Multiobjective clustering around medoids

Julia Handl
University of Manchester
J.Handl@postgrad.manchester.ac.uk

Joshua Knowles
University of Manchester
J.Knowles@manchester.ac.uk

Abstract- The large majority of existing clustering algorithms are centered around the notion of a feature, that is, individual data items are represented by their *intrinsic* properties, which are summarized by (usually numeric) feature vectors. However, certain applications require the clustering of data items that are defined by exclusively extrinsic properties: only the *relationships* between individual data items are known (that is, their similarities or dissimilarities). This paper develops a straightforward and efficient adaptation of our existing multiobjective clustering algorithm to such a scenario. The resulting algorithm is demonstrated on a range of data sets, including a dissimilarity matrix derived from real, non-feature-based data.

1 Similarity-based pattern recognition

A large majority of techniques for pattern recognition are designed for the analysis of data that are described by means of numerical feature vectors. Well-known examples of such techniques are principal component analysis (for visualization), which directly operates on a matrix of data vectors, the k-means algorithm (for clustering), which relies on the computation of numerical averages of feature vectors, or support vector machines (for classification), which map numerical input vectors to a higher dimensional kernel space.

Similarity-based pattern recognition comprises that part of the field that deals with algorithms applicable in a scenario, where only dissimilarity data (or similarity data) describing the *relationships* between data items is available. Such data arises in a variety of different domains, including psychological research, computer vision and bioinformatics. An important example of a bioinformatics application is the sequence-based classification of genes or proteins. Powerful sequence alignment programs like the Basic Local Alignment Search Tool (BLAST, [1]) return similarity scores between individual input sequences. The efficient processing of large amounts of such outputs relies on the application of similarity-based clustering techniques.

Scope of this paper

In previous work, we have developed a multiobjective clustering algorithm and shown its advantages over traditional single-objective clustering techniques [5, 6, 7, 8]. The algorithm was designed for the use on numerical data only, and explicitly exploited the numerical feature representation at several distinct stages during the clustering process. In this manuscript, we demonstrate how the algorithm can be extended so that the processing of similarity-based data can be included.[1]

The remainder of this paper is structured as follows. Section 2 introduces the basics of our existing multiobjective clustering technique. The modifications necessary to directly apply the algorithm to dissimilarity data are introduced in Section 3. Section 4 describes the experimental setup used for the evaluation of the algorithm on feature-vector-derived data, with Section 5 presenting results. In Section 6, the algorithm is applied to a real data set that is inherently similarity-based. Finally, Section 7 concludes.

2 Multiobjective clustering

Our existing multiobjective clustering algorithm MOCK (MultiObjective Clustering with automatic K-determination, [8]) is based on the elitist multiobjective evolutionary algorithm, PESA-II, described in detail in [2]. It optimizes two clustering objectives, *overall deviation* and *connectivity*, which reflect two fundamentally different aspects of a good clustering solution: the global concept of *compactness of clusters*, and the more local one of *connectedness of data points*.

The encoding employed is the locus-based adjacency scheme proposed in [18]. In this graph-based representation, each individual g consists of N genes $g_1, \ldots, g_N$, where N is the size of the clustered data set, and each gene g_i can take allele values j in the range $\{1, \ldots, N\}$. Thus, a value of j assigned to the ith gene, is then interpreted as a link between data items i and j: in the resulting clustering solution they will be in the same cluster. The decoding of this representation requires the identification of all subgraphs, and all items belonging to the same subgraph are assigned to one cluster. This can be done in linear time.

The operators used are standard uniform crossover, and a specialized initialization and mutation scheme. MOCK's initialization is based on minimum spanning trees (MSTs) and the k-means algorithm [17]. For a given data set, the complete MST is computed using Prim's algorithm. Individuals corresponding to different clustering solutions are then obtained by breaking up the MST, using a measure of *interestingness* of individual links and the partitionings prescribed by the k-means solutions. This has the effect of generating solutions with a range of cluster numbers that already provide a good and well-spread approximation to the Pareto front. MOCK's mutation operator allows data items to be linked to one of their L nearest neighbours only. Hence, $\forall i, g_i \in \{nn_{i1}, \ldots, nn_{iL}\}$, where nn_{il} denotes the

[1] Clearly, an algorithm designed for similarity-based data continues to be applicable to feature-based data, as this only requires the choice of a suitable distance function between feature vectors.

lth nearest neighbour of data item i. This has the effect of significantly reducing the size of the search space.

