Multiobjective clustering with automatic determination of the number of clusters

Julia Handl Joshua Knowles

Technical report TR-COMPSYSBIO-2004-02
UMIST, Manchester
http://dbk.ch.umist.ac.uk/handl/mock/

August, 2004

Abstract

We propose a novel approach to data clustering based on the *explicit* optimization of a partitioning with respect to *multiple*, complementary clustering objectives.[1] We argue that this approach may be more robust to the variety of cluster structures found in different data sets, and may be able to identify certain cluster structures that cannot be discovered by other methods. To demonstrate the approach, we describe an algorithm, MOCK (multiobjective clustering with automatic k-determination), which uses a multiobjective evolutionary algorithm (MOEA) to perform the optimization. MOCK uses two complementary objectives based on cluster compactness and connectedness, and returns a set of different trade-off partitionings over a range of different cluster numbers, k. The shape of the trade-off curve (Pareto front) reveals valuable information about the cluster structure, such that it is possible to derive a fully automatic procedure that estimates the best partitioning, hence also determining k automatically. A series of experiments using a suite of 21 real and synthetic data sets reveal that MOCK performs robustly, *with no need of parameter tuning or other choices being necessary*: precisely the same setup is used for all problems. Compared to three traditional clustering algorithms, an advanced ensemble technique, and a statistical method for determining the number of clusters, MOCK is the most robust.

Keywords: Clustering, multiobjective optimization, evolutionary algorithms, determining number of clusters, cluster ensembes, Gap statistic

[1] NB: preliminary results on multiobjective clustering have been previously published by us in PPSN VIII [24].

1 Introduction

Clustering is commonly defined as the task of finding natural groups within a data set such that data items within the same group are more similar than those within different groups. This is an intuitive but rather 'loose' concept, and it remains quite difficult to realize in general practice. Evidently, one reason for the difficulty is that for many data sets, no unambiguous partitioning of the data exists, or can be established, even by humans. But even in cases where an unambiguous partitioning of the data *is* possible, clustering algorithms can drastically fail. This is because most existing clustering techniques rely on estimating the quality of a particular partitioning by means of just one *internal evaluation function*, an objective function that measures intrinsic properties of a partitioning, such as the spatial separation between clusters or the compactness of clusters. Hence, the internal evaluation function is assumed to reflect the quality of the partitioning reliably, an assumption that may be violated for certain data sets. A further fundamental difficulty of clustering is the determination of the number of clusters in the data set. Most existing algorithms require this number to be provided as a parameter, which is a major problem in a setting where the structure of the data is completely unknown. While a number of different approaches to determine the number of clusters automatically have been proposed [14, 20, 37, 50], no reliable method exists to date.

In this paper, we argue that the use of multiobjective optimization may provide a means to overcome some of the limitations of current clustering algorithms. Given that many objective functions for clustering are complementary, the simultaneous optimization of several of these objectives may lead to higher quality solutions and an improved robustness towards different data properties. Here, we demonstrate this idea using a new evolutionary algorithm-based approach, MOCK, which we describe in detail and make publicly available. We also demonstrate the ability of this algorithm to automatically determine the number of clusters in a data set.

The remainder of this paper is organized as follows. Section 2 briefly summarizes related work on clustering and evolutionary algorithms. This is followed by a detailed description of MOCK in Section 3. We then proceed to the evaluation of the algorithm: Section 4 describes the experimental setup employed and Section 5 presents and discusses results. Finally, Section 6 concludes.

2 Related work

Clustering problems arise in a variety of disciplines ranging from sociology and psychology, to commerce, biology and computer science, and algorithms for tackling them continue to be the subject of active research. Consequently, there exists a multitude of clustering methods, which differ not only in the principles of the algorithm used (which of course determine runtime behaviour and scalability), but also in many of their most basic properties, such as the data handled (numerical vs. categorical and proximity data), assumptions on the shape of the clusters (e.g. spherically shaped), the form of the final partitioning (hard vs. fuzzy assignments) or the parameters that have to be provided (e.g. the correct or desired number of clusters). In this paper, we restrict our attention to clustering problems involving numerical data and valid clustering solutions as defined by a strict (i.e. non-fuzzy) partitioning of the data set — thus appropriate clustering algorithms are also a restricted set. For a more extensive survey of clustering problems and algorithms the reader is referred to Jain et al. [27].

Traditional classifications of clustering algorithms primarily distinguish between *hierarchical*, *partitioning* and *density-based* methods. Here, we will discuss a somewhat different categorization that is based on the clustering criterion (implicitly or explicitly) optimized by the algorithm. With regard to this, existing clustering algorithms fall into three major groups. First, algorithms striving for *compact* clusters, a concept which is generally implemented by keeping intra-cluster variation (i.e. variation between same-cluster data items or between data items and cluster representatives) small. This category includes algorithms like k-means [33], average-link

agglomerative clustering [54] or model-based clustering approaches [11, 36]. The resulting methods tend to be very effective for spherical and/or well-separated clusters, but they may fail for more complicated cluster structures.

Methods based on a concept of *connectedness* make up the second group. They employ a more local concept of clustering based on the idea that neighbouring data items should share the same cluster. Algorithms implementing this principle are density-based methods [2, 15] and methods like single-link agglomerative clustering [54]. All of these are well-suited to detect clusters of arbitrary shapes, however, they can lack robustness when there is little spatial separation between clusters.

Objectives based on *spatial separation* can be identified as the underlying criteria for a further class of clustering algorithm. However, spatial separation on its own is a criterion that gives little guidance during the clustering process and can easily lead to trivial solutions (e.g. classifying outliers as individual clusters, but merging the bulk of data items into one cluster). It is therefore usually combined with other objectives, most notably measures of compactness or 'balancedness' of cluster sizes. Unlike the first two groups, the resulting clustering objectives (e.g. Dunn Index, Davies-Bouldin Index [23]) have not been implicitly optimized, but can be tackled by an explicit heuristic optimization method.

Examples of heuristic optimization techniques include simulated annealing [30], tabu search [22], ant-colony optimization [13] and evolutionary algorithms [25]. Of these, evolutionary algorithms (EAs) have been the most frequently used for clustering. However, previous research in this respect has been limited to the single objective case: criteria based on cluster compactness have been the objectives most commonly employed, as these measures provide smooth incremental guidance in all parts of the search space – much better so than clustering objectives based on spatial separation between clusters.

A variety of different EA representations for clustering solutions have been explored in the literature, ranging from a straightforward encoding (with the ith gene coding for the cluster membership of the ith data item), to more complex representations, such as Falkenauer's grouping EA [17]. However, none of these direct encodings significantly reduces the extent of the clustering search space, such that the derivation of operators to explore this space efficiently remains crucial. For this reason, several researchers have chosen to use a more indirect approach that borrows from the popular k-means algorithm: the representation codes for cluster centroid/medoids only; each data item is subsequently assigned to the closest cluster representative [28, 35, 40].

Genuine hybridizations between EAs and k-means have also been introduced, and are of particular interest with respect to feature selection for unsupervised classification. In a number of papers EAs have been used to evolve the features serving as the input for the k-means algorithm. k-means' clustering solutions are then evaluated and the resulting objective values fed back to the EA. In this context, the use of multiple criteria is essential to correctly balance the number of features and solution quality – in order to identify possible trade-offs between feature set and solution quality, multiobjective evolutionary algorithms (MOEAs) have therefore repeatedly been employed [18, 29, 39].

