Random search with $k$-prototypes algorithm for clustering mixed datasets

BY DUC-TRUONG PHAM$^{1,2}$, MARIA M. SUAREZ-ALVAREZ$^1$,* AND YURIY I. PROSTOV$^3$

$^1$Manufacturing Engineering Centre, School of Engineering, Cardiff University, Cardiff CF24 3AA, UK
$^2$Department of Information Systems, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia
$^3$Department of Higher Mathematics, Moscow Institute of Radio Engineering, Electronics and Automation—Technical University, 78 Vernadskogo pr., Moscow 117454, Russia

A new algorithm to cluster datasets with mixed numerical and categorical values is presented. The algorithm, called RANKPRO (random search with $k$-prototypes algorithm), combines the advantages of a recently introduced population-based optimization algorithm called the bees algorithm (BA) and $k$-prototypes algorithm. The BA works with elite and good solutions, and continues to look for other possible extrema solutions keeping the number of testing points constant. However, the improvement of promising solutions by the BA may be time-consuming because it is based on random neighbourhood search. On the other hand, an application of the $k$-prototypes algorithm to a promising solution may be very effective because it improves the solution at each iteration. The RANKPRO algorithm balances two objectives: it explores the search space effectively owing to random selection of new solutions, and improves promising solutions fast owing to employment of the $k$-prototypes algorithm. The efficiency of the new algorithm is demonstrated by clustering several datasets. It is shown that in the majority of the considered datasets when the average number of iterations that the $k$-prototypes algorithm needs to converge is over 10, the RANKPRO algorithm is more efficient than the $k$-prototypes algorithm.

Keywords: clustering; mixed datasets; random search; $k$-prototypes algorithm

1. Introduction

One can understand very sophisticated phenomena by breaking objects into smaller homogeneous parts that can be each analysed and explained separately. For a given dataset, any clustering produced by an algorithm or a human is a hypothesis to suggest (or explain) groupings in the data. This hypothesis is selected among a large set of possibilities and is represented in a formal way. The chosen hypothesis becomes a model for the data. Hence, clustering can potentially constitute a mechanism to classify unlabelled instances of the data.

*Author for correspondence (suarez-alvarezm@cardiff.ac.uk).
This is why clustering algorithms have been studied so extensively (Anderberg 1973). In particular, efficient clustering is a fundamental task in data science, where the goal is to discover similarities within a dataset.

In a number of fields of data mining, an object is represented by a vector variable, namely the feature vector $A$ (Jain et al. 1999). In database applications, the features are called attributes, and $A$ is defined as a set of attributes, $A = (A_1, A_2, \ldots, A_{p+l})$. Each attribute can take on a finite or infinite (continuous) number of possible values. In many traditional applications, it is assumed that all the features are the same type. However, real-life datasets are often mixed, i.e. they consist of both numerical and categorical types. Only two data types of attributes are considered here, namely numerical and categorical because all kinds of attributes can be mapped to these two types. For mixed data, the vector of features $A$ can be split into the vector of numerical features $A^n = (A^n_1, \ldots, A^n_p)$ and the vector of categorical features $A^c = (A^c_1, \ldots, A^c_l)$, i.e. it is represented as $A = (A^n, A^c)$. For numerical or quantitative features, the feature domain $\text{Dom}(A_j)$ can be represented on the real line, while for categorical features (sometimes called nominal or qualitative), the domain is a finite set of different states. Categorical features may be substituted by numerical codes for different states of the feature. A database can then be represented as a matrix of size $N \times (p + l)$, where $N$ is the number of records, $p$ the number of numerical attributes and $l$ the number of categorical attributes. In this case, the $i$-th row of the matrix corresponds to the $i$-th record of the database ($1 \leq i \leq N$). This row is a vector $(x_{i1}, \ldots, x_{ip}, y_{i1}, \ldots, y_{il})$, whose values $x_{i1}, \ldots, x_{ip}$ are numerical, while the values $y_{i1}, \ldots, y_{il}$ are categorical.

Any measure of the degree of closeness (likeness) is called similarity measure (Looney 1997). In clustering analysis of numerical datasets, it is common to calculate the similarity or dissimilarity between two feature vectors $x_1 = (x_{1j}, \ldots, x_{1p})$ and $x_2 = (x_{2j}, \ldots, x_{2p})$ using a square distance measure. Indeed, it is natural to use the Euclidean metric (distance) $\rho_E$ (or $L_2$ metric),

$$
\rho_E(x_1, x_2) = ||x_1 - x_2||_2 = \left( \sum_{j=1}^{p} (x_{1j} - x_{2j})^2 \right)^{1/2},
$$

as a measure for continuous numerical features because this metric is in everyday use. For example, the most popular algorithm for clustering numerical datasets is the $k$-means algorithm that uses the Euclidean distance.

