserving the distances among objects in the original space. Most algorithms for multidimensional scaling have been designed to work on numerical data, but in soft sciences, it is common that objects are described using quantitative and qualitative attributes, even with some missing values. For this reason, in this paper we propose a genetic algorithm especially designed for multidimensional scaling over mixed and incomplete data. Some experiments using datasets from the UCI repository, and a comparison against a common algorithm for multidimensional scaling, shows the behavior of our proposal.

Keywords: Multidimensional scaling, Mixed and incomplete data, Genetic algorithms.

1 Introduction

Multidimensional scaling consists in finding points, usually in the Euclidean plane, that represent objects from another space with higher dimension; in such a way that distances between points in the Euclidean plane coincide, as much as possible, with distances between original objects[1]. Applying multidimensional scaling, over high dimensional objects, allows visualizing how original objects are distributed on the original space.

MDSCAL[2], ISOMAP[3] and LLE[4] are among the most common algorithms for multidimensional scaling. These algorithms use classical optimization techniques, as gradient methods, but using these techniques could lead to local optima. Besides, these algorithms were designed for numerical data. However, in soft sciences like medicine, geology, sociology, etc, there are data analysis problems were objects could be described through quantitative and qualitative attributes (mixed data); moreover, some attribute values could be unknown (mixed and incomplete data). Therefore, current algorithms cannot be directly applied.

For these reasons, in this work we introduce a genetic algorithm for multidimensional scaling over mixed and incomplete data. Using genetic algorithms allows escaping from local optima, allowing a wider exploration of the search space. In this way, it is possible to find a better representation of the original high dimensional objects over the Euclidean plane. In order to reach this objective, in this work, we also introduce new mutation and crossover operators; specifically designed for multidimensional scaling.

The rest of this document is organized as follows. Section 2 provides a brief description of previous algorithms for multidimensional scaling. Section 3 describes those functions that are commonly used for evaluating how good a representation obtained by a multidimensional scaling algorithm is; additionally, in this section, the used distance functions are shown. Section 4, introduces the proposed genetic algorithm, as well as, the new mutation and crossover operators. Section 5 shows some experiments and a comparison against other multidimensional scaling algorithm. Finally, section 6 exposes our conclusions.

2 Related Work

For multidimensional scaling, several algorithms have been proposed; for example: MDSCAL[2], ISOMAP[3], LLE[4], HiperMap[5], FastMap[6], etc. In this section, we describe some of the most common algorithms for multidimensional scaling. These algorithms could be divided according to the way they use for facing the multidimensional scaling problem; i.e., using the global structure of the original data, as, for example, MDSCAL[2] and the algorithm proposed in this paper; or using a local approach through neighborhoods, as, for example, ISOMAP[3] and LLE[4].

2.1 MDSCAL

MDSCAL, proposed by J. B. Kruskal[2], is based on the work of Shepard[7]. MDSCAL uses the gradient optimization algorithm in order to minimize the STRESS function, which evaluates the quality of the Euclidean plane representation (see section 3). The gradient optimization method consists in: starting from an arbitrary initial configuration, to improve this solution by moving it in the direction where the objective function decreases fastest, which is known as the negative gradient direction. This process is repeated until reaching a point that is not possible to improve, ideally when all partial derivatives are zero, which indicates that this point is a local minimum, but it may not be a global minimum.

2.2 ISOMAP

Tenenbaum[3], introduces an algorithm called ISOMAP, which tries to discover the intrinsic structure of a data set, by mean of preserving the local structure. This characteristic is considered as an advantage over algorithms as MDSCAL that are designed to maintain the global structure of the whole dataset.

ISOMAP is based on MDSCAL. The main idea is estimating geodesic distances between distant points using only the distances in the original space. For obtaining distances between close points, the geodesic distance can be used in a simple way;

and for distant points, the geodesic function can be approximated by a sequence of jumps to close points. These jumps can be efficiently computed through a search of shortest paths in a graph *G* with edges connecting each point with its close neighbors.