In contrast to their application for feature selection, MOEAs have not previously been applied for the actual clustering task itself. This is despite the general agreement that clustering objectives can be conflicting or complementary, and that no single clustering objective can deal with every conceivable cluster structure [16, 31]. To date, attempts to deal with this problem have focused on the *retrospective* combination of different clustering results by means of ensemble methods [32, 38, 52, 53, 51, 48]. In order to construct clustering ensembles, different clustering results are retrieved by repeatedly running the same algorithm (using different initializations, bootstrapping or a varying number of clusters) or several complementary methods (e.g. agglomerative algorithms based on diverse linkage criteria such as single link and average link). The resulting solutions are then combined into an ensemble clustering using graph-based approaches [48], expectation maximization [51] or co-association methods [51].

Results reported in the literature demonstrate that clustering ensembles are often more robust and yield higher quality results than individual clustering methods, indicating that the combination of several clustering objectives is favourable. However, ensemble methods do not fully exploit the potential of using several objectives: as they are limited to the *a posteriori* integration of the solutions returned by the individual algorithms, they cannot explore trade-off solutions *during* the clustering process. Individual clusters that cannot be (approximately) detected by any one of the member algorithms on its own are therefore likely to be missed. We aim to overcome this limitation by tackling clustering as a truly multiobjective optimization problem. Note that the term 'multiobjective clustering' has been previously employed in the above cited literature [38]. However, the corresponding algorithm relies (like ensemble techniques) on the *retrospective* combination of clustering results, and, as such, fundamentally differs from our approach.

Before going on to describe MOCK, we note here that much recent literature in clustering and pattern recognition more generally, is directed towards applications in the biological sciences. In many of these applications, the quality of the clustering is of prime importance and the time taken to obtain it is at most secondary. For example, in clustering of microarray data, which is a very important application area, designing experiments, performing them, and subsequent collection of the data typically takes of the order of days or weeks, thus it is of little consequence whether an algorithm takes one second or one minute to run. It is however, most important to cluster the data as effectively as possible. It is in this kind of context (in a broad sense) that we would expect MOCK to be most appropriate and not so for applications where computational efficiency must be a foremost consideration.

The consideration of multiple objectives, at the heart of MOCK, is also relevant to the use of clustering in biology (though more broadly, too) for another reason: biologists will often *prefer* to be offered several alternative partitionings, rather than just one, because these offer different interpretations of the data, which can be actually tested through further experiments. As clustering is itself an ill-posed problem [31], returning alternatives is not necessarily a sign of failure, but merely accords with the fact that different compromises do exist. We do note however that returning too many alternatives can also be undesirable and it is for this reason that we also derive a fully automatic version of MOCK which singles out a 'best' solution; this also allows unbiased evaluation of MOCK.

3 MOCK

3.1 Multiobjective optimization terminology

A general (unconstrained) multiobjective optimization problem (MOP) can be defined as:

$$\begin{aligned} \text{'minimize' } \mathbf{z} = \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})), \\ \text{with } \mathbf{x} = (x_1, x_2, \ldots, x_n) \in X, \end{aligned}$$

where $\mathbf{x}$ is an n-dimensional decision vector or solution, and X is the decision space, i.e. the set of all expressible solutions. The objective function $\mathbf{f}(\mathbf{x})$ maps X into $\mathbb{R}^m$, where $m \geq 2$ is the number of objectives. The vector $\mathbf{z} = \mathbf{f}(\mathbf{x})$ is an objective vector or point. The image of X in objective space is the set of all attainable points, Z. The term 'minimize' appears in quotation marks because, in general, there does not exist a single solution that is minimal on all objectives[2]. Instead, there is a partial ordering of points in objective space:

$$\forall \mathbf{y}, \mathbf{z} \in Z, \quad \mathbf{y} \leq \mathbf{z} \iff \forall i \in 1..m, \quad y_i \leq z_i \wedge \exists j \in 1..m, \quad y_j < z_j.$$

[2]N.B. we assume minimization without loss of generality, since any objective which is to be maximized can be simply multiplied by -1.

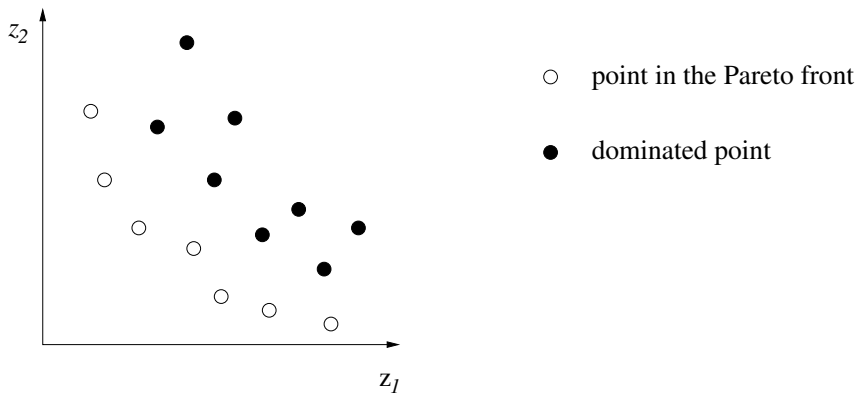

Figure 1: A set of points in the objective space, divided into the Pareto front (nondominated points) and the dominated points.

In the above, $\mathbf{y}$ is said to dominate $\mathbf{z}$. Thus, there is usually a *set* of optimal solutions $X^* \subseteq X$, known as the *Pareto optimal set*:

$$X^* = \{\mathbf{x}^* \in X \mid \nexists\, \mathbf{x} \in X, \quad \mathbf{f}(\mathbf{x}) \leq \mathbf{f}(\mathbf{x}^*)\}. \tag{1}$$

The points in objective space corresponding to the Pareto optima are termed *nondominated* and when plotted, form the *Pareto front*. A set of points and its corresponding Pareto front is plotted in Figure 1.

Similarly, for any set of points $Y \subseteq Z$, an (internally) nondominated front $Y^* \subseteq Y$ exists, such that

$$\forall\, \mathbf{y}^* \in Y^* \quad \nexists\, \mathbf{y} \in Y \text{ such that } \mathbf{y} \leq \mathbf{y}^*. \tag{2}$$

Note that an internally nondominated front may or not correspond to the (true) Pareto front (of the wider objective space, Z). Approximations to the Pareto front, generated by a heuristic search process, will be such *internally* nondominated fronts with no guarantee of optimality.

3.2 Multiobjective Clustering

In recent years there has been a growing interest in developing and applying evolutionary algorithms (EAs) in multiobjective optimization (see, e.g. [10, 5]). EAs are easily adapted to multiobjective problems because their use of a population naturally enables the whole Pareto front to be approximated in a single algorithm run. Many different algorithms exist but most modern MOEAs are characterized by the use of: (1) some kind of 'niching' in the objective space [26], which encourages the population to spread out over the entire Pareto front; and (2) elitism to prevent the loss of nondominated solutions and encourage convergence towards the Pareto front. One example of such an MOEA, the Pareto envelope selection algorithm, version 2 (PESA-II), forms the basis of MOCK.