The generalization of the $k$-means algorithm by Huang (1997b) called $k$-prototypes algorithm is also based on the Euclidean distance. The $k$-prototypes algorithm was introduced to cluster large datasets with mixed numerical and categorical values. It should be noted that both the $k$-means and $k$-prototypes algorithms have a disadvantage, that the process often converges not to a global minimum but to a local minimum. Hence, to avoid this premature convergence one has to modify these algorithms. Recently, an approach to optimization problems called the bees algorithm (BA) has been presented (e.g. Pham & Castellani 2009). The BA combines neighbourhood search with random search. The randomness of the search provides flexibility in the search and, hence, the BA often gives results that are quite close to a global minimum. In this paper, a new tool for clustering mixed datasets is introduced that combines the advantages of both the $k$-prototypes algorithm and the BA.
The paper is organized as follows.
In §2, formal descriptions of both the k-prototypes algorithm and the BA are given.
In §3, random search with the k-prototypes algorithm (RANKPRO) is presented.
In §4, RANKPRO is applied to several datasets. The effectiveness of the RANKPRO and the k-prototypes clustering algorithms is compared and the advantages of the RANKPRO are shown.

2. Preliminaries

(a) The k-means and k-prototypes algorithms

The k-means algorithm (MacQueen 1967) was introduced to cluster numerical datasets. It minimizes the objective function \( J \) for hard (non-fuzzy) \( k \)-partition of a dataset into \( k \) clusters (Bezdek 1980):

\[
J = \sum_{m=1}^{k} \sum_{i=1}^{N} u_{im} \rho_E^2(X_i, Q_m),
\]

\[
u_{im} \in \{0, 1\}, \quad 1 \leq i \leq N, \quad 1 \leq m \leq k, \]

\[
\sum_{m=1}^{k} u_{im} = 1, \quad \forall \ i \quad \text{and} \quad \sum_{i=1}^{N} u_{im} > 0 \quad \forall \ m.
\]

Here \( u_{im} \) is an element of the partition matrix. The condition \( u_{im} = 1 \) means that the record \( X_i \) is assigned to cluster \( m \) with prototype (centre) \( Q_m \). Since \( \rho_E \) defined by equation (1.1) is employed in this paper for clustering of numerical data, \( J \) is the within-group sum of squared errors objective function.

The implementation of k-means may have various forms, in particular its pseudo code can be written as

Step 1. Select randomly \( k \) initial prototypes \( Q_1, \ldots, Q_k \), one for each cluster.

Step 2. For each record \( X_i \) calculates the distances from the record to the prototypes of clusters; finds the nearest prototype \( Q_m \) to the record according to the metric \( \rho_E \) defined by equation (1.1), and allocates the record \( X_i \) to the cluster \( C_m \) with this prototype.

Step 3. For each cluster \( C_m \) find a new prototype \( Q'_m \) so that the sum of square distances \( \sum_{X_i \in C_m} \rho_E^2(X_i, Q'_m) \) is minimum.

Step 4. If prototypes \( Q_1, \ldots, Q_k \) and \( Q'_1, \ldots, Q'_k \) are not the same then take the latest as new prototypes and go to step 2, otherwise stop the procedure.

It is known (e.g. Stevens 1946) that to deal statistically with categorical data, one needs to deal with modes of the data instead of means or medians used to deal with numerical variables. In statistics, the mode is that value which occurs most often or, in other words, has the greatest probability of occurring (e.g. Spiegel 1975). Huang introduced two extensions of the k-means algorithm, namely the algorithms called k-modes (Huang 1997a) and k-prototypes (Huang
The former algorithm was targeted to deal with categorical attributes, while the latter was introduced to cluster large datasets with mixed numerical and categorical values. Huang replaced means of clusters used in the \( k \)-means algorithm by modes, and used a frequency-based method to update modes in the clustering process to minimize the clustering objective function (Huang 1998). In the \( k \)-prototypes algorithm he defined a ‘dissimilarity measure’ that takes into account both numerical and categorical attributes. He applied a metric \( \rho_H \), where \( \rho_H^2 \) is the sum of the square of the numerical metric (1.1) and a weighted categorical metric \( \rho_{\text{cat}} \):

\[
\rho_H^2 = \rho_E^2 + \gamma \rho_{\text{cat}}. \tag{2.2}
\]

The categorical metric \( \rho_{\text{cat}} \) is defined by the number of mismatches of categories between two objects and the weight \( \gamma \) is introduced for the categorical metric to balance the two parts of the sum and to avoid favouring either type of attribute.

The pseudo codes of the \( k \)-modes and \( k \)-prototypes algorithms are very similar to the pseudo code of the \( k \)-means algorithm. The difference between the algorithms is mainly that different dissimilarity measures are used. The \( k \)-means algorithm has a great advantage that it converges rapidly to a local minimum and at each iteration it improves the solution. Consequently, the \( k \)-prototypes algorithm has the same advantage. However, application of both \( k \)-means and \( k \)-prototypes algorithms to datasets also has a disadvantage, that the process normally demonstrates premature convergence, i.e. it converges not to the global minimum but to a local minimum. Hence, one needs to run the procedure many times to reach the global minimum. To increase the effectiveness of the procedure, one has to modify these algorithms.

(b) The bees algorithm

It is known that optimization techniques use various methods, strategies and algorithms. In particular, they include genetic algorithms (GAs) that mimic heuristically biological evolution, namely the principle of the survival of the fittest individual in the process of selection. Similarly to GAs, Swarm intelligence (SI) is a type of artificial intelligence that mimics the collective behaviour of animals. For example, SI includes ant colony optimization (Dorigo et al. 1996) and the BA (Pham & Castellani 2009). The latter is a relatively new technique introduced to mimic nature’s evolutionary principles that drive the search of bees towards an optimal solution.

