Multi-objective clustering ensemble for gene expression data analysis

Katti Faceli a,⁎, Marcilio C.P. de Souto b, Daniel S.A. de Araújo b, André C.P.L.F. de Carvalho c

a Universidade Federal de São Carlos, Campus Sorocaba, Rodovia João Leite dos Santos, Km 110 – Parque Reserva Fazenda Imperial, 18052-780 – Sorocaba, SP, Brazil
b Universidade Federal do Rio Grande do Norte, Departamento de Informática e Matemática Aplicada, Campus Universitário, 59072-970 – Natal, RN, Brazil
c Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, Departamento de Ciências de Computação e Estatística, Caixa Postal 668, 13560-970 – São Carlos, SP, Brazil

⁎ Corresponding author.
E-mail addresses: katti@ufscar.br, kattifaceli@gmail.com (K. Faceli), marcial@dimap.ufrn.br (M.C.P. de Souto), danielsabino@ppgsc.ufrn.br (D.S.A. de Araújo), andre@icmc.usp.br (A.C.P.L.F. de Carvalho).

1. Introduction

The selection of the best clustering algorithm for a given data set is one of the main difficulties in cluster analysis [12,16,17,33]. In fact, there is a large number of clustering algorithms, each one looking for clusters according to a different cluster definition (or clustering criterion) [19,33]. These algorithms search for a homogeneous structure (all clusters conforming to the same cluster definition), whereas data can present a heterogeneous structure (each cluster conforming to a different cluster definition) [19]. A classical approach to address these problems is by using a clustering validation technique. However, most of these techniques are biased toward a given clustering criterion (e.g., cluster compactness) [16].

Alternatively, the problems of algorithm selection and data presenting clusters with heterogeneous structures can be addressed by using cluster ensemble and multi-objective clustering approaches [15,29]. Recently, we have introduced a multi-objective clustering ensemble algorithm (MOCLE, for short), which employs simultaneously concepts from both cluster ensemble and multi-objective clustering algorithms [11,12]. The idea is not only to minimize the intrinsic problems of cluster analysis, but also the limitations of the cluster ensemble and multi-objective clustering methods when they are used separately.

MOCLE is an algorithm that (1) can robustly deal with different structures (heterogeneous structures with different types of clusters at different refinement levels); (2) can find a concise set of high quality alternative structures (partitions); and (3) does not require expertise in cluster analysis. The basic steps of MOCLE are the following. First, a set of initial partitions is created by applying several different clustering algorithms to the data. Next, these partitions are combined and selected by applying a Pareto-based multi-objective genetic algorithm. The initial population and the optimization of several validation measures biased toward distinct clustering criteria form the multi-objective aspect of MOCLE. The ensemble component is given by a special crossover operator that finds the consensus between two partitions. The set of solutions returned by MOCLE is, at the same time, diverse, revealing the different structures present in the data, and concise enough to be analyzed by the domain experts.

In this paper, we apply MOCLE to gene expression microarray data. Cluster analysis of gene expression microarray data is of high interest in the field of functional genomics [9,23,25,27]. Among others, one of the main reasons for this is the need for molecular-based refinement of broadly defined biological classes, with implications to cancer diagnosis, prognosis and treatment. Although the choice of the clustering algorithm for the analysis of microarray data sets is a very important issue, as pointed out by D’haeseleer [9], there are in the literature few guidelines or standard procedures on how these data should be analyzed. The selection of algorithms is basically determined by the familiarity of biological experts with the algorithms rather than the features from the algorithms themselves and from the data [9]. For instance, the widespread use of hierarchical clustering algorithms
is mainly a consequence of its similarity to phylogenetic methods, which biologists are often acquainted to.

Thus, in the previous context, MOCLE can be a very useful tool in that, in order to obtain a good performance from clustering algorithms, it does not demand expertise from the user on cluster analysis or the data domain. As it will be seen later, in order to do so, MOCLE runs several conceptually different clustering algorithms with various parameter configurations, combines the partitions resulting from these algorithms, and selects the partitions with the best trade-offs for different validation measures.

The remaining of this paper presents the following structure. In Section 2 we give a brief background on cluster analysis, present a common problem of the traditional clustering algorithms and introduce two recent approaches that aim at solving this problem: cluster ensembles and multi-objective clustering. Next, we describe in detail our multi-objective clustering ensemble (Section 3). Section 4 contains a description of the clustering algorithms, index used in the evaluation of the techniques, as well as the transformations applied to the data. In Section 5, we describe the experiments performed, presenting the data sets used, the specific choices of the MOCLE’s components and the parameter configurations employed. In Section 6 we show and discuss the results obtained. Finally, in Section 7 we present some final remarks.

2. Background and related work

The aim of a clustering algorithm is to find a structure of clusters in the data, where objects belonging to each cluster share some relevant property regarding the data domain [16,17,33]. Such a structure can be a hierarchy or a (crisp, fuzzy, or probabilistic) partition. In this paper, we will focus on crisp partitions. Thus, hereafter, we use the term partition meaning a crisp partition. In a partition, each object should be assigned to only one cluster, and all objects must be assigned to a cluster. More formally, given a set of objects $X = \{x_1, x_2, \ldots, x_n\}$, a partition of $X$ in $K'$ clusters can be defined as $\pi^j = \{c_{1}^j, c_{2}^j, \ldots, c_{K'}^j\}$ with $K' < n$, such that [33]

1. $c_{j}^j \neq \emptyset$, $j = 1, \ldots, K'$,
2. $\bigcup_{j=1}^{K'} c_{j}^j = X$ and
3. $c_{j}^j \cap c_{l}^l = \emptyset$, $j, l = 1, \ldots, K'$ and $j \neq l$.

The main steps of cluster analysis consist of (1) preparation of the data; (2) application of a clustering algorithm to the data, given a proximity index; and (3) validation of the results to check if the clusters did not occur by chance or as an artifact of the algorithm. An additional step regarding the interpretation of the clusters can be included to establish their meaning. In order to guide the choices that have to be made during this process, expert knowledge on the clustering techniques, the data gathering process and the domain are required [16,17,33].

Traditional clustering algorithms, such as $k$-means (KM) [17], optimize only one clustering criterion (e.g., compactness of the clusters) and are often very effective in this purpose. However, they fail for data in accordance to a different criterion. One alternative to minimize this problem is to combine several clustering criteria to improve the quality of the final solution. Nowadays, there are two main approaches that address the use of multiple clustering criteria: cluster ensemble and the multi-objective clustering.