3.2.1 PESA-II

PESA-II [7, 6] is a well-known algorithm in the evolutionary multiobjective optimization literature, and has been used in comparison studies by several researchers. A high-level description is given in Algorithm 1. Two populations of solutions are maintained: an internal population, IP of fixed size, and an external population EP, of non-fixed but limited size. The purpose of EP is to *exploit* good solutions: to this end it implements elitism by maintaining a large and diverse set of nondominated solutions. The internal population's job is to *explore* new solutions, and achieves this by the standard EA processes of reproduction and variation (i.e., recombination

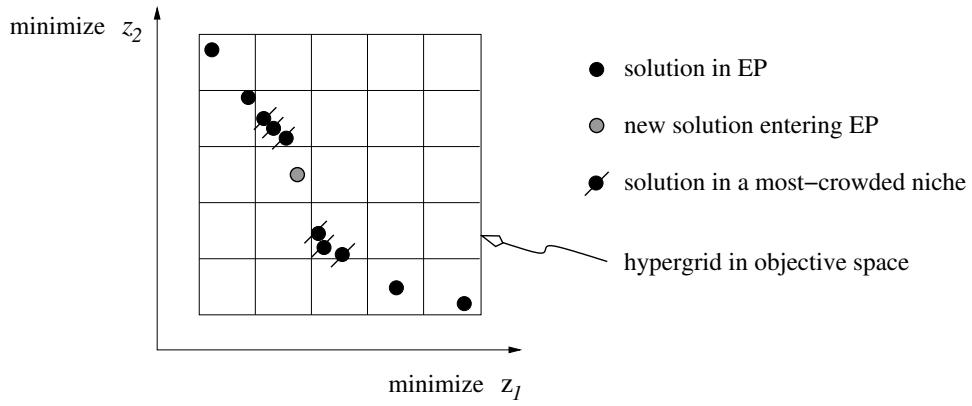

Figure 2: Nondominated solutions in EP as seen in objective space. A (hyper)grid which just covers them is used to place them into niches. Here, the size of EP is limited to 10 but a new solution enters because it will occupy a less crowded niche than the six crossed solutions shown. One of these, selected uniformly at random, would be ejected from EP to maintain its maximum size.

and mutation). Selection occurs at the interface between the two populations, primarily in the update of EP.

The solutions in EP are stored in 'niches', implemented as a hypergrid in the objective space (Figure 2). A tally of the number of solutions that occupy each niche is kept and this is used to encourage solutions to cover the whole objective space, rather than bunch together in one region. To this end, nondominated solutions that try to enter a full EP can only do so if they occupy a less crowded niche than some other solution (lines 36 and 37 of Algorithm 1). Moreover, when the internal population of each generation is constructed from EP (lines 9–12), they are selected uniformly from among the populated niches — thus highly populated niches do not contribute more solutions than less populated ones.

An important advantage of PESA-II is that this niching policy uses an adaptive range equalization and normalization of the objective function values. This means that difficult parameter tuning is avoided, and objective functions that have very different ranges can be readily used. PESA-II can also handle any number of objective functions. For further details on PESA-II, the reader is referred to [6].

3.2.2 Genetic representation and operators

To apply PESA-II to the clustering problem, one needs only to choose a suitable genetic encoding of a partitioning, one or more genetic variation operators (e.g. mutation and/or crossover), and two or more objective functions (we shall use two, described in 3.2.3).

For the encoding, we employ the locus-based adjacency representation proposed in [41]. In this graph-based representation (see Figures 3 and 6), 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. All data items belonging to the same subgraph are then assigned to one cluster. Note that, using a simple backtracking scheme, this decoding step can be done in linear time (see pseudo-code in Algorithm 2).

The locus-based adjacency encoding scheme has several major advantages for our application. Most importantly, there is no need to fix the number of clusters in advance, as it is automatically determined in the decoding step. Hence, we can evolve and compare solutions with different

Algorithm 1 PESA-II (high-level pseudocode)

```
 1: procedure PESA-II(ipszize, epmaxsize, p_m ∈ [0,1], p_c ∈ [0,1], #gens)
 2:     IP := ∅; EP := ∅
 3:     for each i in 1 to ipsize do                          /* INITIALIZATION */
 4:         s_i := initialize_solution(i)
 5:         evaluate(s_i)
 6:         UpdateEP(EP, s_i, epmaxsize)                  /* Procedure defined in line 30 */
 7:     end for
 8:     for gen in 1 to #gens do                              /* MAIN LOOP */
 9:         for each i in 1 to ipsize do
10:             select a populated niche n uniformly at random from EP
11:             select a solution s_i uniformly at random from n
12:             IP := IP ∪ {s_i}
13:         end for
14:         i := 0
15:         while i < ipsize do
16:             if random deviate R(0,1) < p_c then
17:                 s_i, s_{i+1} := crossover(s_i, s_{i+1})
18:             end if
19:             s_i := mutate(s_i, p_m); s_{i+1} := mutate(s_{i+1}, p_m)
20:             i := i + 2
21:         end while
22:         for each i in 1 to ipsize do
23:             evaluate(s_i)
24:             UpdateEP(EP, s_i, epmaxsize)
25:         end for
26:         IP := ∅
27:     end for
28:     return EP, a set of nondominated solutions
29: end procedure

30: procedure UpdateEP(EP, s_i, epmaxsize)          /* Update EP with solution s_i */
31:     if ∃s ∈ EP, s_i dominates s then
32:         EP := EP ∪ {s_i} \ {s ∈ EP, s_i dominates s}
33:     else if s_i is nondominated in EP then
34:         if EP < epmaxsize then
35:             EP := EP ∪ {s_i}
36:         else if ∃s ∈ EP, s_i is in a less crowded niche than s then
37:             EP := EP ∪ {s_i} \ {s, a solution from a most-crowded niche}
38:         end if
39:     end if
40:     update all niche counts
41: end procedure
```

numbers of clusters in just one run of the GA.

Furthermore, the representation is well-suited for the use with standard crossover-operators such as uniform, one-point or two-point crossover. In more traditional encodings for clustering these straightforward crossover operators are usually highly disruptive and therefore detrimental for the clustering process. In a link-based encoding, in contrast, they effortlessly implement merging and splitting operations on individual clusters, while maintaining the remainder of the

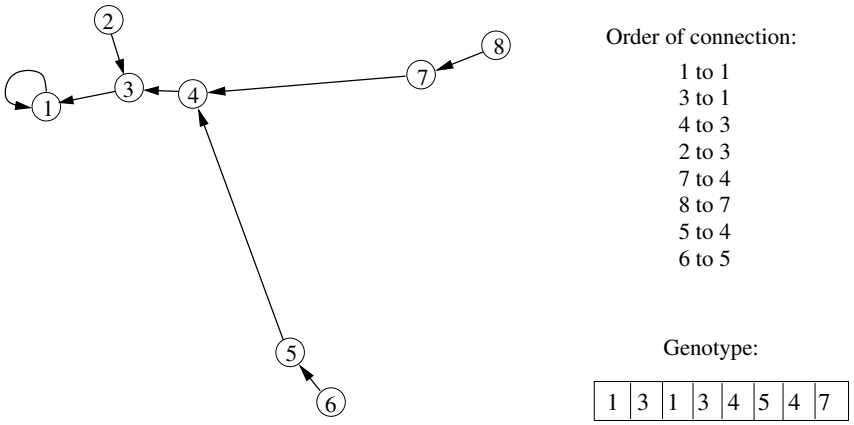

Figure 3: Construction of the minimum spanning tree and its genotype coding. The data item with label 1 is first connected to itself, then Prim's algorithm is used to connect the other items. In the genotype, each gene (i.e. position in the string) represents the respective data item, and its allele value represents the item it points to (e.g. gene 2 has allele value 3 because data item 2 points to data item 3). The genotype coding for the full MST (as shown) is used as the first individual in the EA population. For the initialization of other individuals, see Figure 6.