Some features of the BA are similar to features of Hill climbing (HC), but there is a key difference that BA uses a population of solutions for every iteration instead of a single solution. As a population of solutions is processed in an iteration, the outcome of each iteration is also a population of solutions. Individual solutions in a population are viewed as ‘points’ in the search space.

In application to problems of optimization, a bee means a point in the domain (the search space) of the objective function, while the fitness of the bee means the value of the objective function at this point. It was shown (Pham & Castellani 2009) that using the BA for some optimization problems is more effective than using GA-based techniques (Goldberg 1989).

The BA starts by initializing a set of the following parameters: the number of scout bees \( (n) \) that define the total number of points; the number of best points \( (m) \) out of the total number of \( n \) points; the number of elite points \( (e) \); the size of
neighbourhood \( (d_{\text{ ngh}}) \) around any of the best points; the number of recruited bees \( (r_e) \) within the neighbourhood for the elite points; the number of recruited bees \( (r_g) \) around other selected \( (g = m - e) \) points; and stopping criteria. According to the pseudo code for the BA \( (\text{Pham & Castellani 2009}) \), \( n \) bees are placed in the search space randomly as scout bees. Every bee in the problem space evaluates the fitness of its point in step 2. Subsequently, in step 4, elite bees that have better fitness are selected and saved for the next population. In step 5, good points for neighbourhood search are selected. In step 6, the bees search around these points within the neighbourhood boundaries and their individual fitness is evaluated. More bees will be recruited around elite points and fewer bees will be recruited around the remaining selected points.

The pseudo code for the BA can be written as the following \( (\text{Pham & Castellani 2009}) \).

Step 1. Initialize population with random solutions \( (n \) points discovered by \( n \) scout bees).  
Step 2. Evaluate fitness of the population \( (\text{for } n \text{ points}) \).  
Step 3. While (stopping criteria not met)  
   Step 4. Select \( e \) elite points.  
   Step 5. Select \( g \) good points for neighbourhood search \( (e + g = m) \).  
   Step 6. Recruit \( r_e \) bees around each of selected elite points and \( r_g \) bees around each of selected good points.  
   Step 7. Evaluate fitness of solutions for all of these recruited bees and select the best bee for each neighbourhood.  
   Step 8. Assign remaining \( (n - m) \) bees to search randomly and evaluate the fitness of each of the discovered points.  
Step 9. Form new population.  
Step 10. End While.

Some features of the BA are similar to features of HC, local beam search (LBS) and stochastic beam search (SBS) strategies that are used to solve optimization problems. The description of these techniques can be found elsewhere \( (\text{Russell & Norvig 2003}) \). Indeed, similarly to these methods, the BA starts with a random selection of solutions and then improves the solutions iteratively. However, contrary to the HC and LBS algorithms, it has the ability to converge to a global extremum in a multi-extremum problem, owing to random exploring the search space at each iteration. The BA and SBS algorithms use different procedures for selection of the fittest solutions.

The GA search balances two objectives: using the best solutions and exploring the search space \( (\text{Michalewicz 1996}) \). The BA also tries to balance these objectives. However, to explore the search space it uses random search instead of the crossover and mutation operations used by GAs. The BA works with the most promising and elite solutions. In application to GAs, the term elitism was first introduced by \( \text{De Jong (1975)} \) \( (\text{see also Mitchell 1996}) \) in order to force a GA to retain some of the best individuals at each generation. The introduction of elite individuals enables the algorithm under consideration to preserve the best solutions. Otherwise they could be lost or destroyed by crossover or mutation. This technique has been employed by the BA that works with elite and good points, continuing to look for other possible extrema points while keeping the number of testing points constant.  

The BA also has some common features with controlled random search (CRS) algorithms introduced by Price (1978) (see also Kaelo & Ali 2006 for a review of recent modifications to CRS). Indeed, both CRS and BA employ initial points that are uniformly distributed over the search space and, unlike gradient-based methods, they calculate only the value of the function itself and do not use any property of the function. However, CRS and BA algorithms have also a notable difference. If in the CRS, the region of testing points is gradually contracted by replacing the current worst point with a better point (the trial point) that is chosen by a kind of interpolation, the BA always explores the whole search space.

It has been suggested to use the BA not only for optimization problems but also for clustering (Pham et al. 2007) when a bee represents a potential clustering solution as a set of \( k \) cluster centres. However, again the BA suggests using a random search in the neighbourhoods of selected solutions (partitions) and hence, the local improvement of promising solutions by the BA may be time-consuming.

Additionally, there is a faster way to improve the solutions, namely employment of the \( k \)-prototypes algorithm that converges rapidly to a local optimal solution. In this, the algorithm is similar to gradient-like methods of search of local extrema (Pham & Jin 1995; Wen et al. 2003). It is, therefore, proposed to use the \( k \)-prototypes algorithm to improve promising solutions. Thus, the new RANKPRO algorithm that includes the preservation of elite solutions and random exploring of the search space along with employment of the \( k \)-prototypes algorithm for improvement of solutions is proposed.

3. Description of random search with \( k \)-prototypes algorithm

To deal with mixed datasets, one needs to use appropriate metrics. Hence, the normalization of metrics for mixed datasets is discussed first and then the RANKPRO algorithm is described in detail.