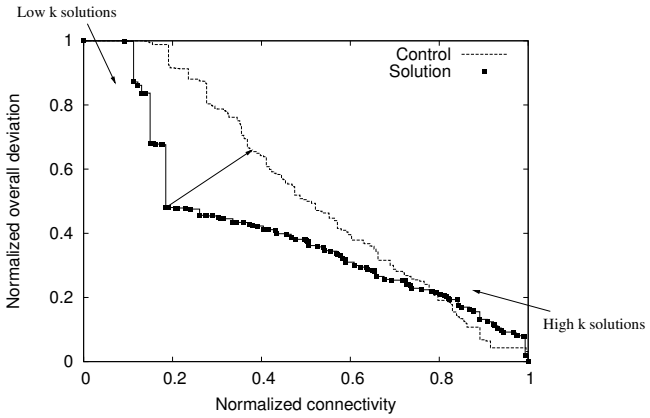


Figure 1: Solution and control reference front for a run of MOCK on a simple four-cluster data set (from [6]). Solutions are scored by their distance to the reference front (their 'attainment score'). The solution with the largest minimum distance to the reference front is indicated by an arrow, and corresponds to the correct four-cluster solution. In general, all solutions situated at distinct knees in the Pareto front are of high immediate interest. Several such solutions can exist and they often correspond to cluster structures on different levels. In a plot of the attainment scores as a function of the number of clusters, such distinct 'knees' are manifest as local maxima.

The algorithm generates clustering solutions that correspond to different trade-offs between the two clustering objectives and contain different numbers of clusters. In order to reduce the number of solutions to consider, an automated technique was developed, which selects good solutions from the resulting Pareto front, and, thus, simultaneously determines the number of clusters k in a data set (see Figure 1). This method of solution selection is based on the shape of the Pareto front. Specifically, it tries to determine 'knees' in the Pareto front that correspond to good solutions. A comparison to a null model, that is, random control data is used in order to correctly determine these knees. The use of the null model helps to abstract both from k-specific biases in the two objectives, and from biases introduced due to the shape of the underlying data manifold. A detailed motivation and description of this methodology is provided in [8].

In previous work [6, 7, 8], MOCK has been compared to three single objective clustering algorithm, an advanced ensemble method [20] and the Gap statistic [21], and results indicated a clear advantage to the multiobjective approach.

3 Adaptation of MOCK for dissimilarity data

As mentioned above, MOCK uses the notion of a feature at three different stages during the clustering process.

1. The k-means algorithm [17] is employed during the initialization phase. k-means is evidently feature based, as it requires the computation of *cluster centroids* (the average feature vector of a cluster), and of the *intra-cluster variance* (the sum of the squared distances between data items and their corresponding cluster centroids):

$$\mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} \vec{i}$$

$$Var(C) = \sum_{C_k \in C} \sum_{i \in C_k} \delta(i, \mu_k)^2,$$

   where C is the set of all clusters, $\vec{i}$ is the feature vector describing data item i, μ_k is the centroid of cluster C_k and $\delta(.,.)$ is the chosen distance function.

2. MOCK's second objective, overall deviation, is almost identical to k-means' criterion of intra-cluster variance. It equally requires the computation of cluster centroids and sums up the distances between data items and their cluster centroids:

$$Dev(C) = \sum_{C_k \in C} \sum_{i \in C_k} \delta(i, \mu_k),$$

   where C is the set of all clusters, μ_k is the centroid of cluster C_k and $\delta(.,.)$ is the chosen distance function.

3. The computation of MOCK's null model requires a principal component analysis (PCA) on the matrix of all feature vectors. Specifically, a null model suggested by Sarle is used [19], which reproduces the shape of the data manifold, approximately. It is based on a Poisson model in the space of principal components: the control data is generated uniformly randomly in PCA-space, within a hyperbox with sides proportional in length to the eigenvalues.

In this section we suggest replacements for each one of these mechanisms, thus enabling MOCK to work directly on dissimilarity data.

3.1 Cluster medoids versus cluster centroids

A concept closely related to that of a cluster centroid is a *cluster medoid*, which is defined as the most centrally located item in a cluster, that is, the item in the cluster whose average dissimilarity to all other items in the same cluster is minimal [12]. Cluster medoids have been employed in a number of clustering algorithms including the algorithms *Partitioning Around Medoids* (PAM, [12]) and *Clustering Large Applications* (CLARA, [12]).

Clearly, the computation of cluster medoids does not require the presence of feature vectors, but can be done for dissimilarity data. The first two problems in the above list can therefore be solved through the replacement of cluster centroids by cluster medoids.

3.1.1 Computational complexity

Unfortunately, a major disadvantage of cluster medoids is the computational complexity of their computation, which

Algorithm 1 Naïve computation of the cluster medoid

```
1:  procedure COMPUTE_MEDOID
2:      set_of_medoids := {}
3:      for each k in 1 to K do
4:          medoid := -1
5:          min_distance := inf;
6:          for each i in 1 to size[k] do
7:              sum_d := 0
8:              for each j in 1 to size[k] do
9:                  sum_d := sum_d+d[i, j]
10:                 if sum_d ≥ min_d then
11:                     break
12:                 end if
13:             end for
14:             if sum_d < minimum_distance then
15:                 min_d := sum_d
16:                 medoid := i
17:             end if
18:         end for
19:         set_of_medoids := set_of_medoids ∪ medoids
20:     end for
21:     return set_of_medoid
22: end procedure
```