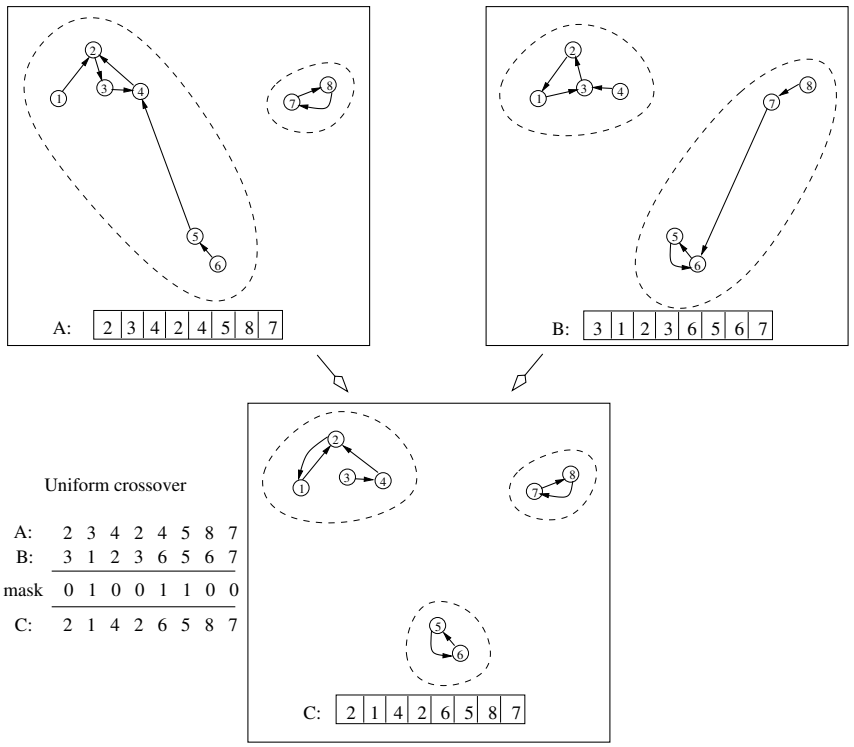

Figure 4: Two parent partitionings, their underlying forests, and their respective genotypes, A and B are shown. A standard uniform crossover of the genotypes yields the child C, which has inherited much of its structure from its parents, but differs from both of them.

partitioning. We choose the uniform crossover [49] in favour of one- or two-point because it is unbiased with respect to the ordering of genes and can generate any combination of alleles from the two parents (in a single crossover event) [56]. An example of the uniform crossover operating on the locus-based adjacency encoding is given in Figure 4. While the encoding results in a very large search space with N^N possible combinations, a suitable mutation operator can

Algorithm 2 Efficient decoding of the locus-based adjacency representation

```
1: procedure DECODE
2:     current_cluster := 1
3:     for each i in 1 to N do
4:         cluster_assignment_i := −1
5:     end for
6:     for each i in 1 to N do
7:         ctr := 1
8:         if cluster_assignment_i = −1 then
9:             cluster_assignment_i := current_cluster
10:            neighbour := g_i
11:            previous_ctr := i
12:            ctr := ctr + 1
13:            while cluster_assignment_neighbour = −1 do
14:                previous_ctr := neighbour
15:                cluster_assignment_neighbour := current_cluster
16:                neighbour := g_neighbour
17:                ctr := ctr + 1
18:            end while
19:            if cluster_assignment_neighbour ≠ current_cluster then
20:                ctr := ctr − 1
21:                while ctr >= 1 do
22:                    cluster_assignment_previous_ctr := cluster_assignment_neighbour
23:                    ctr := ctr − 1
24:                end while
25:            else
26:                current_cluster := current_cluster + 1
27:            end if
28:        end if
29:    end for
30:    number_of_clusters := current_cluster
31: end procedure
```

be employed to significantly reduce the size of the search space. We use a restricted *nearest neighbour mutation* where each data item can only be linked to one of its L nearest neighbour. Hence, $g_i \in \{nn_{i1}, \ldots, nn_{iL}\}$, where nn_{il} denotes the lth nearest neighbour of data item i. This reduces the extent of the search space to just L^N. Note, that the nearest neighbour list can be precomputed in the initialization phase of the algorithm.

Our initialization routine also exploits the link-based encoding and uses the minimum spanning tree (MST). For a given data set, we first compute the complete MST using Prim's algorithm [57]. The ith individual of the initial populations is then initialized by the MST with the $(i-1)$th largest links removed (see Figures 3 and 6). This initialization procedure efficiently computes the equivalent set of solutions that would be obtained from a single-link agglomerative algorithm, run *ipsize* times, with respectively, $k \in 1..ipsize$. Thus it provides the initial population with high quality solutions, each with different k, from which to begin search.

3.2.3 Objective functions

For the clustering objectives, we are interested in selecting optimization criteria that reflect fundamentally different aspects of a good clustering solution. From the groups identified in Section 1, we therefore select two types of complementary objectives: one based on compactness, the other one based on connectedness of clusters. We refrain to use a third objective based

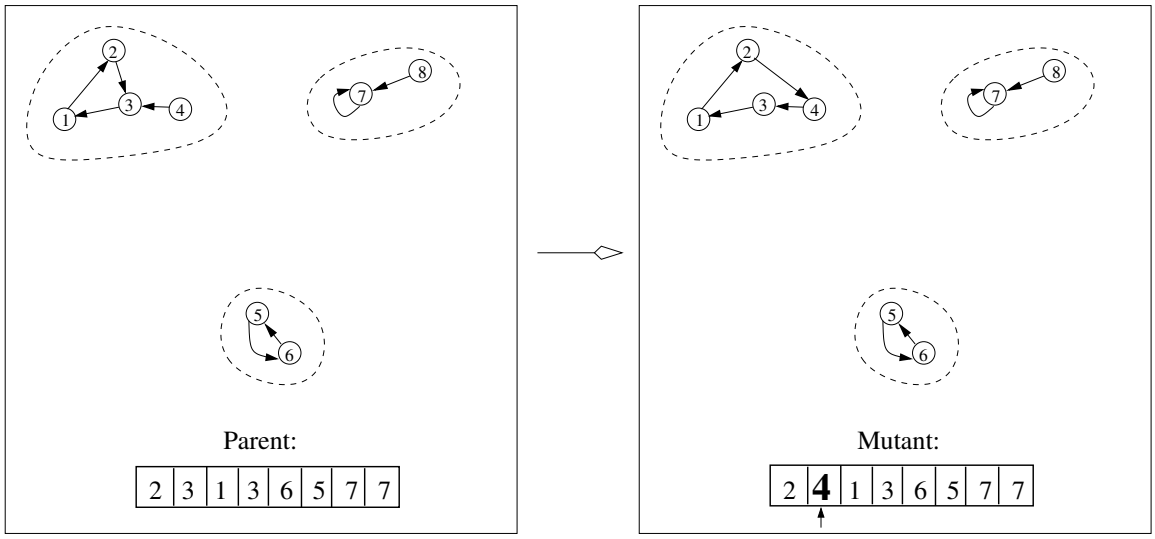

Figure 5: A mutation event in which one gene's allele value is changed. This mutation is *neutral* with respect to the clustering of the items. Notice that for a gene's mutation to be neutral, it is necessary (but not sufficient) that after the mutation, the gene points to a member of the same cluster. Thus, in general, for larger clusters, any individual gene's mutation is more likely to be neutral. Balanced against this effect is the fact that there are more genes that can undergo mutation in a larger cluster. Therefore, overall, smaller and larger clusters have a similar level of stability with respect to mutation — a desirable property for unbiased search.