(a) Normalization of metrics for mixed datasets

It is mentioned above that in clustering analysis of numerical datasets the Euclidean metric (1.1) is commonly used. For datasets with categorical attributes, it is possible to introduce different metrics (e.g. Ralambondrainy 1995; Huang 1998). One of the most used variants of metrics is the distance between two categorical feature vectors \( y_1 = (y_{11}, \ldots, y_{1l}) \) and \( y_2 = (y_{21}, \ldots, y_{2l}) \) (e.g. Huang 1998). It is defined as

\[
\rho_{\text{cat}}(y_1, y_2) = \omega(y_{11}, y_{21}) + \cdots + \omega(y_{1l}, y_{2l}),
\]

(3.1)

where

\[
\omega(y_{1j}, y_{2j}) = \begin{cases} 
0 & \text{for } y_{1j} = y_{2j} \\
1 & \text{for } y_{1j} \neq y_{2j}
\end{cases}
\]

and the square of the metric (3.1) is

\[
\rho_{\text{cat}}^2(y_1, y_2) = \omega^2(y_{11}, y_{21}) + \cdots + \omega^2(y_{1l}, y_{2l}).
\]

(3.2)
Combining $\rho_E$ and $\rho_{\text{cat}}$ for mixed data, one obtains the square distance between two mixed feature vectors $(x_1, y_1)$ and $(x_2, y_2)$

$$\rho^2((x_1, y_1), (x_2, y_2)) = \rho_E^2(x_1, x_2) + \rho_{\text{cat}}^2(y_1, y_2), \quad (3.3)$$

where $\rho_E^2(x_1, x_2)$ is defined by equation (1.1) and $\rho_{\text{cat}}^2(y_1, y_2)$ is defined by equation (3.2).

As mentioned above, the use of metric (2.2) may encounter some obstacles in practical realization because it is not very clear how to find an appropriate weight $\gamma$ for the metric to balance the two parts of the sum and avoid favouring either type of attribute. A direct application of the geometric measures for attributes with large ranges will implicitly assign bigger contributions to the metrics than those for attributes with small ranges. In addition, the attributes should be dimensionless. Indeed, the numerical values of the ranges of dimensional attributes depend on the units of measurements and, therefore, the choice of the units of measurements may greatly affect the results of clustering. Hence, if all attributes are equally important to measure the similarity between feature vectors then one should use a normalization procedure. Here the normalization procedure described in detail in Pham et al. (2006) is employed.

It is natural to assume that the average contribution of the $j$-th feature component to the total measure is equal to its mean and, therefore, the goal of a normalization procedure is the equalization of the attribute contributions. Each record in a database is treated as a random sample of the population under consideration, i.e. one has a database of $N$ observations (samples) and each sample (record) is a realization of possible values of the feature vector $A$ (Pham et al. 2006; Suarez-Alvarez 2010). Using a statistical approach to both numerical and categorical attributes, one can normalize the feature vectors for mixed datasets, and, hence, one can achieve a balance between all attributes disregarding their type. Here, the procedure is briefly described for completeness.

For statistical treatment of feature vectors, one needs to know the probability distributions of their attributes. For a numerical attribute $A^n_j$, the probability distribution identifies the probability of the attribute value falling within a particular interval in the range of possible values. For a categorical attribute $A^c_j$, the probability distribution identifies the probability of certain states occurring. Let $X_{1j}$ and $X_{2j}$ be independent random variables whose values are distributed in accordance with the distribution of the $j$-th attribute. To obtain a new normalized metric for the Euclidean metric, one should calculate the mean contribution of each $j$-th attribute to the metric $E|X_{1j} - X_{2j}|^2$ (here $E$ means the expectation of a variable) and divide the attribute in all records by this mean (if the mean is equal to zero then this attribute should be removed from the feature vector). Using the classic theorem on variance (e.g. Spiegel 1975), one obtains

$$E|X_{1j} - X_{2j}|^2 = EX_{1j}^2 - 2EX_{1j}EX_{2j} + EX_{2j}^2 = 2(EX_{1j}^2 - (EX_{1j})^2) = 2\sigma_j^2.$$ 

Since this result is valid for any distribution of the $j$-th attribute, the normalized Euclidean metric is introduced as

$$\rho_E^*(x_1, x_2) = \left(\sum_{j=1}^{p} \frac{(x_{1j} - x_{2j})^2}{2\sigma_j^2}\right)^{1/2}, \quad (3.4)$$
where \( \sigma_j \) is the standard deviation of the \( j \)-th attribute. The expression (3.4) can be rewritten as

\[
\rho^*_E(x_1, x_2) = \left( \sum_{j=1}^{p} \alpha_j |x_{1j} - x_{2j}|^2 \right)^{1/2},
\]

where \( \alpha_j = 1/E|X_{1j} - X_{2j}|^2 \). To estimate \( \sigma_j^2 \) in equation (3.4), it is possible to use the following unbiased estimator of the sample variance

\[
s_j^2 = \frac{1}{N - 1} \sum_{r=1}^{N} (x_{rj} - \bar{x}_j)^2,
\]

where \( \bar{x}_j = \frac{1}{N} \sum_{r=1}^{N} x_{rj} \) is the sample mean for the \( j \)-th attribute.