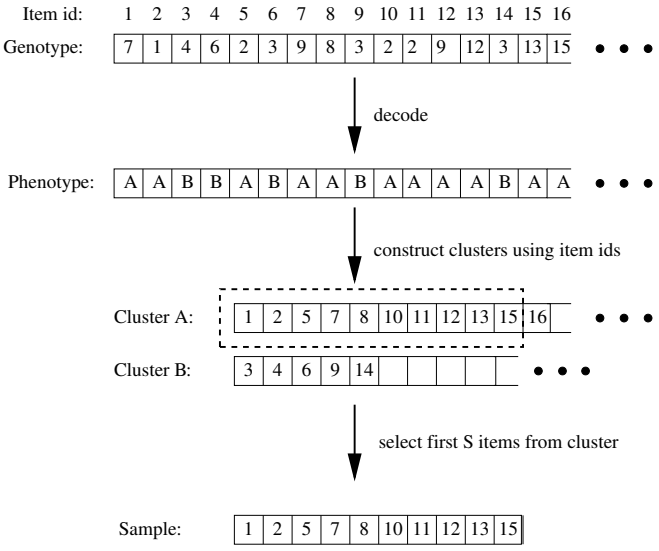


Figure 2: Computation of an approximate medoid for cluster A. For each data item j, a sample consisting of the first 10 data items from cluster A that are different to j are selected, and the total dissimilarity of j to this set of data items is computed. This is repeated for all data items. The data item with the lowest total dissimilarity is selected as the medoid, and the exact overall deviation with regard to this medoid is computed.

is quadratic in the size of the data set (whereas the computation of cluster centroids is only linear in N).

A naïve approach to the computation of cluster medoids is given in Algorithm 1. While the approach is slightly accelerated by the introduction of the break clause in Line 11 (which stops distance computation for a given data item once the current minimum summed distance has been exceeded), the Algorithm remains computationally very expensive. Its only advantage is its constancy in the dimensionality D of the data set (whereas the computation of cluster centroids is linear in D).

However, usually, $N \gg D$, thus there is a significant rise in the complexity of both MOCK's initialization scheme and its second objective, overall deviation. The latter of these is a major concern: MOCK's main computational cost lies exactly in the evaluation of the objective functions, and this rise in complexity results in a significant deterioration of MOCK's time efficiency for large data sets.

3.1.2 Linear-time sampling scheme

We have found that the computational complexity of MOCK's second objective can be reduced to linear time by means of a simple sampling scheme. Specifically, only a sample of S data items in the same cluster are used for the selection of the medoid.

The sampling scheme employed in this paper is illustrated in Figure 2. The only change introduced to Algorithm 1 is a modification of the for-loop in Line 8. In the original for-loop, the distance of a fixed data item i to all other data items j in the same cluster is summed up. In our sampling scheme, the set of items j is reduced to a sample of the first $S = 10$ data items in the same cluster, which are not identical to i (for clusters of size less than 12, the medoid is computed exactly). Here, the actual clusters (and the order of the data items within the clusters) are constructed in a deterministic fashion: data items are assigned to the clusters in the order imposed by the position of their gene in the genotype (their *item id*, see Figure 2).[2]

As a result, the set of items employed for the medoid selection only changes when one of the first 10 data items is removed from the cluster, or a new cluster member is introduced within the region of the genome that is spanned by the first 10 data items. Hence, the sample used within a given cluster is kept *relatively constant*, and this has the effect of providing much better *relative* estimates of sum_d, and hence more accurately determining the medoid. We have found this is crucial for the convergence of the algorithm: a comparison to random sampling schemes (such as random re-sampling in each evaluation or random re-sampling for each data item) shows that these prevent convergence of the algorithm (results not shown).

3.2 Adaptation of the null model

Regarding the third item to be adapted in the list above — MOCK's null model, a new method for the generation of the control data needs to be found.

3.2.1 Permutation of dissimilarity values

A null model commonly used for dissimilarity data is a random permutation of all values within the dissimilarity matrix [10, 16]. However, experimental analysis (see the ex-

[2]Importantly, the order of the input data is randomized, that is, data items belonging to the same cluster are not necessarily neighbouring, but are spread across the genome.

ample in Figure 3) shows that this approach has little capacity to capture the shape of the actual data manifold. In fact, our experiments confirm that – as previously pointed out by [11] – the null hypothesis (of no cluster structure in the data) is implausible even in the absence of clusters (that is, the null hypothesis will usually be rejected for natural data sets, even if these do not contain actual cluster structure). This is due to the fact that a permutation of all dissimilarity values does not simply decorrelate dissimilarity values, but it additionally results in the introduction of non-metric relationships. Specifically (as illustrated in Figure 4) the triangular inequality is no longer valid, which may be implausible for many real data sets.

Also, as no new dissimilarity values are introduced, the shape of the data manifold cannot actually be modeled by means of a simple permutation. For example in Figure 4, the manifold of interest would be the region within the convex hull spanned by the the given data. For this purpose, data items within the entire region would need to be generated, clearly requiring dissimilarity values *distributed* within $[\epsilon, \sigma]$, and not restricted to ϵ and σ only.