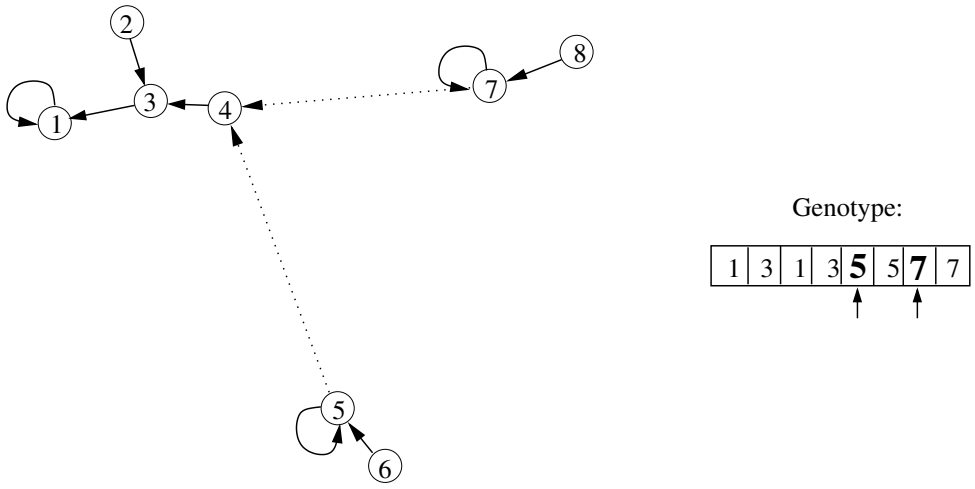

Figure 6: Initialization of the ith individual in the population: the $(i-1)$ longest links are removed from the MST individual. In the genotype, the corresponding genes are changed to link to themselves. Here, the third initial member of the population is shown.

on spatial separation, as the concept of spatial separation is intrinsic (opposite) to that of connectedness of clusters.

In order to express cluster compactness we compute the *overall deviation* of a partitioning. This is simply computed as the overall summed distances between data items and their corresponding cluster centre:

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

where C is the set of all clusters, μ_k is the centroid of cluster C_k and $\delta(.,.)$ is the chosen distance function (see Section 4.3.1). As an objective, overall deviation should be minimized. This criterion is similar to the popular criterion of intra-cluster variance, which squares the distance value $\delta(.,.)$ and is more strongly biased towards spherically shaped clusters.

As an objective reflecting cluster connectedness, we use a measure, connectivity, which evaluates the degree to which neighbouring data-points have been placed in the same cluster. It is computed as

$$Conn(C) = \sum_{i=1}^{N} \left(\sum_{j=1}^{L} x_{i,nn_i(j)} \right), \text{where } x_{r,s} = \begin{cases} \frac{1}{j} \text{ if } \nexists C_k : r, s \in C_k \\ 0 \text{ otherwise,} \end{cases}$$

$nn_i(j)$ is the jth nearest neighbour of datum i, and L is a parameter determining the number of neighbours that contribute to the connectivity measure. As an objective, connectivity should be minimized.

This measures is conceptually similar to the criterion of nearest-neighbour consistency introduced by Ding et al. [12]. The main difference is the use of a gradually decreasing penalty $\frac{1}{j}$, where Ding et al. use a constant; this gives more emphasis to the nearest neighbours, permits a finer distinction between the quality of clustering solutions and allows for the identification of clusters of sizes significantly smaller than L.

Overall deviation (like our mutation operator) requires the one-off computation of the nearest neighbour lists in the initialization phase. Subsequently, both objectives, overall deviation and connectedness, can be efficiently computed in linear time.

A further important aspect in the choice of these objective functions is their potential to balance each other's tendency to increase or decrease the number of clusters. While the objective value associated with overall deviation necessarily improves with an increasing number of clusters, the opposite is the case for connectivity. The interaction of the two is important in order to explore sensible parts of the solution space, and not to converge to trivial solutions (which would be N singleton clusters for overall deviation and only one cluster for connectivity).

3.3 Automatic determination of the number of clusters

One powerful advantage of our multiobjective clustering strategy is the fact that not just one solution, but a whole set of solutions is returned. These individual partitionings correspond to different trade-offs between the two objectives and consist of different numbers of clusters. This may be a very useful feature under certain circumstances: human experts may find it preferable to have the opportunity to choose from a set of clustering solutions. This provides the opportunity to analyze several alternative solutions and bring to bear any specialized domain expertise available. On the other hand, other applications may require the *automatic* selection of just one 'best' solution. In this section, we therefore introduce a method for identifying the most promising clustering solutions in the candidate set; the selection of a single solution then automatically delivers an estimate of the number of clusters in the data set.

The identification of promising solutions from Pareto fronts has been investigated in several recent works [4, 8, 9, 34, 46]. The above papers have generally dealt with the *reduction* of the size of the Pareto front in the absence of additional knowledge about user preferences, and this

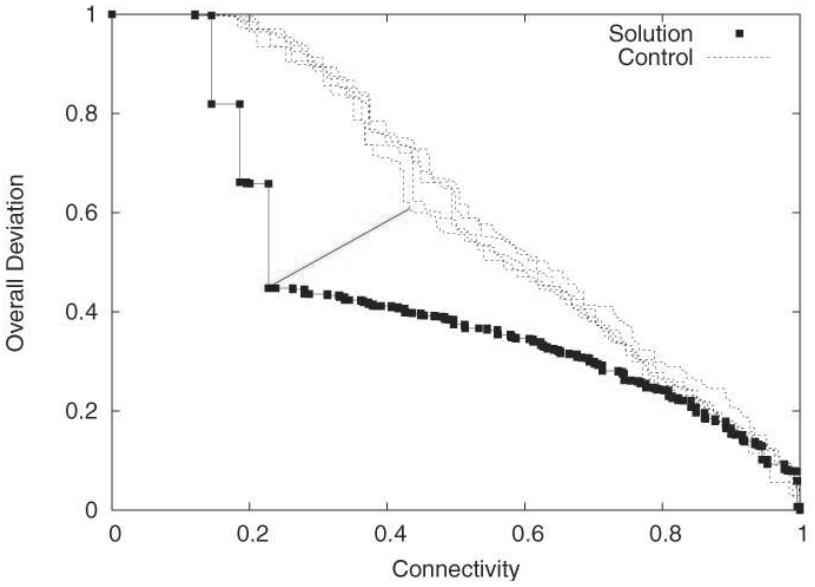

Figure 7: Solution and control reference fronts for a run of MOCK on the *Square1* data set. The solution with the largest minimum distance to the reference fronts is indicated by the angled line, and corresponds to the desired $k = 4$ cluster solution.

is done to guide or focus the search towards the (potentially) more important areas. In contrast, we seek to first obtain the most complete PF possible, and then to *a posteriori* reduce this set to a single solution. Thus, our method is slightly different to those above, and the more so because we also make use of several domain-specific considerations that enable us to make a more effective technique for our particular purposes.

The approach we have developed is inspired by Tibshirani et al.'s Gap statistic [50], a statistical method to determine the number of clusters a the data set. The Gap statistic is based on the expectation that the most suitable number of clusters shows in a significant 'knee' when plotting the performance of a clustering algorithm (in terms of a selected internal evaluation measure) as a function of the number of clusters. As internal evaluation measures are generally biased by the number of clusters (they show an increasing/decreasing trend that is solely due to a change in the number of clusters), the 'knee' can be best identified in a normalized plot, that is a performance plot that takes out the bias resulting purely from a change in the number of clusters. Tibshirani et al. realize this by generating a number of reference partitionings for uniformly random data. From the normalized performance curve, they then identify the smallest number of clusters for which the gain in performance is not higher than would be expected for random data. See Section 4.1.4 for more details on the implementation of the Gap statistic.