In this section, we describe two important cluster ensemble methods [13,29] and one multi-objective clustering algorithm [15]. The latter is closely related to this work. The main difference between the cluster ensemble and the multi-objective approaches is in the way they combine the clustering criteria. The former makes an a posteriori combination of a set of partitions obtained via different clustering criteria (heterogeneous ensemble). In contrast, the multi-objective approach simultaneously optimizes several clustering criteria. Both approaches are better suited to deal with different data conformations than the traditional clustering algorithms. However, each of them presents its own limitations.

2.1. Cluster ensembles

A typical cluster ensemble algorithm is composed of two steps: the generation of a diverse set of individual (base) partitions and their posterior combination to produce a consensus partition. This last step is accomplished via a consensus function [18]. There are several different ways to accomplish these steps [18,31]. Regarding the type of base partitions used as input for the consensus function, an ensemble can be homogeneous or heterogeneous. In a homogeneous ensemble, all base partitions are generated by the same clustering algorithm. In contrast, in a heterogeneous ensemble, the base partitions are generated by different clustering algorithms. In this paper, we are interested in heterogeneous ensembles. Two relevant examples of such ensembles can be seen in [13,29]. With respect to the consensus function, the main existing approaches are based in co-association, graph/hypergraph partitioning, mutual information or re-labeling [31].

The cluster ensemble method proposed in [29] is one of the most popular approaches for cluster ensemble [18]. In their work, the authors formalize the cluster ensemble problem as a combinatorial optimization problem in terms of shared mutual information. In order to tackle the combinatorial complexity of the problem, they propose three algorithms (consensus functions): CSPA (cluster-based similarity partitioning algorithm), HGPA (hyper-graph partitioning algorithm) and MCLA (meta-clustering algorithm). A supra-consensus function, based on the shared mutual information, can be applied to select the best partition among those produced by these three algorithms.

The CSPA algorithm starts with the construction of a new similarity matrix, according to the base partitions. The entries of this matrix denote the fraction of partitions in which two objects are assigned to the same cluster. This matrix is then employed to cluster the objects using any similarity-based clustering algorithm, producing the consensus partition.

In the HGPA algorithm, the combination is dealt with as a problem of partitioning a hypergraph. In this hypergraph, the clusters of the base partitions are represented as hyperedges. The hypergraph is partitioned by cutting a minimal number of hyperedges.

The MCLA algorithm considers the combination of partitions as a problem of finding the correspondence among the clusters of the base partitions. First, a meta-graph is constructed, where each cluster of the base partitions is a vertex. The edge weights are proportional to the similarity between the vertices. There are no edges between vertices from the same partition. Next, the meta-graph is partitioned. The clusters that were assigned to the same group (meta-cluster) are considered correspondents. The objects are then assigned to the meta-clusters that they are more strongly associated with, generating the consensus partition.

Another cluster ensemble method, based on graph partitioning, is the hybrid bipartite graph formulation (HBGF) [13]. In this method, first, a bipartite graph is constructed using the set of base partitions, modeling their objects and clusters simultaneously as vertices. Next, the graph is partitioned by a traditional graph
The goal of these ensemble methods is to find a consensus partition that agrees (resembles), as much as possible, with all base partitions used as inputs. However, this aim can lead to two main problems. First, a large number of base partitions of poor quality can also result into a poor consensus partition, even if among the base partitions there are few of them of excellent quality. The second problem regards the difficulty in the generation of a heterogeneous structure (a partition with different types of clusters). For instance, even if each of the base partitions contains one cluster of high quality, according to one of the cluster definitions, this cluster will not appear in the final consensus partition, because its information will be “overwritten” by the poor quality clusters from other partitions. Thus, although considering multiple criteria in the individual partitions, the consensus function does not consider different clustering criteria for different regions of the feature space [19].

Other disadvantage found in cluster ensemble methods, in general, is that, like the traditional clustering methods, they produce only a single partition as final result. This limits the amount of information that can be extracted from the data. Besides, as in most of the traditional clustering algorithms, these methods rely on the fine adjustments of their parameters to obtain a high quality consensus partition. In several cases, the user must supply the number of clusters in advance. However, for real data sets, this number is usually not known a priori. Because of this complex parameter setting, the validation step for these methods requires a good deal of expertise, similar to what happens with traditional clustering algorithms.

2.2. Multi-objective clustering

In contrast to cluster ensemble algorithms, multi-objective clustering optimizes two or more clustering criteria simultaneously, producing a set of partitions as output. These partitions represent different trade-offs for these criteria. Such an approach overcomes the limitations of obtaining only one solution, as well as the difficult parameter adjustments present in both the traditional clustering algorithms and the cluster ensemble techniques.

In this context of multi-objective clustering, Handl and Knowles in [15] described a Pareto-based multi-objective evolutionary algorithm, named multi-objective clustering with automatic K-determination (MOCK), able to simultaneously optimize two complementary clustering criteria: overall deviation and connectivity. MOCK returns a large number of different trade-off partitions over a range of different cluster numbers that are an approximation of the Pareto optimal set (set of solutions). However, as the number of alternatives increases, the analysis becomes harder [15]. To avoid this difficulty, MOCK includes a mechanism to automatically select the best partitions from the set of solutions. The selection is based on the shape of the approximation of the Pareto front.

As MOCK is the algorithm most closely related to ours, in order to put our results into perspective, we will compare them with those obtained with MOCK (Section 6).

3. MOCLE

In an attempt to overcome the difficulties of the traditional algorithms for cluster analysis, MOCLE combines characteristics from cluster ensemble and multi-objective clustering methods [11,12]. As any cluster ensemble method, MOCLE is composed of two main steps: (1) generation of a diverse set of base partitions and (2) determination of a consensus partition—Fig. 1. However, our approach differs from cluster ensemble methods in two ways. First, it looks for a set of “consensus” partitions instead of only one. In fact, its set of solutions can contain partitions that are combinations of other partitions, or partitions of high quality that already appeared in the set of individual partitions. Second, it combines pairs of partitions, iteratively, in an optimization process, instead of the usual combination of all partitions at the same time. This iterative combination/selection of the partitions avoids the negative influence of low quality base partitions, which
can decrease the quality of the results of the traditional cluster ensemble methods.