The same idea that has been applied to numerical features is applied here to the categorical features, namely we divide the contribution of each attribute to the distance measure by the contribution mean. Let \( Y_{1j} \) and \( Y_{2j} \) be independent random variables whose values are distributed in accordance with the distribution of the \( A^c_j \)-th attribute. If the attribute \( A^c_j \) can take \( q_j \) values \( \{y_1, y_2, \ldots, y_{q_j}\} \) and the probabilities \( \{p_{j1}, p_{j2}, \ldots, p_{jq_j}\} \) of these values are known, then

\[
E\omega^2(Y_{1j}, Y_{2j}) = E\omega(Y_{1j}, Y_{2j}) = \sum_{r,s=1 \atop r \neq s}^{q_j} p_{jr} p_{js} = \sum_{r,s=1}^{q_j} p_{jr} p_{js} - (p_{j1}^2 + \cdots + p_{jq_j}^2)
\]

or

\[
E\omega^2(Y_{1j}, Y_{2j}) = (p_{j1} + \cdots + p_{jq_j})^2 - (p_{j1}^2 + \cdots + p_{jq_j}^2) = 1 - (p_{j1}^2 + \cdots + p_{jq_j}^2).
\]

Hence, if \( \beta_j \) is defined as \( \beta_j = 1/E\omega^2(Y_{1j}, Y_{2j}) \) then it follows from the above equality

\[
\beta_j = \frac{1}{1 - (p_{j1}^2 + \cdots + p_{jq_j}^2)}.
\]

Thus, the normalization of the Euclidean mixed metric is

\[
\rho^*((x_1, y_1), (x_2, y_2)) = \left[ \sum_{j=1}^{p} \alpha_j |x_{1j} - x_{2j}|^2 + \sum_{j=1}^{l} \beta_j \omega^2(y_{1j}, y_{2j}) \right]^{1/2},
\]

where \( \alpha_j = 1/E|X_{1j} - X_{2j}|^2 \) and \( \beta_j = 1/E\omega^2(Y_{1j}, Y_{2j}) \).

For any distribution of the \( j \)-th attribute \( \alpha_j \) can be calculated using the following estimation

\[
\hat{E}|X_{1j} - X_{2j}|^2 = \frac{2}{N(N - 1)} \sum_{1 \leq r < s \leq N} |x_{rj} - x_{sj}|^2,
\]

which is the same as the one obtained with the above-mentioned unbiased estimator of the sample variance (Suarez-Alvarez 2010), and to estimate $E\omega(Y_{1j}, Y_{2j})$ one can use the sample mean

$$
\hat{E}\omega^2(Y_{1j}, Y_{2j}) = \frac{1}{N^2} \sum_{r,s=1}^{N} \omega(y_{rj}, y_{sj}). \quad (3.9)
$$

The estimation (3.9) is a biased estimator of $E\omega^2(Y_{1j}, Y_{2j})$, hence for small databases it is better to use the following estimation

$$
\hat{E}\omega^2(Y_{1j}, Y_{2j}) = \frac{2}{N(N-1)} \sum_{1 \leq r < s \leq N} \omega(y_{rj}, y_{sj}). \quad (3.10)
$$

It is possible to show that equation (3.10) is an unbiased and consistent estimator (Suarez-Alvarez 2010).

(b) Pseudo code of the random search with k-prototypes algorithm

If there are a dataset and a set of $k$-prototypes $S = \{Q_1, \ldots, Q_k\}$ then a record $A_i$ may be allocated to cluster $C_m$ whose prototype $Q_m$ is the nearest to the record according to the normalized mixed metric (3.7). Hence, a set of prototypes $S$ gives a partition of the dataset to $k$ clusters $\{C_1, \ldots, C_k\}$. In this paper, a set of prototypes $S = \{Q_1, \ldots, Q_k\}$ is called an approximate solution to the clustering problem if it gives a partition to $k$ non-empty clusters.

The clustering algorithm has to minimize the objective function

$$
J(S) = \sum_{m=1}^{k} \sum_{i=1}^{p+l} u_{im} [\rho^*(A_i, Q_m)]^2, \quad \begin{cases} 
\sum_{m=1}^{k} u_{im} = 1, & \forall i \quad \text{and} \quad \sum_{i=1}^{p+l} u_{im} > 0 \forall m.
\end{cases} \quad (3.11)
$$

The condition $u_{im} = 1$ for an element of the partition matrix means as above in equation (2.1) that the record $A_i$ is assigned to cluster $C_m$ with prototype $Q_m$. Equation (3.11) can be written as

$$
J(S) = \sum_{m=1}^{k} \sum_{A_i \in C_m} [\rho^*(A_i, Q_m)]^2. \quad (3.12)
$$

To start the RANKPRO, one has to assign a set of parameters, namely the number ($n$) of the approximate solutions to the clustering problem that are considered at each step, the number ($e$, $1 \leq e < n$) of elite solutions that are kept to the next step of the algorithm, the number ($r = n - e$) of solutions that are used for random search, and the number ($n_{iter}$) of iterations of the $k$-prototypes algorithm applied to each solution to improve it. As stopping criterion, one can take either the approach of the process time to the given maximum time $t_{max}$ or the approach of the number of process iterations to the given maximum number of iterations.
The pseudo code for RANKPRO is described as the following.