Several aspects of this idea can be carried over to our case of two objectives. Intuitively, we equally expect the structure of the data to be reflected in the shape of the Pareto front. From the two objectives employed, overall deviation decreases with an increasing number of clusters, whereas connectivity decreases. Hence, we can say that we gain an improvement in overall deviation δD at the cost of a degradation in connectivity δC. For a number of clusters smaller than the true number k, we expect the ratio $R = \frac{\delta D}{\delta C}$ to be large: the separation of two clusters will trigger a great decrease in overall deviation, with only a small increase in connectivity. When we surpass the correct number of clusters this ratio will diminish: the decrease in overall deviation will be less significant but come at a high cost in terms of connectivity (because a true cluster is being split). Using this knowledge, let us consider a plot of the Pareto front. Due to the natural bias of both measures, the solutions are approximately ordered by the number of clusters they contain: k gradually increases from left to right. The distinct change in R

occurring for the correct number of clusters can therefore be seen as a 'knee'. In order to help us correctly determine this knee, we can again use uniformly random reference data distributions. Clustering a number of such distributions using MOCK, provides us with a set of 'reference fronts' (see Figure 7).

Unfortunately, a normalization of the original 'solution front' using the 'reference fronts' is not as straightforward as the normalization of the performance curve for the Gap statistic. This is because both solution and control fronts contain not just one, but a set of solutions for every value of k, and it is therefore not clear how individual points in the solution front should be normalized. We overcome this problem by a heuristic approach that is explained in the following.

- Given both solution and reference fronts, we set $k_{min} = 1$ and identify k_{max}, the highest number of clusters shared by all fronts. Subsequently, we restrict the analysis to solutions with a number of clusters $k \in [k_{min}, k_{max}]$. Solution points that are dominated by any reference point are also excluded from further consideration.

- For each front, we then determine the minimum and maximum value of both overall deviation and connectivity, and use these to normalize each front to lie within the region $[0, 1] \times [0, 1]$.

- We further normalize points by taking the square root of each objective, a step motivated by the observation that both overall deviation and connectivity show a non-linear development with respect to k. Overall deviation decreases very rapidly for the first few k, while changes for higher number of clusters are far less marked. Connectivity, in contrast rises very quickly for higher numbers of clusters, while initial changes in the degree of connectivity are rather small. This results in an uneven sampling of the range of objective values, which — in a plot of the Pareto front — shows in a high density of points at the tails, with fewer solution points in the centre. Taking the square root of the objective values is an attempt to reduce this 'squeezing' effect, and give a higher degree of emphasis to small (but distinct) changes in the objectives. By this means, the algorithm becomes more precise at identifying solutions situated in all parts of the Pareto front, in particular those at the tails which may correspond e.g. to partitionings with elongated cluster shapes or a high number of clusters.

- For both solution and reference fronts, we subsequently compute the attainment surfaces [19]. The attainment surface of a Pareto front is uniquely defined by the points in the front and divides up the objective space into two regions: one that is dominated by the discovered nondominated points, and a region that is not dominated by them (see Figure 7). For each point in the solution front we then compute its distance to the attainment surfaces of each of the reference fronts, and we refer to this distance as the 'attainment score'. For a given solution point p, we compute its attainment score as the Euclidean distance between p and the closest point on the the reference attainment surface.

- Finally, we plot the attainment scores as a function of the number of clusters k (see Figure 8). The maximum of the resulting curve provides us with the number of clusters k. The solution corresponding to the highest attainment score for this k is selected as the best solution.

The above methodology provides one single partitioning, hence simulating the functioning of existing clustering techniques, which may be the most convenient mode of operation for a user without much knowledge about clustering. The reader should note however, that MOCK has the potential to provide far more information than just a single clustering solution. In particular, the shape of the Pareto front, and the location of the knee, reveal valuable information about the structure of the data, i.e.. the degree of compactness and of spatial separation between clusters. Moreover, the plot of attainment scores may contain a number of distinct local maxima,

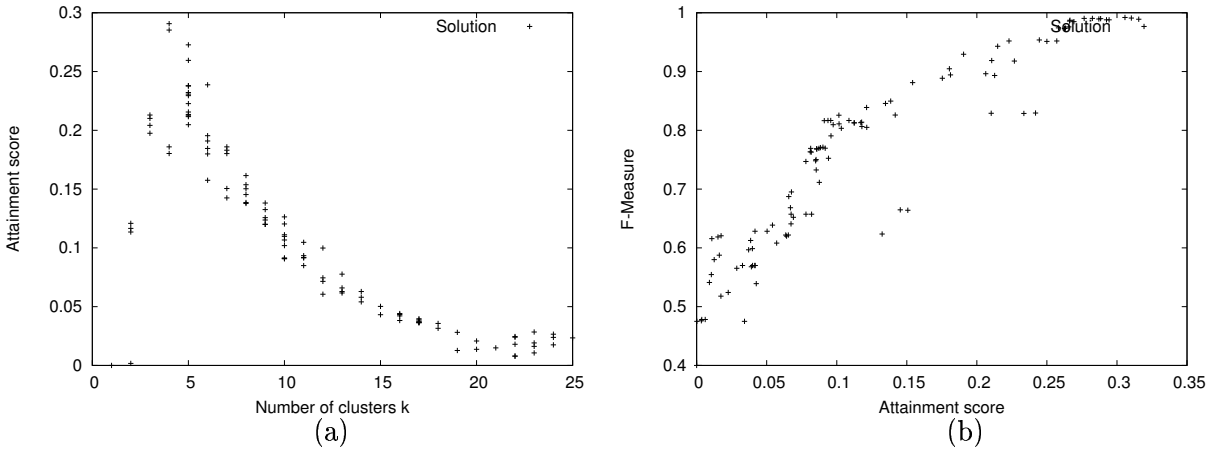

Figure 8: Attainment scores for the *Square1* data set. (a) Plot of the scores as a function of k. The global maximum at $k = 4$ is clearly visible. (b) Scatter plot to indicate the high correlation between the attainment score and the F-Measure. Ideally, all points would lie on a monotonically increasing curve. (Recall that an F-measure of one represents a perfect partitioning of the data with respect to the known class labels).

which may correspond to cluster structures on different levels. And finally, the attainment score itself provides an estimate of the quality of each individual solution. Hence, unlike more traditional 'black-box' clustering algorithms, MOCK indicates the degree of confidence to have in a particular solution and can reveal the lack of structure in the data. In our results in Section 5, we provide commentary on several attainment/reference surface plots, describing the structure that each reveals.

3.4 Parameter settings

The default parameter settings that we advise using for MOCK (and which we use over the entire range of data sets in our comparison study — next section) are given in Table 3.4. To keep computational complexity low, only five reference fronts are used, but the reader should note that a higher number of control (and, optionally, solution) fronts may yield an improved robustness of the algorithm.

We use a large external population in order to avoid losing any of the non-dominated solutions generated during the clustering process (i.e., simulating an archive of non-finite size). In our experiments, a size of 1000 proves to be more than sufficient; with all other parameter settings fixed, the final number of nondominated solutions is usually only around 100.

In the definition of connectivity, the choice of a reasonably large value of L helps to prevent outliers from being classified as individual clusters. Here, we chose a value of $L = 20$, which still permits to robustly detect clusters consisting of down to five data items.

The space of solutions is restricted to partitionings with a number of clusters in the range $\{1, \ldots, 25\}$ and a minimum cluster size of two data items. This is achieved by setting each of its objective values to ∞, preventing it from entering EP (and making it extinct).