ISOMAP consists of three steps: First, the neighbors of each point are determined; this can be done by relating each point with all points into a neighborhood with fixed radius *r*, or relating each point with its *K* nearest neighbors. These relations are represented as a weighted graph, where weights represent the distance between neighbors. After, the geodesic distances among every pair of points are approximated by computing the shortest paths in the graph *G*, using the Dijkstra algorithm[8]. Finally, in order to build a representation on the Euclidean plane, ISOMAP applies MDSCAL over the distance matrix obtained from the graph *G*.

2.3 LLE

LLE, proposed by Roweis and Lawrence[4], tries to reduce the dimensionality through local structures, mapping objects using local neighborhoods. LLE uses the nearest neighbors of each point $\vec{X}_i$, assigning weights W_{ij} to each neighbor $\vec{X}_j$. Then, it tunes the weights by minimizing the function (1), using the least-squares method.

$$\mathcal{E}(W) = \sum_i |\vec{X}_i - \sum_j W_{ij}\vec{X}_j|^2 \qquad (1)$$

After, LLE computes those points $\vec{Y}_i$, in the Euclidean plane, that best rebuilds the neighborhood. The points$\vec{Y}_i$ are obtained by minimizing function (2), also using the least-squares method.

$$\Phi(Y) = \sum_i |\vec{Y}_i - \sum_j W_{ij}\vec{Y}_j|^2 \qquad (2)$$

3 Evaluation and Distance Functions

In this section, we describe some functions that are commonly used for evaluating how good a representation obtained by a multidimensional scaling algorithm is.

3.1 Evaluation Functions

In the literature, there are some measures for evaluating how good a representation of a high dimensional dataset on the Euclidean plane is. The mean square error (MSE), the STRESS[9] function and a variant of the last called SSTRESS[10], are among the most common evaluation functions used for multidimensional scaling.

The mean square error (MSE) is defined as:

$$MSE = \frac{1}{m^2}\sum_{ij}(d_{ij} - D_{ij})^2 \qquad (3)$$

whereD_{ij} is the distance between the *n*-dimensional objects o_i and o_j; $d_{ij}$is the distance between the points in the Euclidean plane representing the objects o_i and o_j; and *m* is the number of *n*-dimensional objects to be mapped on the Euclidean plane. Smaller values of the MSE measure correspond to better representations.

The STRESS function is defined as:

$$STRESS = \sqrt{\frac{\sum_{ij}(d_{ij} - D_{ij})^2}{\sum d_{ij}^2}} \tag{4}$$

whered_{ij} and$D_{ij}$are defined as before. Again, smaller values of the STRESS function correspond to better representations.

A variant of the STRESS function is the SSTRESS function, which is defined as:

$$SSTRESS = \sqrt{\frac{\sum_{ij}({d_{ij}}^2 - {D_{ij}}^2)^2}{\sum_{ij}(d_{ij}^2)^2}} \tag{5}$$

whered_{ij} andD_{ij} are defined as before. Also, smaller values of the SSTRESS function correspond to better representations.

3.2 Distance Functions

In this section, we define the distance functions used for comparing n-dimensional objects, as well as, points on the Euclidean plane. For numerical data we used the Euclidean distance [1] and for mixed data we used the Heterogeneous Overlapped Euclidean Measure (HEOM)[11]. This function was designed for directly comparing objects described by quantitative and qualitative attributes, where some values could be unknown (mixed and incomplete data). Let $\{O_1, O_2, O_3, \dots, O_m\}$ be a set of *m n*-dimensional objects.

$$HEOM(O_i, O_j) = \sqrt{\sum_{a=1}^{n} d_a(x_a, y_a)^2} \tag{6}$$

wherex_a and y_a, a=1,...,*n*, represent the values of the attributes of the objects O_i andO_j, respectively.

$$d_a(x,y) = \begin{cases} 1 & if\ x\ or\ y\ are\ missing \\ overlap(x,y) & if\ a\ is\ qualitative \\ rn_{diff_a}(x,y) & if\ a\ is\ quantitative \end{cases} \tag{7}$$

$$overlap(x,y) = \begin{cases} 0 if\ x = y \\ 1 otherwise \end{cases} \tag{8}$$

$$rn_{diff_a}(x,y) = \frac{|x-y|}{max_a - min_a} \tag{9}$$

where max_a and min_a are the maximum and minimum values of the attribute *a*.