3.2.2 Multidimensional scaling

The null model proposed in this paper is based on the observation that, in the case of MOCK's two objectives, the biases introduced by the shape of the data manifold are independent of its actual location or orientation. This property can be exploited in the generation of the control data.

Multidimensional scaling (MDS, [3]) is a visualization technique specifically designed to derive spatial position for data items based on the dissimilarity information between data items only. Multidimensional scaling provides an embedding of a given data set into a Euclidean space of specified dimensionality (usually 2D for visualization), such that distances between data items are preserved, but the actual positions of individual data items are meaningless. Clearly, this problem is underdetermined: multidimensional scaling therefore only returns one out of many possible solutions, and infinitely many alternative solutions can be obtained through simple translations or rotations of a single solution.

Metric multidimensional scaling is an exact method for scaling, and it is equivalent to performing a principal component analysis on the squared, double-centred dissimilarity matrix. The computational complexity of metric multidimensional scaling rises as N^3, where N is the size of the data set. Non-metric multidimensional, in contrast, comprises a set of approximation methods, which are computationally more efficient than principal component analysis and do not require the dissimilarity data to satisfy metric constraints.

In this work, we therefore decide for the use of a method for non-metric multidimensional scaling, the so-called Sheppard-Kruskal algorithm [14], which iteratively minimizes stress between the original dissimilarities and the Euclidean distances in target space. In our experiments, the target space is chosen to be 10-dimensional, independent of the real dimensionality of the input data. In each iteration, the Sheppard-Kruskal heuristic (randomly) selects one

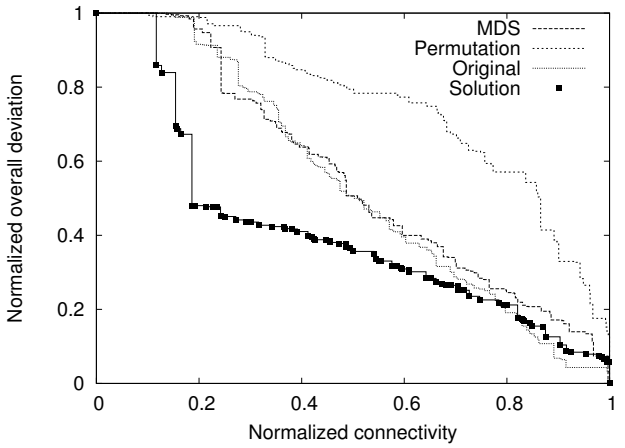


Figure 3: Comparison of the three different null models on a two-dimensional four cluster data set. It can be seen that the original null model (based on feature vectors), and the MDS-based null model result in highly similar control fronts. The reference front obtained with the permutation model, in contrast, is convex and unrealistic.

data item as the current reference point, and then updates the position of every other data item j in each dimension $c \in [1, 10]$ by means of the following gradient descent rule:

$$p_{jc} \leftarrow p_{jc} - \frac{lr \cdot (d(i,j) - d_e(i,j))}{d_e(i,j)} \cdot (p_{jc} - p_{ic}).$$

Here $lr = 0.05$ is the learning rate, $d(i,j)$ is the dissimilarity between data items i and j, and $d_e(i,j)$ is the Euclidean distance of their current positions in target space. In our experiments, the algorithm is iterated for $10 \times N$ iterations.

Once the embedding of data items has been generated, each data item can be described by its 10-dimensional position in target space. Hence, MOCK's standard null model for feature-based data becomes applicable: the correlation matrix of all position vectors is subjected to a principal component analysis[3] and the control data is generated in eigenspace.

4 Experimental setup

The following experimental section aims to investigate three main issues: the impact of the use of medoids, the efficiency of the proposed sampling scheme, and the performance of the modified null model. For this purpose, three implementations of MOCK are compared: these are, the standard feature-based version of MOCK (as described in [8]), and two versions of MOCK around medoids, one of them using the exact computation of the medoids, MOCK-am (exact), and the other one using the proposed sampling scheme, MOCK-am.

The three algorithms are run across a range of data sets of different size (several hundred to several thousand data

[3]Note that the rank of this matrix is much smaller than that of the one encountered for metric multidimensional scaling.