More formally, MOCLE works as follows. Consider a set of $n$ objects $X = \{x_1, x_2, \ldots, x_n\}$. A set of $n'$ objects $P = \{p_1, p_2, \ldots, p_n\}$, base partitions is constructed. $\pi_i = \{c_i^1, c_i^2, \ldots, c_i^{n'}\}$ is a partition of $X$ in $K^i$ clusters, such that $\cup c_i^j = X$. Then, the set of “consensus” partitions, $P = \{\pi_1, \pi_2, \ldots, \pi_{n'}\}$, is found by the optimization of several objective functions using a Pareto-based multi-objective genetic algorithm.

In order to serve to our purposes, we made two main adaptations in the application of the genetic algorithm. The first regards the initial population and the second is the introduction of a special crossover operator.

An important issue regarding our technique is that, like MOCK, it does not require a fine adjustment of parameters for its application to different data sets. The values of the parameters that we have employed depend mainly on the size of the data set. Thus, the user can easily adjust MOCLE’s parameters, without any additional knowledge on the algorithm or the data.

In summary, MOCLE automatically performs important steps of cluster analysis (Fig. 1). It runs several conceptually different clustering algorithms with various parameter configurations, combines the partitions resulting from these algorithms, and selects the partitions with the best trade-offs for different validation measures. In this manner, MOCLE represents a useful approach to exploratory data analysis. It results in a concise and stable set of high quality clusters or partitions with clusters of several densities, for example). It is important to have partitions with different types of clusters at several refinement levels, so that MOCLE can receive as much information as possible to find the largest number of possible existing structures.

Each partition in $P_i$ is an individual, represented by an array of sets (Fig. 2). Each set, in its turns, represents a cluster and contains the labels of its objects.

3.2. Crossover operator

The special crossover operator, together with the initial population, is responsible for the ensemble aspect of MOCLE. This operator finds the consensus between two parent partitions. Any existing cluster ensemble method that can be applied to a pair of partitions can be used as MOCLE’s crossover operator. In this context, first, using binary tournament, two parents are selected to be combined: $\pi^1$ and $\pi^2$, respectively, with $K^1$ and $K^2$ clusters. The number of clusters $K^F$ in the resulting consensus partition $\pi^F$ is randomly chosen in the interval $[K^1, K^2]$. Next, the parent partitions are combined using the ensemble method chosen. The result is a consensus partition with $K^F$ clusters (or, at most $K^F$ clusters, depending on the ensemble chosen). Fig. 3 illustrates the functioning of our crossover operator.

3.3. The optimization process

As previously mentioned, a Pareto-based multi-objective genetic algorithm is used to optimize several objective functions and construct the set of solutions $P$. Any Pareto-based multi-objective algorithm can be employed for this purpose.

The use of this class of genetic algorithms results in a set of partitions, instead of a single partition produced by traditional clustering and cluster ensemble methods. This is an important feature in domains like Bioinformatics, where data can have several interpretations. $P$ is an approximation of the Pareto optimal set [7,1]. That is, the “consensus” partitions represent the best compromises of the objective functions that could be obtained by the genetic algorithm. The approximation of the Pareto optimal set will be referred here just as Pareto set, for short.

The objective functions to be optimized should represent validation indices able to measure the quality of partitions in different ways. They should complement each other, favoring a different clustering criterion among those considered for the creation of the initial population. The objective functions are responsible for the selection of the high quality partitions and the robustness of MOCLE with respect to different data conformation.

With our special crossover operator, the partitions are combined in pairs, iteratively, during the evolutionary process. The consensus partitions, generated at each iteration, are also considered in the next combinations. This iterative process avoids the negative influence of low quality partitions, which is present in most of the traditional cluster ensemble methods. The low quality partitions are gradually discarded, while the best individual partitions and the good combinations are kept for further combination.

3.4. Restricted search space

In order to make the set of solutions concise, we attempt to restrict the search space to the base partitions and their

\[ \pi^1 = \{c_1^1, c_2^1, c_3^1\}, \pi^2 = \{c_1^2, c_2^2, c_3^2\}, \quad \pi^F = \{c_1^F, c_2^F, c_3^F\} \]

Fig. 2. Example of representation of an individual. $\pi^1 = \{c_1^1, c_2^1, c_3^1\}$ is a partition of the data set $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ in three clusters, $\pi^2 = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and $\pi^F = \{x_2, x_6\}$.
combination. Thus, we do not apply a mutation operator. Therefore, our use of genetic algorithm aims at selecting the best partitions, and not to explore all the space of possible partitions. In the typical Pareto-based multi-objective clustering scenario, differences in the assignment of only one object to a different cluster in two partitions can result in a different trade-off of the measures optimized. This can result in a high number of very similar partitions in the approximation of the Pareto set obtained.

In contrast, we argue that, in the context of clustering, the aim should not be the generation of the most complete Pareto set approximation possible. Indeed, having solutions representing each region of the Pareto front is enough to provide a relevant set of alternative partitions. Therefore, MOCLE aims to generate a concise set of solutions that are representative of the Pareto set. As already mentioned, MOCLE relies on the ability of the clustering algorithms in finding high quality partitions according to their clustering criteria. Starting with a set of potentially good partitions, MOCLE uses the multiple objectives to select the best compromises. New partitions are created only by means of the crossover operator and represent the consensus among other existing partitions. As this crossover operator only produces combinations of existing partitions and no mutation is employed, the search space will not be explored in detail. Thus, a large amount of similar partitions will not be produced by MOCLE, favoring the concision of the set of solutions obtained.

3.5. Computational cost

MOCLE and MOCK, like other multi-objective clustering techniques, are time-consuming. Nevertheless, there is a large number of applications, such as those in Bioinformatics, where the quality of the solutions obtained is more important than the computational time spent to find them. In fact, the computational complexity of the clustering methods represents often a small portion of the total time involved in these applications.

The high computational cost associated with multi-objective clustering techniques is, mainly, due to the large number of different tasks that need to be performed so that high quality final results can be achieved. For example, in the cases of MOCLE and MOCK, the computational effort of generating the initial population depends on how many and which clustering algorithms are used (e.g., k-means and single linkage have different computational complexity).