4 Proposed Method

Genetic algorithms are heuristic optimization and search methods, which try to follow the natural evolution process. These algorithms maintain a set (population) of possible solutions (individuals), usually initialized in a random way, which are combined using genetic operators (mutation, crossover, etc.), during some iterations (generations). In each generation, all individuals from the population are evaluated (fitness) and some of the best individuals are selected for the population of the next generation. Genetic algorithms have the ability of escaping from local optima, since the mutation operator allows a wider exploration of the search space. Algorithm 1 shows the generic genetic algorithm used in this work, which is based on the basic genetic algorithms[14].

For the above mentioned reasons, we have chosen genetic algorithms for solving the multidimensional scaling problem over mixed and incomplete data. However, common mutation and crossover operators are designed to solve general optimization problems; but for some problems, like multidimensional scaling, they produce a slow convergence rate, or sometimes avoid finding a suitable solution. Therefore, in our approach, we propose new mutation and crossover operators, specially designed for speeding up the search of a suitable solution for multidimensional scaling avoiding to fall in local optima. Additionally, we consider that it is important to start from a non-completely random population, in order to reduce the number of generations needed to find a good solution. Thus, we also propose a new way for selecting the initial population.

Algorithm 1: Generic genetic algorithm
```
1.  POPULATION=create_Initial_Population( P_SIZE)
2.  For i=1 to G-generations
3.      Evaluate fitness of each individual in POPULATION
4.      NEXT_POPULATION←Best K individuals form POPULATION
5.      For j=1 to P_SIZE-K
6.          Randomly choose two individuals P and Q from POPULATION
7.          (R,S)=Crossover(P,Q);
8.          if ( mutation_probability(R,S) )
9.              then
10.                 R'=Mutation(R)
11.                 S'=Mutation(S)
12.                 NEXT_POPULATION ← NEXT_POPULATION ∪ {R',S'}
13.             else
14.                 NEXT_POPULATION ← NEXT_POPULATION ∪ {R,S}
15.     POPULATION ←NEXT_POPULATION
16. End
```

4.1 Individual Representation

Individuals in our approach for multidimensional scaling will be a sequence of real numbers; where each consecutive pair corresponds with the representation on the Euclidean plane of an object in the original n-dimensional space (see Fig. 1).

n-dimensional objects	O_1		O_2		, ... ,	O_m	
	↓		↓			↓	
Points on the Euclidean plane	$(X_1$	$Y_1)$	$(X_2$	$Y_2)$	, ... ,	$(X_m$	$Y_m)$

Fig. 1. Individuals of the proposed genetic algorithm

4.2 Initial Population

The objective of creating a non-completely random initial population is to have a better initial approximation, which allows getting a good solution in less iterations. We propose to generate the initial population using bidimensional projections of the original n-dimensional objects. For objects described by n attributes, it is possible to build $n(n-1)/2$ bidimensional projections. First, we build all bidimensional projections (see Fig. 2). Then, in order to know how good each projection is for representing the original n-dimensional objects, each bidimensional projection is evaluated using the STRESS function,. The best K projections are included in the initial population. Additionally, we add to the initial population those individuals obtained by moving each point of the best bidimensional projection, around a neighborhood with a given radius r.

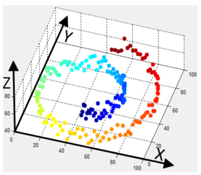
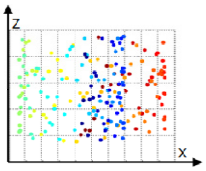
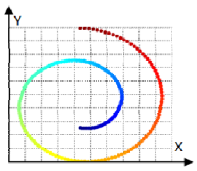
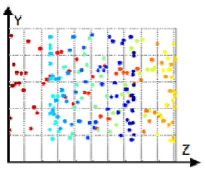
			
(a) Tridimensional objects with attributes X,Y,Z.	(b) Projection using attributes X,Z.	(c) Projection using attributes X,Y	(d) Projection using attributes Y,Z

Fig. 2. Bidimensional projections of tridimensional objects

4.3 Mutation

In order to speedup the search of good representations, we propose a new mutation operator based on locally positioning 3 randomly selected points, such that the distances among these 3 points coincide exactly with the distances of the corresponding n-dimensional objects.