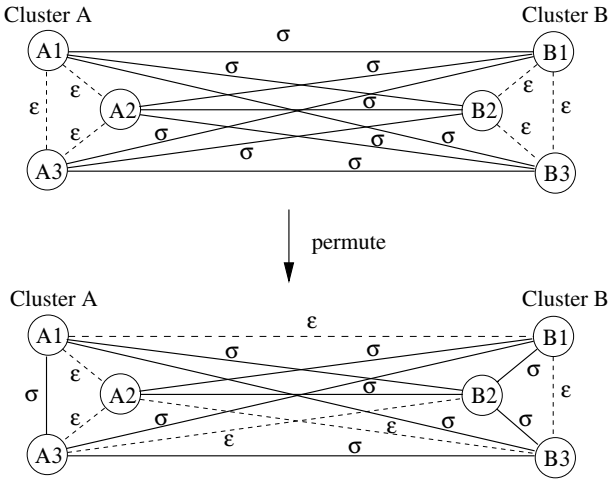


Figure 4: Permutation of the dissimilarity matrix. Shown is a data set containing two clusters of three data items each. The data items in a cluster are very close (indicated by a distance value of ϵ) and those in different clusters are very distant (indicated by a distance value of σ), with $\sigma \gg \epsilon$. After the symmetry-preserving permutation of the distance matrix, the triangular inequality is clearly violated. For example $d(A1, A2) + d(A2, A3) < d(A1, A3)$.

items), different dimensionality (2, 10, 50 and 100), and with different numbers of clusters (4, 10, 20 and 40) of arbitrary orientation. The data sets are denoted as Dc-kc, where D denotes the dimensionality of the data set, and k denotes the number of clusters. For each type of configuration of D and k, 10 different instances are generated. Importantly, these data sets are originally feature-based, but the two versions of MOCK-around-medoids are provided the dissimilarity information only. More details about the data sets, and a comparison of the standard version of MOCK to three established single-objective clustering methods across the same set of data sets can be found in [8]. The data sets are available at [4].

4.1 Parameter settings for MOCK

MOCK-around-medoids can be run with the same parameter settings as recommended for MOCK (see [8]). In our experiments, parameter settings (given in Table 1) are identical for all three versions of MOCK and are kept constant over all experiments.

4.2 Evaluation

Clustering quality is evaluated using the Adjusted Rand Index, an external measure of clustering quality, which is is a generalization of the Rand Index.

The Rand Indices are based on counting the number of pair-wise co-assignments of data items. The Adjusted Rand Index additionally introduces a statistically induced normalization in order to yield values close to 0 for random partitions. Using a representation based on the contingency table defined by two partitionings U and V, (one being the known correct classification and one being the partition under evaluation), the Adjusted Rand Index [9] is given as

$$R(U,V) = \frac{\sum_{lk}\binom{n_{lk}}{2} - \left[\sum_l \binom{n_{l\cdot}}{2} \cdot \sum_k \binom{n_{\cdot k}}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_l \binom{n_{l\cdot}}{2} + \sum_k \binom{n_{\cdot k}}{2}\right] - \left[\sum_l \binom{n_{l\cdot}}{2} \cdot \sum_k \binom{n_{\cdot k}}{2}\right] / \binom{n}{2}},$$

where n_{lk} denotes the number of data items that have been assigned to both cluster l and cluster k, where $l \in U$ and $k \in V$. It takes values in the interval $[0, 1]$ and is to be maximized.

In Section 6, we additionally use cluster Purity, another external measure, which is defined as the percentage of a cluster occupied by the predominant data type (according to the known class labels $t \in T$:

$$P(C_k) = \sum_{C_k \in C} \max_{t \in T} \frac{N_{tk}}{N_k},$$

where N_k is the size of the cluster C_k, and N_{tk} is the number of elements of class t within this class. The Purity $P(C)$ of an entire partitioning is then computed as the mean cluster Purity over all clusters. It is limited to the range $[0, 1]$ and is to be maximized.

Table 1: Parameter settings for MOCK, where N is data set size. For details see [8].

Parameter	*setting*
Number of generations	500
External population size	1000
Internal population size	10
#(Initial solutions) $fsize$	100
Initialization	Minimum spanning tree and k-means ($L = 10$)
Mutation type	L nearest neighbours ($L = 10$)
Biased mutation rate p_m	$p_m = \frac{1}{N} + (\frac{l}{N})^2$
Recombination	Uniform crossover
Recombination rate p_c	0.7
Objective functions	Overall deviation and connectivity ($L = 10$)
#(Reference distributions)	1

5 Experimental results

Table 2 and Table 3 show the results obtained by the individual algorithms. Runtimes are given in Table 4. The results reveal that there is very little performance difference between all three versions of MOCK (much higher performance differences were observed in our comparison to average link, single link and k-means [8]). This is true both with respect to the algorithms' performance at generating high quality solutions (see Table 2) and their performance at identifying these solutions from the Pareto front (see Table 3). Evidently, the replacement of centroids by medoids does not affect clustering quality. Also, the use of a sampling scheme for medoid determination effects only minor differences in clustering quality. Finally, the proposed null model seems to approximate closely the quality of the null model for feature-based data.

Table 2: Number of clusters k and Adjusted Rand Index of the best solution generated by MOCK, MOCK-am and MOCK-am (exact). Values for all algorithms are averages over 21 $\times$ 10 runs (that is, 21 runs per instance, and there are 10 instances per configuration). Standard deviations are small in all cases. To emphasize the small differences between the algorithms, the last column shows the difference between the largest and smallest mean value of the Adjusted Rand Index, δ_{max}.