Besides the construction of the initial population, MOCLE and MOCK require a set of pre-processing operations before the optimization process starts. For instance, if the overall deviation and connectivity are used as the objective functions, a dissimilarity matrix and a list of the nearest neighbors of each object need to be calculated [15,11,12]. These calculations are $O(n^2d)$ and $O(n^2\log n)$, respectively, where $n$ is the number of objects in the data set and $d$ is its dimensionality. Then, based on these, the output of such objective functions can be calculated during the optimization process. More precisely, the computational complexities are $O(nd)$ and $O(nv)$, respectively, for the overall deviation and the connectivity, where $v$ is the number of nearest neighbors to be used for the calculation of the connectivity. In this paper, we employ MOCLE with such objective functions implemented with Euclidean distance and Pearson correlation, these calculations are performed twice (MOCK is implemented only with either of them).

One of the differences between MOCK and MOCLE computational complexity regards the crossover and mutations operators. MOCLE uses an ensemble algorithm for crossover and no mutation, while MOCK employs a traditional uniform crossover operator and a mutation operator. In MOCLE, the algorithm MCLA is used for crossover. The complexity of MCLA is $O(nk^2r^2)$, where $k$ is the number of clusters in the consensus partition and $r$ is the number of partitions to be combined [29]. In our case, we combined only two parent partitions. Thus, the complexity of our operator becomes $O(nk^2)$.

Another difference between the computational complexity of MOCK and MOCLE is due to the fact that, in contrast to MOCLE, MOCK includes a model selection phase. The implementation of this phase is computationally expensive. In fact, according to Handl and Knowles [15], this step governs MOCK’s overall complexity. Also, there are differences due to the computational complexity of the multi-objective genetics algorithms utilized in the optimization process of the MOCLE and MOCK. MOCLE employs the non-dominated sorting genetic algorithm (NSGA-II) whose computational complexity is $O(MP^2)$, where $M$ is the number of objectives and $P$ is the size of the population [32]. MOCK uses PESA-II whose computational complexity is $O(M(P + N)^2)$, where $N$ represents the size of the archive containing the nondominated solutions. That is, in the worst case PESA-II can be more time-consuming than NSGA-II.

4. Material and methods

4.1. Clustering methods and recovery measure

Four clustering algorithms are used to generate the base partitions that compose the initial population: complete linkage (CL), average linkage (AL), k-means, and spectral clustering (SPC). These algorithms have been chosen to provide a wide range of recovery effectiveness, as well as to increase the results generality. In the generation of the partitions, these methods are implemented with Pearson correlation and Euclidean distance.

The class of hierarchical clustering algorithms, more specifically the agglomerative ones, are procedures for transforming a proximity matrix into a dendrogram [17]. Such algorithms start with each object representing a cluster. Then, the algorithms gradually merge the current clusters into larger clusters. That is, they start with a trivial clustering in which each object is in a unique cluster, ending with a trivial clustering in which all objects belong to the same cluster. In this paper two broader used variations of the hierarchical clustering are employed: complete linkage and average linkage. These variations differ in the way the distance between two clusters is calculated. The CL employs the farthest distance of a pair of objects from two clusters to define the inter-cluster distance. In the case of the AL, the distance between two clusters is calculated by the average distance between all the objects in one group and all the objects in the other group. AL has been extensively used in the literature of gene expression analysis [4,5,25,27], although experimental results have shown that, in many cases, the CL outperforms it [9].

Another method widely used in the gene expression data analysis is the k-means algorithm [25,27]. K-means is a partitioning iterative algorithm that optimizes the fitting between clusters and their representatives using a predefined number of clusters [17,20]. Starting with prototypes values from randomly selected objects, the method works on two alternating steps: (1) an allocation step, where all objects are allocated to the cluster with the least dissimilar prototype and (2) a representation step, where a prototype is constructed for each cluster. A problem of this algorithm is its sensitivity to the selection of the initial centroids. This could lead to the convergence to a local minimum [17]. In order to prevent the local minimum problem, a number of runs with different initializations are often performed. The best run, based on some cohesion measure, is taken as the result. Another characteristic of this method is its robustness to noisy data.
Spectral clustering is a general class of algorithms. Its algorithms use the spectrum of similarity matrices to reduce the dimensionality of a data set and, then, apply a basic clustering algorithm, such as k-means or graph cut-based methods, on this lower dimension data [24]. For instance, assume that we have a graph, whose nodes are the objects and the edges are weighed by the similarity between two nodes. Spectral clustering algorithms can be interpreted as performing a random walk in this graph, finding clusters by ignoring edges rarely transversed in the walk. More specifically, in this paper, for a given similarity matrix S obtained with a Gaussian similarity function, we (1) calculate its normalized Laplacian matrix, and (2) perform an eigenvalue decomposition of this matrix. Next, we choose the k eigenvectors related to the k lowest eigenvalues, and perform k-mean clustering with them. Among other interesting features, spectral clustering algorithms make no assumptions on the data distribution. They are also able to find clusters that are not in convex regions of the search space.

Regarding the index employed to measure the success of the algorithm in recovering the true partition of the data sets, as in [22], we use the corrected Rand (CR) [17,22]. The corrected Rand index results in values from −1 to 1, with the value 1 indicating a perfect agreement between the partitions, and values near 0 or negatives corresponding to cluster agreement found by chance. Different from the majority of other indices, the corrected Rand is not biased toward a given algorithm or number of clusters in the partition [17,21].

Formally, let \( U = \{u_1, \ldots, u_k \} \) be the partition found by a clustering solution, and \( V = \{v_1, \ldots, v_l \} \) be the partition formed by an a priori information independent of the partition \( U \) (the gold standard). The corrected Rand is defined as follows:

\[
CR = \frac{\sum_i \sum_j (n_{ij} - \bar{n}_{i} \bar{n}_{j})}{\sqrt{2} \left( \sum_i n_{i} \bar{n}_{i} + \sum_j n_{j} \bar{n}_{j} \right) - \left( \sum_i n_{i} \bar{n}_{i} \right) \left( \sum_j n_{j} \bar{n}_{j} \right)}
\]

where (1) \( n_{ij} \) represents the number of objects in the clusters \( u_i \) and \( v_j \); (2) \( \bar{n}_{i} \) indicates the number of objects in the cluster \( u_i \); (3) \( n_{j} \) indicates the number of objects in the cluster \( v_j \); (4) \( n \) is the total number of objects; and (5) \( \bar{n}_{i}^{2} \) is the binomial coefficient \( a!/b!(a-b)\).