Step 1. Initialization. Randomly select \( n \) solutions \( S_i \), \( 1 \leq i \leq n \).
Step 2. While (the stopping criterion is not yet met) consider the selected \( n \) solutions.
Step 3. Apply \( n_{\text{iter}} \) of iterations of the \( k \)-prototypes algorithm to each solution to improve it. The application of the algorithm to the solution has to be stopped if it becomes stable, which means that it has reached a local minimum.
Step 4. For each \( S_i \), calculate the objective function \( J(S_i) \).
Step 5. Select \( e \) solutions with the best values of the objective function for further study. At the next step, the remaining solutions \( r = n - e \) are removed and replaced by randomly selected solutions.
Step 6. End While.

The solution of the clustering problem is the best solution \( S \) obtained at the last step.

One can see that the RANKPRO, like the BA, uses a population of solutions for every iteration instead of a single solution. The BA uses a random search in the neighbourhood of selected solutions and, therefore, the local improvement of promising solutions by the BA may be time-consuming. The employment of the \( k \)-prototypes algorithm that converges very fast to a local optimal solution is a faster way to improve the solutions. In this the algorithm is similar to gradient-like methods for search of local extrema.

We can see that at every iteration new random solutions are added and therefore the RANKPRO explores the search space effectively.

It is important to note that in order to save time in the process of improving the solutions, the RANKPRO algorithm is applied to the solutions not until its convergence but to a limit of \( n_{\text{iter}} \) times. This number can be estimated by prior study of a specific dataset.

4. Application to datasets

(a) Comparing the effectiveness of the clustering algorithms

The comparison of the effectiveness of the clustering algorithms is not an easy task. Goldberg & Deb (1991) review and compare several selection schemes used in GAs. They note that many claims and counterclaiims have been presented regarding the superiority of one selection scheme over another. However, most of these claims are based on limited (and uncontrolled) simulation experience, while surprisingly little analysis was performed to understand relative expected fitness ratios, convergence times or the functional forms of selective convergence. A similar situation may be encountered in the area of comparison of clustering algorithms. Hence, the following procedure for comparing the effectiveness of the clustering algorithms has been suggested.

The above methods (the RANKPRO and the \( k \)-prototypes algorithms) are applied to several benchmark datasets from the UC Irvine repository (Asuncion & Newman 2007) after normalization of the data in accordance with the above-described method. The effectiveness of the clustering algorithms is compared
with one another for different initial parameters and datasets. The scheme described below compares the average minimum values of the objective function obtained during the runs of the algorithms, applied to the same dataset over the same time.

Let the RANKPRO algorithm with a specified set of initial parameters be applied to a specified dataset during the given time $t_{\text{exec}}$ and $J(t_{\text{exec}})$ be the minimum value of the objective function (3.12) obtained by the execution of the algorithm during this time. If the algorithm is run a given number $n_r$ of simulations then one obtains a set of $J^{(m)}(t_{\text{exec}})$ values of the objective function ($m = 1, \ldots, n_r$). The average value $J_{\text{av}}(t_{\text{exec}}) = \frac{1}{n_r} \sum_{m=1}^{n_r} J^{(m)}(t_{\text{exec}})$ is a characteristic of the effectiveness of the algorithm with the specified set of initial parameters during $t_{\text{exec}}$. The less is $J_{\text{av}}(t_{\text{exec}})$ the greater is the effectiveness of the algorithm. However, if one sets the value $t_{\text{exec}}$ large then all algorithms may give the same value of the objective function, namely its global minimum value. Hence, it is pointless to compare the algorithms for large values of $t_{\text{exec}}$. Evidently, $J_{\text{av}}(t_{\text{exec}})$ depends also on the values of $n_r$. However, if $n_r$ is large enough then variation of the $J_{\text{av}}(t_{\text{exec}})$ values will be small.

To compare the effectiveness of the RANKPRO and the $k$-prototypes algorithms, the latter is also applied to the same dataset. If the $k$-prototypes algorithm converges before the process time reaches $t_{\text{exec}}$ then the algorithm is run again and again while the allowed process time is not expired. Each time after convergence of the $k$-prototypes process, the minimum value of the objective function is recorded. The average of the minimum values of the objective function obtained during these runs is taken as $J_{\text{av}}(t_{\text{exec}})$. 

(b) Adult dataset

The ADULT dataset, also known as the Census Income dataset, has 48 842 records and 45 222 records without missing values with 14 attributes and one class attribute. Each record has eight categorical attributes plus a class attribute, while the rest of the attributes are numerical. This is a standard dataset that has been studied a number of times to test clustering algorithms (e.g. Huang 1998). For the ADULT dataset, figure 1 shows the graphs of the average values of $J_{\text{av}}(S)$ versus $t_{\text{exec}}$ in seconds. The number of simulations is $n_r = 100$ in all cases.

One can see from figure 1a that if the parameters $n$, $e$ and $r$ are fixed ($n = 8$, $e = 1$ and $r = 7$) and the parameter $n_{\text{iter}}$ is varied, then the best performance is for $n_{\text{iter}} = 5$. In all cases, the RANKPRO algorithm gave smaller values for the average of the objective function $J_{\text{av}}(t_{\text{exec}})$, i.e. the RANKPRO algorithm is more efficient than the $k$-prototypes algorithm.

Figure 1c shows that it is more efficient to keep only one elite solution ($e = 1$) than two elite solutions ($e = 2$), irrespective of the number of iterations ($n_{\text{iter}} = 3$ or $n_{\text{iter}} = 5$) of the $k$-prototypes algorithm applied to improve the solutions.