Problem	MOCK		MOCK-am		MOCK-am (exact)		δ_{max}
	k	Rand Index	k	Rand Index	k	Rand Index	
2d-4c	4.06667	0.987992	4.09048	0.986113	4.07143	0.986285	0.001879
10d-4c	4.12381	0.995501	4.17143	0.994385	4.33333	0.994346	0.001155
50d-4c	4	1	4	1	4	1	0
100d-4c	4	1	4	1	4.00476	0.999922	0.000078
2d-10c	10.7143	0.933782	10.1905	0.934426	10.0762	0.935702	0.00192
10d-10c	13.5952	0.961277	14.5476	0.952446	13.8333	0.954041	0.008831
50d-10c	10.0238	0.997742	10.0048	0.997657	10.0048	0.997926	0.000269
100d-10c	10.0286	0.99976	10.0143	0.998886	10.0048	0.999116	0.000874
2d-20c	20.081	0.938703	20.0381	0.937385	19.819	0.93689	0.001813
10d-20c	20.4905	0.996973	20.5381	0.9963	20.5381	0.996222	0.000751
50d-20c	21.8952	0.89365	21.0429	0.908075	20.9143	0.907499	0.014425
100d-20c	20.9476	0.883184	20.4143	0.897921	20.181	0.906481	0.023297
2d-40c	42.4524	0.859737	38.1286	0.804444	38.0714	0.809926	0.055293
10d-40c	43.6333	0.987919	43.019	0.985108	43.4048	0.985541	0.002811
50d-40c	46.8619	0.808289	45.7571	0.834114	44.919	0.841328	0.033039
100d-40c	47.5762	0.791701	45.419	0.845056	44.8667	0.857862	0.066161

Table 3: Number of clusters k and Adjusted Rand Index of the best local maxima, and number of local maxima $\#lmax$ in a plot of the attainment scores. Values are averages as explained in Table 4. Standard deviations are small in all cases. To emphasize the small differences between the algorithms, the last column shows the difference between the largest and smallest mean value of the Adjusted Rand Index, δ_{max}.

Problem	MOCK			MOCK-am			MOCK-am (exact)			δ_{max}
	k	Rand Index	$\#lmax$	k	Rand Index	$\#lmax$	k	Rand Index	$\#lmax$	
2d-4c	4.06667	0.959832	10.4	4.35714	0.93258	8.64286	4.00952	0.934438	8.8	0.027252
10d-4c	4.17143	0.993823	10.3571	4.58571	0.988375	9.1381	4.50476	0.990262	10.119	0.005448
50d-4c	4.02857	0.978285	5.59524	4.27619	0.960199	4.53333	4.10952	0.970708	4.38571	0.018086
100d-4c	4.53333	0.93326	5.66667	4.40952	0.949157	4.6381	4.31905	0.959383	4.7	0.026123
2d-10c	10.7714	0.913508	11.1429	10.6476	0.904715	7.29048	15.5619	0.944748	7.93333	0.040033
10d-10c	14.3381	0.954993	14.1143	16.1524	0.94162	7.5619	14.25	0.943825	7.75	0.013373
50d-10c	11.3095	0.948293	6.98571	11.0048	0.957499	4.66667	11.219	0.952947	4.41429	0.009206
100d-10c	12.7095	0.903848	5.89048	11.519	0.94725	4.20476	12.3381	0.920303	3.57619	0.043402
2d-20c	20.581	0.913725	7.99048	20.6048	0.900547	6.32857	20.2429	0.905306	6.58571	0.013178
10d-20c	21.5667	0.98437	7.4381	21.1905	0.984628	5.51905	21.5238	0.983934	5.75238	0.000694
50d-20c	22.0286	0.873543	8.78095	21.6905	0.886708	7.8381	21.3286	0.890048	6.92381	0.016505
100d-20c	21.6857	0.857716	8.7	21.4714	0.864728	7.31429	20.9762	0.881586	6.49524	0.02387
2d-40c	42.6476	0.847592	11.1952	35.8857	0.784563	8.85238	36.2143	0.788858	8.62381	0.063029
10d-40c	44.2333	0.979675	7.74286	40.4143	0.956166	4.26667	38.7429	0.940302	3.8381	0.039373
50d-40c	48.181	0.790105	11.5905	41.4524	0.791553	9.45238	41.8048	0.801962	9.14762	0.011857
100d-40c	47.8952	0.770879	11.3905	42.6238	0.807944	9.50476	41.8333	0.816679	9.31429	0.0458

Table 4: Size of the problem N, and the corresponding runtimes (in seconds) for the three different versions of MOCK (averages over all instances/runs). The fastest performer from this sample is highlighted in bold. The time complexity of the original version of MOCK increases linearly with the size of the data set and the dimensionality. The runtime of MOCK-am is independent of dimensionality, but for MOCK-am (exact) it is quadratic in the size of the data set.