4.2. Data transformation

In many practical situations a data set could present objects whose attributes or features values (in our case, genes) lie within different dynamic ranges [17,22]. In this case, for proximity indices, such as Euclidean distance, features with large values will have a larger influence than those with small values. However, this does not usually reflect their importance for defining the clusters. This problem is often tackled by transforming the feature values, so that they lie within similar ranges. There are several approaches to perform transformations of attribute values [17,22].

The data sets used in our studies have only the case involving numeric values. Thus, for the Euclidean distance, we analyze two different forms of feature (gene) transformation, which have been widely used in clustering applications [17,22]: one is based on the z-score formula (standardization) and the other scales the gene values to \([0, 1]\). The transformation, which uses the z-score formula, translates and scales the axes, so that the transformed feature (gene) will have zero mean and unit variance. Hereafter, for short, we will refer to this transformation as \( Z \). The second procedure (normalization), named here \( Z_2 \), involves the use of the maximum and minimum values on the gene. Assuming non-negative values, this transformation is bounded by 0.0 and 1.0, with at least one observed value at each of these end points [22]. A large scale study on the impact of these procedures for cancer gene expression data has been recently presented in [6].

5. Experimental design

In this section, we present the main aspects taken into account in the analysis performed. We first describe the data sets used in the experiments and, then, discuss the empirical aspects considered in the set of experiments carried out for this work.

5.1. Data sets

Six microarray data sets are included in this analysis. They were built from data available in [2,3,10,14,30,34]. As Table 1 illustrates, these data sets present different values for aspects like the number of objects (second column), number of classes (third column), distribution of objects within the classes (fourth column), dimensionality (fifth column), and dimensionality after feature selection (last column).

To be more precise, we focus our study on microarray data related with cancer. In this context, one of the main goals is to identify previously unknown cancer subtypes for which gene expression profiles are homogeneous within a subtype, but different between subtypes [2,10,34]. The discovery of new subtypes of a disease could aid the decision-making process related to the choice of existing treatments, as well as in the development of new target-specific therapeutics [23,25,27].

Microarray technology is usually available in two different platforms, cDNA and Affymetrix [23,25,27]. Measurements of Affymetrix arrays are often estimates on the number of RNA copies found in the cell sample, while cDNA microarrays values are ratios of the number of copies in relation to a control cell sample. Data from these platforms have distinct distributions. Hence, normalization procedures would have divergent impacts on each platform. In order to make our analysis less complex, we will approach only data produced by Affymetrix microarrays.

One feature of the data produced via Affymetrix microarray technology is the variability of the magnitude of the expression level of a gene. For example, in a given array, there could be a gene whose expression level is around 10 and other whose level is around 10,000. In fact, following other works, for our data sets, all genes with expression level below 10 are set to the minimum level of a gene. For example, in a given array, there could be a gene whose expression level is around 10 and other whose level is around 10,000. In fact, following other works, for our data sets, all genes with expression level below 10 are set to the minimum level of a gene.

Furthermore, in order to discard uninformative genes, we perform the following procedure. For each gene \( j \) (attribute), we compute the mean \( m_j \). However, before doing so, in order to remove extreme values, we discard the 10% largest and smallest

<table>
<thead>
<tr>
<th>Table 1 Data set description.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data set</strong></td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>Armstrong [2]</td>
</tr>
<tr>
<td>Drakovic [10]</td>
</tr>
<tr>
<td>Gordon [14]</td>
</tr>
<tr>
<td>Su [30]</td>
</tr>
<tr>
<td>Yeoh [34]</td>
</tr>
</tbody>
</table>
values. As a result, we transform every value $x_i^p$ to

$$y_{ij} = \log_2(x_i^p/m_j)$$

After the previous transformation, we select for further analysis genes whose expression level differed by at least $l$-fold, in at least $c$ samples, from their mean expression level across samples [25,27]. With few exceptions, the parameters $l$ and $c$ were chosen in order to yield a filtered data set with around at least 10% of the original number of genes (features). Finally, it is important to point out that the data transformed with the previous equation is only used in the filtering step.

### 5.2. Empirical study

In order to illustrate MOCLE’s ideas, we implemented MOCLE employing the multi-objective genetic algorithm non-dominated sorting genetic algorithm [8], with the meta-clustering algorithm [29] as the crossover operator and three different objective functions.

As the objective functions, we adopted two measures that have been successfully used in multi-objective approaches: overall deviation and connectivity. Besides its original implementation with Euclidean distance, we implemented two versions of each of these algorithms using two proximity indices: Pearson correlation coefficient ($\rho$) and Euclidean distance ($\mathcal{Z}$). From here on, we will call the combination algorithm/proximity index/sample transformation a configuration.

For each configuration, we generated a set of partitions with different numbers of clusters. Although the known class labels are not used in any way during the clustering, we take the number of known classes ($c$) as a reference to establish a range of numbers of clusters to be used in the generation of the initial population. Thus, we generated partitions with $k$ in the range $[c - 2, c + 2]$. For the data sets where $c = 2$, we set the minimum of the range to 2. This procedure generates an initial population of different size for each data set.

Moreover, in order to minimize the occurrence of suboptimal solutions, we run KM and SPC 30 times for each value of $k$, with a random choice of initial parameters. Among all 30 partitions produced for a given $k$, we selected the partition with the lowest squared error for the initial population. For the algorithms AL and CL, we generated the hierarchies and cut them in order to produce one partition for each value of $k$. To run the algorithms KM, AL and CL, we employed the Matlab statistical tool box. We also used a Matlab toolbox to run the experiments with the spectral algorithm (http://www.mathworks.com/products/statistics/).

In this way, the initial input to MOCLE are partitions originated from different configurations: for example, partitions generated with the data untransformed and Pearson correlation, partitions with transformed data and Euclidean distance, among other possibilities. However, during the run of the genetic algorithm, the objectives functions are evaluated based only on the untransformed data.

In the terms of parameter configurations, for the case of MOCLE, we proceeded as follows. We set the maximum number of generations to 50. In some preliminary experiments, we observed that increasing the number of generations did not modify the Pareto front approximation obtained. For each data set, the internal population was set to the initial partitions generated with the individual clustering algorithms, as previously described. The number of nearest neighbors (or more correlated objects)—$v$—necessary to calculate the connectivity and connectivity with correlation was set to 5% of the number of objects in the data set.