Figure 1d confirms the above conclusions. If the parameters $n$, $e$ and $r$ are fixed ($n = 8$, $e = 2$ and $r = 6$) and the parameter $n_{\text{iter}}$ is varied, then the best performance is for $n_{\text{iter}} = 5$. The RANKPRO algorithm is more efficient than the $k$-prototypes algorithm. However, the performance of the former algorithm is worse than its performance in the case $e = 1$, i.e. it is more efficient to keep only one elite solution rather than two.
Figure 1. The average values of $J_{av}(S)$ versus $t_{exec}$ for the ADULT dataset. Comparison of the $k$-prototypes algorithm and the RANKPRO algorithm for varying values of parameters: (a) the values of $n_{iter}$ are different, $n_{iter} = 3, 5, 7, 9$, while other parameters are constant: $n = 8$, $e = 1$ and $r = 7$; (b) values of $n$ and corresponding values of $r$ are different, $n = 6 \div 10$ and $r = 5 \div 9$, $(r = n - e)$, while the parameters $n_{iter}$ and $e$ are fixed, $n_{iter} = 5$ and $e = 1$; (c) the following values of parameters: (i) $n_{iter} = 3$, $e = 1$, $r = 7$ and $n = 8$, (ii) $n_{iter} = 5$, $e = 1$, $r = 7$ and $n = 8$, (iii) $n_{iter} = 3$, $e = 2$, $r = 6$ and $n = 8$, and (iv) $n_{iter} = 5$, $e = 2$, $r = 6$ and $n = 8$; (d) different values of $n_{iter}$. $n_{iter} = 3, 5, 7$ and $9$, while other parameters are constant: $n = 8$, $e = 2$ and $r = 6$.

One can see from figure 1b that if the number of randomly chosen solutions $n$ is varied, while other parameters $e$ and $n_{iter}$ are fixed ($n_{iter} = 5$ and $e = 1$), the best performance is for $n = 8$ ($r = 7$). In all cases, the RANKPRO algorithm is more efficient than the $k$-prototypes algorithm.

(c) Shuttle dataset

The full Shuttle dataset, also known as Statlog (Shuttle) dataset, has 58 000 records with nine numerical attributes and one class attribute. This is a standard dataset that has been used a number of times to test the clustering and classification algorithms (e.g. Garcke et al. 2001). For this dataset, figure 2 shows the graphs of average values of $J_{av}(S)$ versus $t_{exec}$ in seconds. The number of simulations is $n_r = 100$ in all cases.

The RANKPRO algorithm is more efficient than the prototypes algorithm. One can see from figure 2a that if the parameters \( n, e \) and \( r \) are fixed \((n = 8, e = 1 \text{ and } r = 7)\) and the parameter \( n_{iter} \) is varied, then the best performance is for \( n_{iter} = 5 \). In all cases, the RANKPRO algorithm gave smaller values for the average of the objective function \( J_{av}(t_{exec}) \), i.e. the RANKPRO algorithm is more efficient than the \( k \)-prototypes algorithm.

Figure 2b shows that for the Shuttle dataset, contrary to the ADULT dataset, there is no evident advantage of keeping just one elite solution \((e = 1)\) or two elite solutions \((e = 2)\). In all cases, the RANKPRO algorithm is more efficient than the \( k \)-prototypes algorithm.

Figure 2c shows that if the parameters \( n, e \) and \( r \) are fixed \((n = 8, e = 2 \text{ and } r = 6)\) and the parameter \( n_{iter} \) is varied then the best performance is for \( n_{iter} = 3 \). The RANKPRO algorithm is more efficient than the \( k \)-prototypes algorithm.

One can see from figure 2d that if the number of randomly chosen solutions \( n \) is varied, while other parameters \( e \) and \( n_{iter} \) are fixed \((n_{iter} = 5 \text{ and } e = 1)\), there is no evident advantage for a specific number \( n \). In all cases, the RANKPRO algorithm is more efficient than the \( k \)-prototypes algorithm.

Figure 3. The average values of $J_{av}(S)$ versus $t_{exec}$ for the Covertype dataset. Comparison of the $k$-prototypes algorithm and the RANKPRO algorithm with varying values of parameters: (a) the values of $n_{iter}$ are different, $n_{iter} = 6, 8, 10, 12$, while other parameters are constant: $n = 9, e = 1$ and $r = 8$; (b) the following values of parameters: (i) $n_{iter} = 6, e = 1, r = 8$ and $n = 9$, (ii) $n_{iter} = 8, e = 1, r = 8$ and $n = 9$, (iii) $n_{iter} = 6, e = 2, r = 7$ and $n = 9$, and (iv) $n_{iter} = 8, e = 2, r = 7$ and $n = 9$; (c) the values of $n_{iter}$ are different, $n_{iter} = 6, 8, 10, 12$, while other parameters are constant: $n = 9, e = 2$ and $r = 7$.

Figure 4. The average values of $J_{av}(S)$ versus $t_{exec}$ for the Connect-4 dataset. Comparison of the $k$-prototypes algorithm and the RANKPRO algorithm with different values of $n_{iter}$, $n_{iter} = 3, 5, 7, 9$, while other parameters are constant: $n = 8, e = 1$ and $r = 7$. 