Problem	N	MOCK	MOCK-am	MOCK-am (exact)
2d-4c	1253.5	**53.4**	120.429	1022.27
10d-4c	1108.3	**67.4**	107.705	1143.5
50d-4c	1045.2	112.2	**89.2238**	818.386
100d-4c	1117.5	191.6	**96.7857**	893.876
2d-10c	3120.2	**162.5**	535.733	7300.68
10d-10c	2955.9	**241.2**	533.41	10996.4
50d-10c	2796.4	**402.5**	483.238	7594.29
100d-10c	2773.9	668.6	**507.405**	7686.06
2d-20c	1235.7	**50.1**	106.943	813.19
10d-20c	1171.9	**71**	111.205	1076.53
50d-20c	1171.1	132.1	**99.8571**	1412.52
100d-20c	1124.9	208.3	**96.7476**	1266.8
2d-40c	2268.1	**104.4**	290.914	3510.94
10d-40c	2355.5	**154.4**	310.224	5297.56
50d-40c	2159.7	**273.9**	289.333	5767.57
100d-40c	2106.2	451.8	**279.114**	5655.16

Table 5: Randomly selected computer files.

type	.cc	.tex	.bib	.ps	.pdf	Total
#files	223	271	60	102	255	911

As far as runtime is concerned, the exact version of MOCK-am is very slow, as expected. The sampling version of MOCK-am is much more efficient, and, for high-dimensional data, outperforms the feature-based version.

6 Application: computer file clustering

In many clustering applications, there is no simple or obvious way to describe the data to be clustered in terms of feature vectors. Describing the dissimilarities between the individual data items can often be easier. Moreover, when objects are so difficult to describe that it is even difficult to devise bespoke measures of dissimilarity, there is one measure which can always be used: the universal similarity metric (USM) [15], which is based on an entirely general information-theoretic concept. This metric is formally defined as:

$$d(i,j) = \frac{\max\{K(i|j), K(j|i)\}}{\max\{K(i), K(j)\}}$$

where i and j are two data items, and $K(.)$ is the Kolmogorov complexity. The metric measures the dissimilarity of two objects as: the minimum space needed to describe one of them, given the other, as a fraction of the 'larger' object.

Because this metric can be used to cluster any type of data, without need for special knowledge of its content, it has already been used, in approximated form, to tackle difficult clustering problems in bioinformatics, such as three-dimensional protein structure 'alignment' [13]. It is thus interesting for us to use data derived from applying a similar approach in order to test MOCK-am further. To this end, we construct our own USM-derived dissimilarity matrix from data where we can be sure of the correct class labels.

6.1 Constructing the dissimilarity matrix

The objects to cluster are computer files from the second author's computer — in this sense the data is real-world, rather than artificial. A script was written to randomly select files of one of the following five types: .cc, .tex, .bib, .ps, and .pdf, from anywhere on the computer (using the Linux 'locate' function). The numbers of files making up the data set are shown in Table 5.

Beyond this, the approach follows that taken in [13]. To approximate the Kolmogorov complexity of an object, its zipped size is taken.[4] The computation of the approximate USM metric for these files is:

$$d(i,j) = \frac{\mathrm{gzip}(\mathrm{cat}(i,j)) - \min(\mathrm{gzip}(i), \mathrm{gzip}(j))}{\max(\mathrm{gzip}(i), \mathrm{gzip}(j))}$$

where $\mathrm{gzip}(.)$ denotes the size of the zipped file, and $\mathrm{cat}(.,.)$ is the concatenation of the two files. Notice how the Kolmogorov complexity of one file given the other is approximated using a simple concatenation and subtraction procedure.

The 911×911 dissimilarity matrix obtained is available from [4]. The diagonal entries of the matrix are not quite zero because of the imperfection of gzip as an approximation of the true Kolmogorov complexity. Similarly, some values in the matrix are slightly above 1. However, almost all of the off-diagonal entries lie in the range 0.75–1.05.

6.2 Comparison to average-link

As a comparison algorithm, we selected average-link agglomerative clustering. There are two reasons for this choice: 1. because it was the most competitive algorithm with MOCK in our recent study [8], and 2. because it is a typical algorithm used in similarity-based clustering, very popular in bioinformatics, and the one used in [13].

Agglomerative clustering algorithms are hierarchical algorithms, which start with the finest partitioning possible (that is, singletons) and, in each iteration, merge the two least distant clusters. They terminate when the target number of clusters has been obtained. Alternatively, the entire dendrogram can be generated and be cut at a later point. Different versions of agglomerative clustering differ in the linkage metric used to compute 'distance' between clusters. For the linkage metric of average link, the distance between two clusters C_i and C_j is computed as the average dissimilarity between all possible pairs of data elements i and j with $i \in C_i$ and $j \in C_j$.

The settings used for MOCK-am were as given in Section 4.

6.3 Results

Results for both algorithms are given in Table 6. The Adjusted Rand Index scores for the two algorithms show that MOCK-am handles this tough data set much better than average link. However, the scores of 0.671992 and 0.641006 might seem to suggest that MOCK-am is still not doing very well on an absolute scale. On closer inspection it can be seen that the low score is partly due to the fact that MOCK-am prefers solutions with an average of around 15 clusters, which has a big effect on the reward from the Adjusted Rand Index. When these clusters are measured for their Purity, however, values of 0.946 and 0.944 result. This shows that MOCK-am is identifying homogeneous groups of files by type; it is only that all files of one type do not necessarily 'fit together'. In other words, there is a hierarchy to the files, as one might expect from files that are often related to each other by their subject as well as their type.