Table 2 illustrates the values of each parameter for MOCLE, for each data set. In this table, $n'$ stands for the number of partitions in the initial population and $v$ for the number of nearest neighbors (or most correlated objects for the connectivity by correlation). The fourth column ($k$ range) represents the interval from which the numbers of clusters used to generate MOCLE’s initial population were chosen.

In order to be consistent, in the case of MOCK, we set the number of nearest neighbors to the same value used with MOCLE. We also employed five control fronts, the default value. The maximum number of clusters (max $k$) corresponds to the largest number of cluster that a partition could have, so that it could be chosen to be in the recommended solution set. Also to be consistent with parameter values of MOCLE, when possible, we set max $k$ to the respective upper bound used in MOCLE. However,

<table>
<thead>
<tr>
<th>Data set</th>
<th>$n'$</th>
<th>$v$</th>
<th>$k$ range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrong</td>
<td>48</td>
<td>4</td>
<td>[2, 4]</td>
</tr>
<tr>
<td>Dyrskjot</td>
<td>64</td>
<td>2</td>
<td>[2, 5]</td>
</tr>
<tr>
<td>Gordon</td>
<td>48</td>
<td>10</td>
<td>[2, 4]</td>
</tr>
<tr>
<td>Su</td>
<td>80</td>
<td>9</td>
<td>[8, 12]</td>
</tr>
<tr>
<td>Yeoh</td>
<td>80</td>
<td>13</td>
<td>[4, 8]</td>
</tr>
<tr>
<td>Chowdary</td>
<td>48</td>
<td>6</td>
<td>[2, 4]</td>
</tr>
</tbody>
</table>
for the data sets Armstrong, Gordon and Chowdary, where the maximum should be four. MOCK encountered problems when executing with these values—it halted before finding any kind of solution. In order to overcome this problem, we set the maximum to five. We also encountered a problem with the data set Yeoh. We could not find a parameter configuration where MOCK could run. This probably occurred due to the very high dimension of this data set. Following the suggestion of MOCK’s authors (personal communication), for this data set, we used a different version of MOCK. In this case, uniform random noise was employed during the generation of the control data, instead of PCA.

As the results of MOCK, as in the case of traditional clustering algorithms, depend on proximity index used, we run MOCK for the same configurations adopted for the individual algorithms. Thus, for each data set, we have four different configurations for MOCK: (1) MOCK with Pearson correlation coefficient and the samples without any kind of data transformation (\(E_{Z1}\)); (2) MOCK with Euclidean distance and the samples without any kind of data transformation (\(E_{Z2}\)); (3) MOCK with Euclidean distance and the samples transformed with \(Z_1\)—standardization (\(E_{Z1}^*\)); and (4) MOCK with Euclidean distance with the samples transformed with \(Z_2\)—scaling to \([0,1]\) (\(E_{Z2}^\ast\)).

Finally, as MOCLE and MOCK are not deterministic, we run 10 times each of them with the same initial configurations.

6. Results and discussion

In order to put the results achieved with MOCLE into perspective, we discuss them in the context of the individual clustering algorithms used to form its initial population, as well as the results obtained with MOCK. First, as measure of quality, we use the corrected Rand obtained in the different scenarios. It is important to mention that the CR is used only for evaluation purposes. That is, we did not employ the CR, neither any kind of information on the membership of the objects to the true class, during the clustering process. In addition to the evaluation of the quality of the partitions, we discuss some issues on the numbers of clusters of the best partitions and the size of the sets of solutions of MOCLE and MOCK.

Besides the analysis mentioned, it is important to point out that MOCK and MOCLE are able to detect more than one underlying structure existing in a data set, as discussed in [11,12]. In this paper, we focus our attention to finding only one solution of MOCLE and MOCK.

6.1. Initial population

In order to analyze MOCLE from the perspective of its initial population, we do as follows. We calculated the CR between each partition in the initial population and \(n_{\text{known}}\). Then, for each configuration (clustering algorithm, proximity index and data transformation), we selected for further comparison the partition that presented the largest CR. By doing so, we illustrate the best partition that each configuration was able to find. These results are shown in Table 3. Besides the best CR for each configuration in the initial population, such a table shows the corresponding number of clusters found for each partition. In the context of this table, for each data set, the best CR obtained for a given algorithm, along with the different configurations employed, is highlighted in boldface. Also, for each data set, the best CR of all is marked with an *.

Before starting the discussion on the results obtained with MOCLE, we should observe some important aspects of the results presented by the individual clustering algorithms—Table 3. If, for each data set, we compare the CR of the partitions generated with a given clustering algorithm, along the different configurations employed, we can observe that, for the majority of the cases, there was no single proximity index and data transformation configuration able to always lead to the best CR values. That is, for each data set and clustering algorithm, a different configuration led to the best CR. This issue is further discussed in [6]. Exceptions were the data sets Yeoh and Chowdary, where the best partitions were always found with the Pearson correlation. For the data set Yeoh, one should further observe that no partition of high quality was found (the largest CR of the partitions in the initial population was equal to 0.39). This could have happened due to the very high dimensionality of this data set, which could have obscured the underlying structure of the data. Another possible reason for this poor result could be the fact that underlying structure of the data is not in accordance with any of the clustering criteria used with the individual clustering algorithms.

Thus, in order to analyze a data set with the traditional clustering algorithms, the domain expert has to know a priori what configuration should be employed or, as it is more often, he/she has to employ several configurations and select, according to some criterion, the best partition generated. However, there is no effective strategy to select which algorithm, proximity index and data transformation are more suitable for a given data set nor an unbiased strategy to select a partition given a set of them [9,16].

6.2. MOCK and MOCLE

In order to evaluate the results from MOCLE and MOCK, we first verify if the algorithms were able to find at least one good
solution, which is evaluated according to its CR with respect to the true structure. We consider a good solution, that with a medium (0.5 ≤ CR < 0.7) to large (0.7 ≤ CR ≤ 1) value of CR. In order to do so, we proceed as follows.

For each of the 10 runs of MOCLE and MOCK, we calculated the CR between each solution partition, $\pi_i^S \in \Pi_S$, and the known structure of the data set, $\pi_i^{known}$. Then, we selected the best partition in $\Pi_S$ (the partition $\pi_i^S$ with the largest CR when compared with $\pi_i^{known}$). Next, we calculated the mean (M) and standard deviation (SD) of the CR of these partitions over the 10 runs.