Algorithm 3 Identification of the best solution

```
 1: procedure IDENTIFY
 2:     SF := solution front
 3:     for each i in 1 to #(control fronts) do
 4:         CF_i := ith control front
 5:     end for
 6:     k_SF := MAXK(SF)                                  /* Procedure defined in line 35 */
 7:     for each i in 1 to #(control fronts) do
 8:         k := MAXK(CF_i)
 9:         k_CF := min(k_CF, k)
10:     end for
11:     k_max := min(k_SF, k_CF)
12:     SF := FILTER(SF, k_max)                           /* Procedure defined in line 38 */
13:     for each i in 1 to #(control fronts) do
14:         CF_i := FILTER(CF_i, k_max)
15:     end for
16:     SF := NORMALIZE(SF)                               /* Procedure defined in line 42 */
17:     for each i in 1 to #(control fronts) do
18:         CF_i := NORMALIZE(CF_i)
19:     end for
20:     best_solution := −1
21:     best_score := −1
22:     for each s in SF do
23:         for each i in 1 to #(control fronts) do
24:             for each c ∈ CF_i do
25:                 s_score := DISTANCE(s, c)             /* Procedure defined in line 47 */
26:                 best_score := max(best_score, s_score)
27:                 if best_score = s_score then
28:                     best_solution := s
29:                 end if
30:             end for
31:         end for
32:     end for
33:     return best_solution
34: end procedure

35: procedure MAXK(solution set S)
36:     return largest k encountered in S
37: end procedure

38: procedure FILTER(solution set S, K)
39:     remove all solutions with k <= K from S
40:     return S
41: end procedure

42: procedure NORMALIZE(solution set S)
43:     normalize the range of objective values encountered in S to [0, 1]
44:     take the square root of each objective value
45:     return S
46: end procedure

47: procedure DISTANCE(s,c)
48:     return the shortest Euclidean distance between s and the attainment surface passing through c
49: end procedure
```

Table 1: Parameter settings for MOCK.

Parameter	*setting*
Number of generations	200
External population size	1000
Internal population size	$\max(50, \frac{N}{20})$
Initialization	Minimum spanning tree
Mutation type	L nearest neighbours ($L = 20$)
Mutation rate p_m	$1/N$
Recombination	Uniform crossover
Recombination rate p_r	0.7
Objective functions	Overall deviation and connectivity ($L = 20$)
Constraints	$k \in \{1, \ldots, 25\}$, cluster size > 2
Number of reference distributions	5

4 Experimental setup

4.1 Contestant methods

We evaluate MOCK by comparing the algorithm to:

(i) three traditional and conceptually different clustering algorithms: k-means, single link agglomerative clustering and average link agglomerative clustering,

(ii) a cluster ensemble integrating all three of the above, and

(iii) the Gap statistic for estimating the number of clusters in a data set.

While the four methods comprising (i) and (ii) all require k, the target number of clusters, to be specified by the user, (iii) allows us also to evaluate MOCK's performance at automatically estimating the number of clusters. Each of these contestant methods is described in greater detail below.

Note that for (i), we select a number of conceptually simple and well-proven algorithms. These remain by far the most commonly in-use and it is a stern test for any algorithm to compete effectively with them over a diverse range of data sets. More importantly for us, though, the choice of these algorithms reflects our wish to demonstrate that MOCK achieves a high level of performance not because the objectives it optimizes are new or different, but rather, because MOCK uses more than one of them. Thus, k-means and single-link agglomerative clustering, in particular, provide a most relevant baseline because each uses a clustering objective that is conceptually very similar to one of MOCK's. The additional use of clustering ensembles enables us to determine if there is an additional benefit to optimizing different objectives during clustering, as compared with the retrospective combination of partitions that is the basis of ensemble methods.

4.1.1 k-means

Starting from a random partitioning, the k-means algorithm repeatedly (i) computes the current cluster centers (i.e., the average vector of each cluster in data space) and (ii) reassigns each data item to the cluster whose center is closest to it. It terminates when no more reassignments take place. By this means, the intra-cluster variance, that is, the sum of squares of the differences between data items and their associated cluster centers, is locally minimized.

Our implementation of the k-means algorithm is based on the batch version of k-means, that is, cluster centers are only recomputed after the reassignment of all data items. As k-means

can sometimes generate empty clusters, these are identified in each iteration and are randomly reinitialized. This enforcement of the correct number of clusters can prevent convergence, and we therefore set the maximum number of iterations to 100. To reduce suboptimal solutions k-means is run repeatedly (100 times) using random initialisation (which is known to be an effective initialization method [42]) and only the best result in terms of intra-cluster variance is returned.

4.1.2 Hierarchical clustering

As a second and third method, two agglomerative hierarchical clustering algorithms are implemented. Both follow the same scheme, but employ different linkage metrics, namely *average link* and *single link*. In general, an agglomerative clustering algorithm starts with the finest partitioning possible (i.e., singletons) and, in each iteration, merges the two least distant 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 $j \in C_j$. For the linkage metric of single link, the distance between two clusters C_i and C_j is computed as the smallest dissimilarity between all possible pairs of data elements i and j with $i \in C_i$ and $j \in C_j$. The algorithm terminates when the target number of clusters has been obtained.[3]

4.1.3 Cluster ensemble

Finally, we compare against the cluster ensemble proposed by Strehl and Ghosh in [48]. Our choice of this particular ensemble technique, the 'knowledge reuse framework', is motivated by its high profile and popularity in the literature on cluster ensembles. Research in this field is still in an early stage and no thorough comparison of different ensemble techniques exists, but recent studies [51] suggest that graph-based approaches (of which this is one) may be more robust towards varying data properties than co-association based techniques.

Strehl and Ghosh's framework employs three conceptually different ensemble methods namely (1) CSPA (Cluster-based Similarity Partitioning Algorithm), (2) HGPA (Hyper-Graph Partitioning Algorithm) and (3) MCLA (Meta-Clustering Algorithm). The solutions returned by the individual combiners then serve as the input to a supra-consensus function, which selects the best solution in terms of average shared mutual information.

For the implementation of this cluster ensemble we use Strehl and Ghosh's original matlab code [47] with the correct number of clusters provided. In order to generate the input labels we use the above described algorithms, that is, k-means, average link and single link hierarchical clustering. As ensemble methods generally benefit from being provided partitionings of higher resolution (i.e. comprising more clusters), we run each algorithm for all $k \in \{2, \ldots, 20\}$. [4] The resulting 57 labelings then serve as the input to Strehl and Ghosh's method.

4.1.4 Gap statistic

As mentioned above, the Gap statistic [50] is an automated method to detect the 'knee' exhibited by the curve of an algorithm's performance as a function of the number of clusters k. Thus, the clustering problem is solved for a range of different values of k and, for each k, the resulting partitioning $C = \{C_1, .., C_k\}$ is evaluated by means of the intra-cluster variance, which is given by

[3] Other stopping criteria for the agglomerative algorithms are possible but in order to facilitate a straightforward comparison between methods, we stop based on k, as for k-means. Other methods of stopping have their own associated problems [31] and require additional knowledge, e.g. of distances between points, thus we would not expect them to perform significantly better.