We randomly select tree points a,b,c from an individual, that represent the n-dimensional objects A,B,C. Then, in order to generate the new points a',b',c', we move each point in a different way, as follows (see Fig. 3):

a) To allow exploring different parts of the search space, the point a is randomly moved to the position a' into a neighborhood with a given radius r(see Fig.3b).

b) Point *b* is moved to its correct position regarding *a'*, i.e., the distance between *a' and b'* must be the same than the distance between the objects *A* and *B* in the original *n*-dimensional space. In order to obtain the new position *b'*, we move the point *b* in the direction of the straight line connecting *b* with *a'*(see Fig. 3c)

c) Point *c* is moved to its correct position regarding *a'* and *b'*, i.e., the distance between *c'* and *a',* as well as, between *c'* and *b',* must be the same as the distance, in the original *n*-dimensional space, between the objects *A* and *B*and the objects *B* and *C,* respectively. (see Fig. 3d)

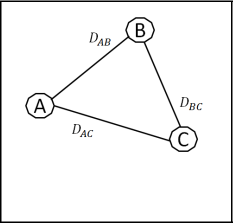
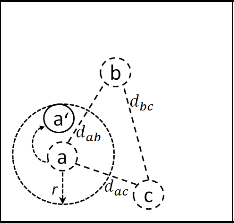
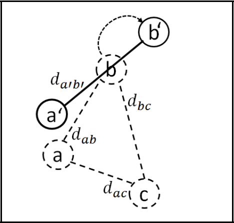
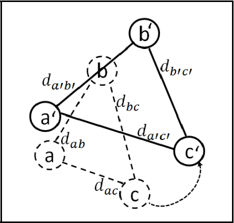
			 b' $d_{a'b'}$ b $d_{b'c'}$ a' d_{bc} $d_{a'c'}$ d_{ab} c' a d_{ac} c
(a) *n*-dimensional objects *A, B, C*	(b) Point *a* is randomly moved to *a'*	(c) Point *b* is moved to its correct position *b'* regarding *a'*	(d) Point *c* is moved to its correct position *c'* regarding *a'* and *b'*

Fig. 3. Mutation operator

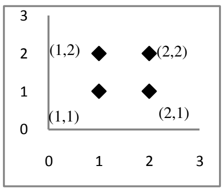
4.4 Crossover

The main objective of the crossover operator is to combine the best characteristics from two individuals in order to produce new individuals that could be a better solution. Following this idea, given a set of *n*-dimensional objects $O=\{O_1,O_2,\ldots,O_m\}$ and two individuals $P=\{(x_{1,1},x_{1,2}), (x_{2,1},x_{2,2}), \ldots, (x_{m,1},x_{m,2})\}$ and $Q=\{(y_{1,1},y_{1,2}),(y_{2,1},y_{2,2}),\ldots,(y_{m,1},y_{m,2})\}$, as well as their respective distance matrices *M(O)*, *M(P)* and *M(Q).* In order to obtain an offspring from *P* and *Q*, first we obtain the sum of each column in the distance matrices (see Fig. 4)