As Pareto-based multi-objective algorithms, MOCLE and MOCK return a set of solutions with $n^2$ partitions. The objective of MOCK is to generate an approximation of the Pareto front as complete as possible. Since such a set is usually very large, which makes it infeasible to be used in practice, MOCK employs a selection procedure. This procedure chooses, according to some quality criterion, a subset of partitions from the approximation of the Pareto front to be the recommended partitions. Thus, the recommendation set is the actual output from MOCK. However, we have observed in previous works that, in most of the cases, MOCK’s recommendation set contained only partitions with very small CR (poor quality) when compared to the others included in the corresponding complete Pareto front approximation [11,12].

Thus, in this paper, in order to have a broader idea of the performance of MOCK, we consider two sets of solution: (1) the complete Pareto front approximation obtained, referred to as MOCK-C, which has usually a very large number of partitions; and (2) the set of recommended solutions, which is the actual output from MOCK. This set will be denoted by MOCK-R.

Different from MOCK, the purpose of MOCLE is to generate a set $\Pi_S$ as concise (small) as possible, while maintaining the quality of the solutions. Thus, in such a context, instead of having a further step to select a subset of the solutions found as MOCK does, the complete set of solutions provided by MOCLE is usually small enough to allow the domain expert to individually explore each of the solutions to validate it.

### 6.2.1. MOCK

In terms of results obtained with the multi-objective clustering algorithms, Table 4 shows the mean (M) and standard deviation (SD) of the best CR for each MOCK configuration. The largest values of CR are highlighted in boldface. In order to not overload Table 4 with too much information, the number of clusters of the best partitions is shown in Table 5.

Observing the results of MOCK (Table 4), a similar behavior to that of the individual clustering algorithm occurred: for each data set, the partition with best CR was obtained with a different configuration of proximity index and data transformation. An interesting aspect of the results achieved with MOCK is the poor quality (very small CRs) of the partitions produced with Pearson correlation, in both sets, MOCK-C and MOCK-R. In fact, for all but Chowdary data set, MOCK with such a proximity index configuration found only partitions with CR very close or much smaller than 0.1. To a certain extent, this is an unexpected result in that for several data sets the best CR of the individual algorithms was achieved with Pearson correlation (Armstrong, Su and Chowdary).

We can also see that, in most of the cases where MOCK-C contained partitions with large CRs, these partitions were not selected to be part of MOCK-R. That is, most of the time, MOCK was not able to give as recommendation to the user the best partition found. Indeed, only for two data sets MOCK was able to recommend partitions presenting CR values close to the best ones obtained in the complete set of solutions: one configuration for Su data set and all configurations for Chowdary. As previously mentioned, we have already observed this problem in the context of other experimental analysis [11,12]. Such analysis included data sets of reduced dimensionality. Thus, this discards the high dimensionality of the data used in this paper as the source of MOCK’s difficulty [11,12].

### 6.2.2. MOCLE

Now, we focus our attention to the results obtained with MOCLE. Table 6 illustrates the mean (M) and standard deviation (SD) of CR and $k$ for MOCLE. We can observe that MOCLE, in most of the cases, found partitions of intermediate or large CR values (CR > 0.5). The only exception occurred with the data set Yeoh, which also imposed difficulties to the individual algorithms, as previously discussed. In fact, no algorithm with any configuration found a satisfactory partition for this data set.
6.3. Comparing the algorithms

If we compare MOCLE’s results (Table 6) with those of the individual clustering algorithms (Table 3), we observe that in two cases MOCLE selected the best initial partition (Dyrskjot and Chowdary). In the other cases, MOCLE found partitions of quality slightly above the best individual ones. In fact, looking at Tables 3, 4 and 6, we can observe that, in most of the cases, the initial population presented at least one partition of higher quality than the partitions obtained with MOCK and MOCLE.

Comparing MOCK and MOCLE, we also observe that, for MOCK-C, in most of the cases, at least one configuration contained a partition with larger CR. In contrast, when we look at MOCK-R, which would be the actual output of MOCK, MOCLE always found a solution with a larger CR.

In the case of MOCLE, we must consider that the objective functions use the data with no transformation, although such transformation was considered in the construction of the initial population. Thus, MOCK with EZ is the configuration of MOCK more similar to MOCLE. In this context, even using MOCK-C as reference for the comparison, MOCK performed better than MOCLE.

Furthermore, if we compare MOCLE with the partitions in the initial population obtained with configuration EZ, MOCLE always found a partition of similar (Armstrong, Dyrskjot, Su and Yeoh) or much larger (Gordon and Chowdary) CR. For the last case, the partitions of the initial population obtained with EZ showed very small CR. In contrast, the partitions that were given as output by MOCLE are similar—in terms of CR—to those in the initial population that were generated with EZ. Thus, this illustrates the positive impact of the addition of the connectivity objective function using Pearson correlation. Another evidence for this is the fact that MOCK using EZ and the same objectives as MOCLE, with exception of the connectivity objective function using Pearson correlation, presented the worst partitions. In fact, as a whole, we can observe that MOCLE was able to select either a good partition obtained with the Euclidean distance or a good partition obtained with Pearson correlation.

6.4. Numbers of clusters

Now, we turn our attention to the number k of clusters of the partitions with best CRs. For instance, looking at Table 3, we can see that, for most of the data sets, the best partitions in the initial population contain the true number of clusters. On the other hand, in most of the cases, for MOCK-C (which presented the best CRs for MOCK) and MOCLE (Tables 5 and 6) the best partitions for each data set present a number of clusters larger than the true one. For MOCK-C, in several cases, k is even larger than the maximum value allowed for MOCLE (e.g., Armstrong, Su and Yeoh). It could be the case that if MOCLE was fed with partitions presenting values for k much larger than the exact number of classes, the algorithm could be able to output partitions with larger CRs. This question should be investigated in further works.

For the data sets Gordon and Chowdary, the value of k for the best partitions was the true one for MOCK-C, MOCLE and the initial population.