7 Conclusion

The adaptation of our existing multiobjective clustering algorithm, MOCK, in order to deal with dissimilarity data has been described. In a comparison to the feature-based ver-

[4] We used Gnu gzip for this.

Table 6: Results for MOCK-am and average link agglomerative clustering on the real data set (averages over 21 runs). For each algorithm, the first row describes the best (best in terms of the Adjusted Rand Index) solution found. For MOCK-am, the second row describes the best local maximum in the attainment plot. For average link agglomerative clustering, the second row describes the solution obtained for a number of clusters $k = 15$ in order to match the k selected by MOCK-am. This is necessary because there is a tendency for Purity to increase as k increases. Therefore, to make a sound comparison the same k should be used.

MOCK-am			Average link		
k	Rand Index	Purity	k	Rand Index	Purity
14.28	0.671992	0.946	29.52	0.283362	0.835173
15.04	0.641006	0.944	15	0.282964	0.801601

sion, very little drop in performance was seen when it was applied to data originally in feature-vector form. The time complexity of the new algorithm is also favourable for data sets with high-dimensional feature vectors. Perhaps more importantly, MOCK-am can be used on dissimilarity-based data from applications where feature vectors are not available, as shown by the clustering of computer files via the universal similarity metric.

Acknowledgements: Thanks to Pablo Moscato for sending a dissimilarity-based data set, and thereby triggering this work. JH gratefully acknowledges support of a scholarship from the Gottlieb Daimler- and Karl Benz-Foundation, Germany. JK is supported by a David Phillips Fellowship from the Biotechnology and Biological Sciences Research Council (BBSRC), UK.

Bibliography

[1] S. F. Altschul, T. L. Madden, A. A. Schaeffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[2] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 283–290, San Francisco, California, 2001. Morgan Kaufmann Publishers.

[3] T. F. Cox and M. A. A. Cox. *Multidimensional Scaling*. Chapman and Hall, 2001.

[4] J. Handl and J. Knowles. Supporting material. http://dbk.ch.umist.ac.uk/handl/mock/ .

[5] J. Handl and J. Knowles. Evolutionary multiobjective clustering. In *Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature*, pages 1081–1091. Springer-Verlag, 2004.

[6] J. Handl and J. Knowles. Multiobjective clustering with automatic determination of the number of clusters. Technical Report TR-COMPSYSBIO-2004-02, UMIST, Manchester, UK, 2004. Available from http://dbk.ch.umist.ac.uk/handl/mock/ .

[7] J. Handl and J. Knowles. Exploiting the trade-off: the benefits of multiple objectives in data clustering. In *Proceedings of the Third International Conference on Evolutionary Multicriterion Optimization*, pages 547–560. Springer-Verlag, 2005.

[8] J. Handl and J. Knowles. Improvements to the scalability of multiobjective clustering. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005. To appear.

[9] A. Hubert. Comparing partitions. *Journal of Classification*, 2:193–198, 1985.

[10] L. Hubert. Approximate evaluation techniques for the single-link and complete-link hierarchical clustering procedures. *Journal of the American Statistical Association*, 69:698–704, 1974.

[11] L. J. Hubert and F. B. Baker. *An Empirical Comparison of Baseline Models for Goodness-of-Fit in r-Diameter Hierarchical Clustering*. Academic Press, Inc, 1977.

[12] L. Kaufman and P. J. Rousseeu. *Finding Groups in Data*. John Wiley and Sons, 1989.

[13] N. Krasnogor and D. A. Pelta. Measuring the similarity of protein structures by means of the universal similarity metric. *Bioinformatics*, 20(7):1015–1021, 2004.

[14] J. B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage, 1978.

[15] M. Li, J. H. Badger, X. Chen, S. Kwon, P. Kearney, and H. Zhang. An information-based sequence distance and its application to whole mitochondrial genome phylogeny. *Bioinformatics*, 17(2):149–154, 2001.

[16] R. F. Ling. A probability theory of cluster analysis. *Journal of the American Statistical Association*, 68:159–169, 1973.

[17] L. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297. University of California Press, 1967.

[18] Y.-J. Park and M.-S. Song. A genetic algorithm for clustering problems. In *Proceedings of the Third Annual Conference on Genetic Programming*, pages 568–575. Morgan Kaufmann, 1998.

[19] W. S. Sarle. Cubic clustering criterion. Technical report, SAS Technical Report A-108, Cary, NC: SAS Institute Inc, 1983.

[20] A. Strehl and J. Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *Journal on Machine Learning Research*, 3:583–617, 2002.

[21] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a dataset via the Gap statistic. Technical report, Stanford University, 2000.