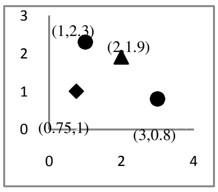
$$M(O)=\begin{array}{ccc}\left[\begin{matrix} D_{1,1}D_{1,2} & \cdots & D_{1,m} \\ \vdots & \ddots & \vdots \\ D_{m,1}D_{m,2} & \cdots & D_{m,m}\end{matrix}\right] \\ S(O_1) \qquad\qquad S(O_m)\end{array} M(P)=\begin{array}{c}\left[\begin{matrix} d_{x^{1,1}}d_{x^{1,2}} & \cdots & d_{x^{1,m}} \\ \vdots & \ddots & \vdots \\ d_{x^{m,1}}d_{x^{m,2}} & \cdots & d_{x^{m,m}}\end{matrix}\right] \\ S(P_1) \qquad\qquad S(P_m)\end{array} M(Q)=\begin{array}{c}\left[\begin{matrix} d_{y^{1,1}}d_{y^{1,2}} & \cdots & d_{y^{1,m}} \\ \vdots & \ddots & \vdots \\ d_{y^{m,1}}d_{y^{m,2}} & \cdots & d_{y^{m,m}}\end{matrix}\right] \\ S(Q_1) \qquad\qquad S(Q_m)\end{array}$$

Fig. 4. Distance matrices for a set of *n*-dimensional objects $O=\{O_1,O_2,\ldots,O_m\}$ and two individuals $P=\{(x_{1,1},x_{1,2}),(x_{2,1},x_{2,2}),\ldots,(x_{m,1},x_{m,2})\}$ and $Q=\{(y_{1,1},y_{1,2}),(y_{2,1},y_{2,2}),\ldots,(y_{m,1},y_{m,2})\}$

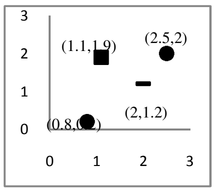
Then, we choose those points that their sum of distances to all other points is more similar to the sum of the distances from the corresponding *n*-dimensional object to all other objects. This is, for selecting the point that will represent the *n*-dimensional object O_i in the offspring, we choose the corresponding point from *P* iff $|S(O_i)-S(P_i)|\leq|S(O_i)-S(Q_i)|$, otherwise we choose the corresponding point from *Q* (see Fig. 5).



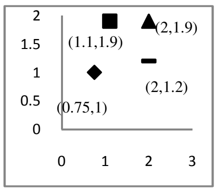
(a) *n*-dimensional objects



(b) An individual *P* STRESS= 0.4080



(c) An individual *Q* STRESS=0.3766



(d) Offspring obtained from *P* and *Q* STRESS=0.1868

Fig. 5. Crossover operator

5 Experiments

In our experiments, we used five datasets from the UCI[12] repository, four with mixed data and one numerical dataset (see Table 1). The last was included to evaluate the performance of the proposed genetic algorithm in numerical datasets. We compare the results of the proposed genetic algorithm against MDSCAL[2], which follows a global approach as the proposed genetic algorithm. We did not compare against ISOMAP[3] and LLE[4],because these algorithms follow a local approach. Since genetic algorithms have a random component, the results of our algorithm correspond to the average of 10 executions. All experiments were done on a PC computer with an1.6GHzAMD processor, with 2GB of RAM, running Windows 7. The proposed algorithm was implemented in MATLAB 7.10.0. For MDSCAL, we used the implementation included in MATLAB 7.10.0.

In our proposed genetic algorithm, we used those parameter values suggested by Schaffer in [13], i.e., 30 individuals in the initial population, 0.95 for crossover probability, and 1.0 for mutation probability. The last choice is because the proposed mutation operator allows fixing the relative distances of tree points; and applying it repeatedly would lead to improve the global representation. For the radius, we made tests using *r*=1.0, *r*=0.5 and *r*=0.01, in order to evaluate the sensitiveness of the proposed genetic algorithm to different values of the radius. We fix the maximum number of generations to 20, since empirically we realize that increasing the number of generations do not contribute to obtain better solutions for the multidimensional scaling problem. For comparing objects in the mixed and incomplete datasets we used the HEOM distance function, and for the numerical dataset we used the Euclidean distance.