6.5. Size of the solution’s sets

Another important aspect of our analysis illustrates the effectiveness of MOCLE in obtaining a concise set of partitions with large CRs as output. This aspect is the number of partitions in the sets of solutions. To put this into perspective, in Table 7, we show the number of partitions (nS) in MOCK-C and MOCK-R. The mean (M) and the standard deviation (SD) of nS over the 10 runs is given. In the case of MOCLE, we present the number of partitions in the initial population (nI), as well as the size of the corresponding set of solutions (nS), in Table 8. For nI, we show the mean (M) and the standard deviation (SD) over the 10 executions. In order to give an idea of the reduction (or increase) of the size of the final population (nS) with respect to the initial population (nI), the column nS/nI indicates the ratio between nS and nI.

Based on Table 8, we can see that MOCLE’s purpose of producing a concise set of solutions was achieved. In most of the cases, we can observe a great reduction in the size of MOCLE’s nS, when compared with its initial population size. The only exception happened with the data set Gordon, where, on average, we had an increase in the number of solutions. For this data set, differently to what occurred to the remaining ones, we can observe a great variation in the number of solutions along the different runs, indicated by a large standard deviation. For the other data sets, in the worst case we observe a reduction of 88% in the number of initial solutions; and, in the best case, a reduction of 70.8%.

Compared to the set of solutions obtained with MOCLE, we can observe from Table 7 that MOCK-C is, in general, very large. This was an expected result, as previously discussed, in that MOCK is intended to generate the most complete approximation of the Pareto front possible. There were two exceptions: the configuration using PZ, for all data sets, in which all partitions in MOCK-C also presented a very low CR, and all configurations for the data set Dyrskjot.

6.5. Size of the solution’s sets

Table 6

<table>
<thead>
<tr>
<th>Data set</th>
<th>CR</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrong</td>
<td>0.50</td>
<td>3</td>
</tr>
<tr>
<td>Dyrskjot</td>
<td>0.61</td>
<td>5</td>
</tr>
<tr>
<td>Gordon</td>
<td>0.91</td>
<td>2</td>
</tr>
<tr>
<td>Su</td>
<td>0.50</td>
<td>12</td>
</tr>
<tr>
<td>Yeoh</td>
<td>0.18</td>
<td>8</td>
</tr>
<tr>
<td>Chowdary</td>
<td>0.92</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7

<table>
<thead>
<tr>
<th>Data set</th>
<th>Set of partitions</th>
<th>nS</th>
<th>nI</th>
<th>M</th>
<th>SD</th>
<th>M</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armstrong</td>
<td>MOCK-C</td>
<td>62</td>
<td>90</td>
<td>6.8</td>
<td>0.0</td>
<td>6.0</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>MOCK-R</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Dyrskjot</td>
<td>MOCK-C</td>
<td>18.8</td>
<td>18.8</td>
<td>6.1</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOCK-R</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Gordon</td>
<td>MOCK-C</td>
<td>111.7</td>
<td>19.9</td>
<td>4.2</td>
<td>1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOCK-R</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Su</td>
<td>MOCK-C</td>
<td>79.7</td>
<td>11.7</td>
<td>6.0</td>
<td>2.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOCK-R</td>
<td>0.8</td>
<td>0.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Yeoh</td>
<td>MOCK-C</td>
<td>29.0</td>
<td>8.1</td>
<td>7.7</td>
<td>2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOCK-R</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Chowdary</td>
<td>MOCK-C</td>
<td>73.8</td>
<td>9.1</td>
<td>54.3</td>
<td>4.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOCK-R</td>
<td>0.8</td>
<td>0.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Besides the poor quality of the partitions in MOCK-R, previously discussed, in Table 7 we can observe another problem with MOCK-R. For several cases, MOCK did not recommend a solution. In some cases, this occurs for all runs (mean of $n^3$ equal to zero) and in other cases, just some of the runs recommended solutions (mean of $n^3$ less than one).

7. Final remarks

In the paper [11], which is the most closely related to the results presented in this current paper, we applied two versions of MOCLE to one artificial data set, two benchmark data sets from UCI repository and two real world data sets. These two real data sets were the only related to gene expression data. Both versions of the algorithm employed the same four clustering algorithms (complete linkage, average linkage, k-means and SNN) to construct the initial population. We employed the same objective functions as MOCK: connectivity and overall deviation. For the crossover operator, we utilized the MCLA. The difference between the versions was the multi-objective genetic algorithms used for MOCLE’s optimization process (NSGA-II and SPEA). Also, we used only the Euclidean distance as the proximity measure. In this case, except from one data set, we did not do any kind of data transformation.

As motivated in the introduction, in this paper, differently from Faceli et al. [11] that analyzed five data sets representing problems of very distinct domains, we focused our analysis on gene expression data sets. That is, we developed an investigation of the behavior of MOCLE and MOCK into the context of gene expression data. Also, in contrast to Faceli et al. [11], for the first time, in order to construct the initial population, besides using partitions that were built with the Euclidean distance as proximity measure, we added partitions that were generated with Pearson correlation coefficient.

In this current paper, for the case of the Euclidean distance, we did use and investigate the impact of employing different kinds of data transformations to construct the initial population: (1) untransformed data, (2) data transformed to have a mean of 0 and standard deviation of 1 (standardization), and (3) the data scaled to the interval $[0, 1]$. A minor difference between the experiments developed in this paper and the one in [11] is the substitution of SNN algorithm for the spectral clustering.

Besides the differences in forming the initial population and data set used, with respect to Faceli et al. [11], we included a version of the connectivity using Pearson correlation as a third objective function. Adding this function, together with the other modifications, as could be seen in the experimental results, has brought advantages. The version of MOCLE presented in this paper was able to select either a good partition obtained with the Euclidean distance or a good partition obtained with Pearson correlation. This behavior would not have been possible without the inclusion of the Pearson correlation.

Indeed, with these new features, in this paper, we showed that MOCLE was very effective in selecting to the final set of solutions partitions with high quality (large corrected Rand). Such partitions were initially built either according to Euclidean distance or according to Pearson correlation. Furthermore, the resulting sets were very concise when compared to the size of the initial population used as input.

As in the evaluation of objective functions MOCLE uses only the untransformed data, the algorithm was not able to capture, in the initial population, good quality partitions that were built using some kind of data transformation. Thus, as a future work, we will investigate a strategy to include these transformed data in the evaluation of MOCLE’s objective functions.

Acknowledgments

We would like to thank the financial supported from FAPESP and CNPq. We also are grateful to Julia Handl for her prompt assistance with MOCK.

References