**RANKPRO clustering algorithm**

The full Covertype dataset has 581,012 records with 44 categorical attributes and 10 numerical. However, only the first 100,000 records were used. This is a dataset that predicts forest cover type from cartographic variables. It has been used several times to test classification algorithms (e.g. Blackard & Dean 1999). For this dataset, figure 3 shows the graphs of average values of \( J_{av}(S) \) versus \( t_{exec} \) in seconds. Since this dataset is large, the values of \( t_{exec} \) should be greater than the values for other datasets, consequently we have chosen the number of simulations \( n_r = 10 \) instead of \( n_r = 100 \) in this case.

One can see from figure 3a that if parameters \( n, e \) and \( r \) are fixed (\( n = 9, e = 1 \) and \( r = 8 \)) and parameter \( n_{iter} \) is varied, then it is difficult to give any preference to a specific value of \( n_{iter} \). However, the RANKPRO algorithm gave smaller values for the average of the objective function \( J_{av}(t_{exec}) \) in all cases.

Figure 3b shows that for the Covertype dataset, there is an advantage in keeping just one elite solution (\( e = 1 \)) rather than two elite solutions (\( e = 2 \)). In all cases, the RANKPRO algorithm is more efficient than the \( k \)-prototypes algorithm.

Figure 3c shows that in the case of two elite solutions, there is an advantage in keeping \( n_{iter} = 6 \). Again the RANKPRO algorithm is more efficient than the \( k \)-prototypes algorithm.

The full Connect-4 dataset contains all positions in the game of connect-4. It has 67,557 records with 42 categorical attributes, each attribute corresponding to one connect-4 square. All 67,557 records have been used.

For this dataset, figure 4 shows the graphs of the average values of \( J_{av}(S) \) versus \( t_{exec} \) in seconds. The number of simulations is \( n_r = 100 \).

One can see in figure 4 that if the parameters \( n, e \) and \( r \) are fixed (\( n = 8, e = 1 \) and \( r = 7 \)) and the parameter \( n_{iter} \) is varied, then there is no evident advantage for a specific number \( n_{iter} \).

For \( t_{exec} \geq 2.5 \) s, the RANKPRO algorithm is less or equally efficient than the \( k \)-prototypes algorithm. The explanation could be the following. For a specified dataset, it is assumed that \( n_{iter} \) prescribed for the RANKPRO algorithm is less than the average number of iterations that the \( k \)-prototypes algorithm needs to converge. Hence, it is deduced that the RANKPRO algorithm does not spend much time exploring current solutions. For the Connect-4 dataset, the \( k \)-prototypes algorithm converges very fast in comparison with the other datasets under consideration. Indeed, the average number of iterations that the \( k \)-prototypes algorithm needs to converge is 10.07, 26.29, 24.88 and 1.53 for the Adult, Shuttle, Covertype and Connect-4 datasets, respectively. One can see that in the case of Connect-4 dataset, the \( k \)-prototypes algorithm converges to a local minimum very fast and it can be applied to the data many times during a prescribed \( t_{exec} \). Hence, the above assumption that the RANKPRO algorithm spends less time than the \( k \)-prototypes algorithm to explore local minima is not satisfied and the RANKPRO algorithm loses its advantage. In this case, the algorithms have approximately equal effectiveness.

*Proc. R. Soc. A (2011)*
A new clustering algorithm called RANKPRO, Random Search with $k$-prototypes algorithm, has been presented. The algorithm combines the advantages of the BA and $k$-prototypes algorithms. The RANKPRO algorithm has been applied to various datasets, including datasets with mixed numerical and categorical values. It balances two objectives: to explore the search space effectively owing to random selection of new solutions, and to improve promising solutions fast owing to employment of the $k$-prototypes algorithm.

To estimate the distances between records of the data, normalized metrics are used. Since a mixed database is treated as a random sample of an object under consideration, the normalized metrics have been obtained using a statistical approach. These normalized metrics are more general than the metric introduced in Huang (1997b) for mixed datasets.

It is expected that the new RANKPRO algorithm will have lower probability for premature convergence than the $k$-prototypes algorithm owing to the employment of random search. On the other hand, the application of several iterations of the $k$-prototypes algorithm for very fast improvement of the promising (elite) solutions resembles gradient-like methods. Hence, this is a more effective procedure than attempting to improve promising solutions by random neighbourhood search as is used in the BA.

The results obtained demonstrate the efficiency of the new algorithm. It is shown that in the majority of the datasets considered, when the $k$-prototypes algorithm needs many iterations for convergence, the RANKPRO algorithm is more efficient than the $k$-prototypes algorithm. However, if for a specific dataset, the $k$-prototypes algorithm converges to a local minimum very fast (in just a few iterations) then the algorithms have approximately equal effectiveness.

The authors have used above several benchmark datasets for testing the RANKPRO algorithm. They are grateful to the creators of the UC Irvine repository. Thanks are due to Prof. Fedor M. Borodich (Cardiff University) for his valuable comments on the paper. The authors would like to thank the European Commission for their support of this work as part of the Project ‘Multi-Role Shadow Robotic System for Independent Living’ (Grant Agreement EC-FP7-ICT 247772).

References


Suarez-Alvarez, M. M. 2010 Design and analysis of clustering algorithms for numerical, categorical and mixed data. PhD thesis, Cardiff University, Cardiff, UK.