Table 1. Databases used for the experiments

Datasets	objects	attributes	quantitative attributes	qualitative attributes	objects with missing values
Balloons	16	4	0	4	0
Servo	162	4	2	2	0
Post operative	79	9	1	8	3
Pittsburgh bridges	80	13	3	10	38
Hayes roth	133	6	6	0	0

Table 2 shows the results obtained by the proposed genetic algorithm, for each radius, and MDSCAL. From this table, we can notice that for mixed and incomplete datasets, our algorithm obtained better quality representations than MDSCAL in most of the cases, mainly using r=0.5. The only dataset where MDSCAL obtained better results was *Balloons*, which is a very small dataset where traditional optimization techniques can reach good results. For the numerical dataset, as it was expected, MDSCAL obtained the best result, but the difference with the proposed method was 1.4×10^{-3}. Also, it can be noticed that our genetic algorithm gets different performance for different values of r. In general, too small values could lead to slow convergence rates avoiding finding good solutions in a few iterations; on the other hand, too big values could lead to instability, which also could produce less quality solutions.

Table 2. Comparison of the proposed GA against MDSCAL

Dataset	Fitness	GA r=1	GA r=0.5	GA r=0.01	MDSCAL
Balloons	Stress	0.3383	0.3248	0.3606	**0.3149**
Servo	Stress	0.4077	**0.4027**	0.4338	0.6456
Post operative	Stress	0.3840	**0.3701**	0.4222	0.6424
Pittsburgh bridges	Stress	0.4051	0.4113	**0.4050**	0.9984
Hayes roth	Stress	0.0062	0.0062	0.0052	**0.0038**
Average		0.3083	**0.3030**	0.3254	0.5210

6 Conclusions

In this paper, we have introduced a genetic algorithm for multidimensional scaling over mixed and incomplete data. In order to speed up the search, we proposed to generate the initial population using bidimensional projections of the original n-dimensional objects also introduced new mutation and crossover operators, which were specially designed for multidimensional scaling.

From the experiments, we can conclude that, for mixed and incomplete datasets, our algorithm allows getting better results that MDSCAL, obtaining representations with up to 60% smaller STRESS values. For numerical datasets our algorithm performs well obtaining results very similar to those obtained by conventional algorithms for multidimensional scaling.

As future work, based on the results presented in this paper, we will study new mutation and crossover operators, which could improve even more the efficiency and efficacy of the genetic algorithm. Another research line could be studying the sensitivity of the proposed algorithm to the radius parameter.

References

1. Cox, T.F., Cox, M.A.A.: Multidimensional Scaling. Chapman & Hall, London (1994)
2. Kruskal, J.B.: Non metric multidimensional scaling: A numerical method. Psychometrika 29(2), 115–129 (1964)

3. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290, 2319–2323 (2000)
4. Roweis, S.T., Lawrence, K.S.: Nonlinear dimensionality reduction by locally linear embedding. Science 290, 2323–2326 (2000)
5. An, J., Xu, Y.J., Ratanamahatana, C.A., Chen, Y.P.P.: A dimensionality reduction algorithm and its application for interactive visualization. Journal of Visual Languages and Computing 18, 48–70 (2006)
6. Faloutsos, C., Lin, K.L.: FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In: ACM SIGMOD International Conference on Management of Data, pp. 163–174 (1995)
7. Shepard, R.N.: The analysis of proximities: Multidimensional scaling with an unknown distance function. Psychometrika 27(2), 125–140 (1962)
8. Dijkstra, E.W.: A note on two problems in connection with graphs. Numerische Mathematik 1, 269–271 (1959)
9. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a non metric hypothesis. Psychometrika 29(1), 1–27 (1964)
10. Takane, Y., Young, F.W., de Leeuw, J.: Non metric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. Psychometrika 42, 7–67 (1977)
11. Wilson, D.R., Martinez, T.: Improved heterogeneous Distance Functions. Journal of Artificial Intelligence Research (JAIR) 6(1), 1–34 (1997)
12. Merz, C., Murphy, P.: UCI repository of machine learning databases. Technical report, Univ. of California at Irvine, Department of Information and Computer Science (1998)
13. Schaffer, J.D., Caruana, A.R., Eshelman, L.J., Das, R.: A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In: Proceedings of the Third International Conference on Genetic Algorithms, pp. 51–60. Morgan Kaufmann Publishers, San Mateo (1989)
14. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)