[4] While in [48] the correct value of k only is used, results presented by the same authors in [21] indicate that the use of a range of different numbers of clusters yields improved solution quality. Preliminary experiments on our own data sets confirm this.

$$V(C) = \sum_{C_l \in C} \sum_{i \in C_l} (\delta(i, \mu_l))^2.$$

Here C_l is the lth cluster in the partitioning, μ_l is the corresponding cluster center, and $\delta(i, \mu_l)$ gives the dissimilarity between data item i and μ_l. The intra-cluster variance is affected by the number of clusters, such that a plot $Var(k)$ showing the evolution of $V(C)$ as a function of the input parameter k exhibits a decreasing trend that is solely caused by the finer partitioning and not by the actual capturing of structure within the data. The Gap statistic overcomes this effect through a normalization of the performance curve. B reference curves $R_b(k)$ (with $b \in \{1, ...B\}$) are computed, which are the performance curves obtained with the same clustering algorithm for uniform random reference distributions. Using these, the normalized performance curve ('Gap curve') for $Var(k)$ is then given as

$$Gap(k) = \frac{1}{B} \sum_{b=1}^{B} \log(R_b(k)) - \log(Var(k)).$$

The most suitable number of clusters is determined as the first k, for which the observed decrease in variance is significantly higher than would be expected for random data. If no such k is found, we set the number of clusters to 20.

For our implementation of the Gap statistic we use the above described k-means algorithm. We compute the performance curves for $k \in \{1, ..., 20\}$, and, for each k, we generate $B = 20$ reference distributions.

4.2 Evaluation

The clustering results of the different algorithms are compared using an objective evaluation function: the *F-Measure* is an external evaluation function that requires knowledge of the correct class labels and compares a generated clustering solution to this 'gold standard'. Essentially, the measure is based on the ideas of precision and recall from information retrieval. Each *class* i (as given by the class labels of the used benchmark data set) is regarded as the set of n_i items desired for a query; each *cluster* j (generated by the algorithm) is regarded as the set of n_j items retrieved for a query; n_{ij} gives the number of elements of class i within cluster j. For each class i and cluster j precision and recall are then defined as $p(i, j) = \frac{n_{ij}}{n_j}$ and $r(i, j) = \frac{n_{ij}}{n_i}$, and the corresponding value under the F-Measure is

$$F(i, j) = \frac{(b^2 + 1) \cdot p(i, j) \cdot r(i, j)}{b^2 \cdot p(i, j) + r(i, j)},$$

where we chose $b = 1$, to obtain equal weighting for $p(i, j)$ and $r(i, j)$. The overall F-measure for the partitioning is computed as

$$F = \sum_i \frac{n_i}{n} \max_j \{F(i, j)\},$$

where n is the total size of the data set. F is limited to the interval $[0, 1]$ and should be maximized.

4.3 Experimental data

MOCK's clustering performance is evaluated on a total of 21 data sets from the following three different groups:

$2d$ synthetic data: This group of two-dimensional synthetic data sets permits the modulation of specific data properties. In particular, we design data sets that exhibit difficult features

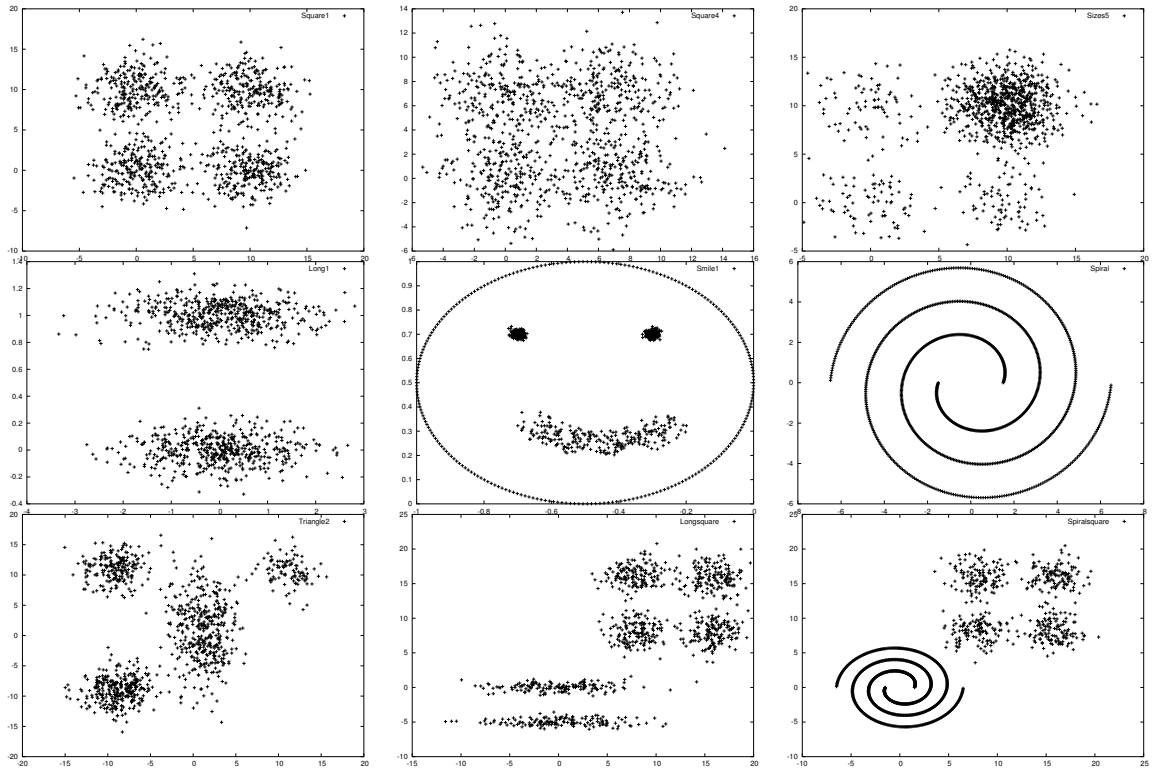

Figure 9: Sample instances of the two-dimensional synthetic data sets. *Square1* consists of four clusters of equal size and spread. The *Square* series and the *Sizes* series are variations of this data set, that vary the degree of overlap and the relative size of clusters respectively. Due to the lack of spatial separation between clusters, these problems are hard to solve for algorithms based on optimizing connectedness or spatial separation. The *Long* series, *Smile* series, *Triangle* series and the *Spiral* data set contain elongated cluster shapes that are hard to identify for algorithms based on cluster compactness. Finally, *Longsquare* and *Spiralsquare* combine different kinds of clusters and are therefore impossible to solve for both methods based on cluster compactness and those based on connectedness.

such as high overlap between clusters, unbalanced clusters, elongated cluster shapes and combinations of these. The individual clusters are predominantly described by means of two-dimensional normal distribution $N(\vec{\mu}, \vec{\sigma})$ (the only exceptions being the use of a geometrical description for both clusters in the *Spiral* data and one cluster in each data set of the *Smile* series). The number of clusters, the sizes of the individual clusters, and the mean vector $\vec{\mu}$ and vector of the standard deviation $\vec{\sigma}$ for each normal distribution are manually fixed. In each run of the experiments, a new 'instance' is sampled from these distributions, and the size of each instance is 1000 (except for the *Longsquare* and the *Spiralsquare* data set, which have a size of 900 and 1500 respectively). In this paper, we can only present results on a representative set of this type of data (see Figure 9 for sample instances), but complete results are available at [1], which also provides a detailed description of all the individual data sets.

xD-yC **synthetic data:** The 2d synthetic data sets are useful for the analysis of the algorithms' performance with respect to specific data properties. However, for sake of generality, we have additionally introduced a range of synthetic test sets, in which several key determining features or properties of each data set are randomly generated within some range. These test sets are denoted as xD-yC, where x indicates the dimensionality of the data and y gives the number of clusters. Each one of the sets consists of 50 different instances and each individual instance is generated as follows. We specify a set of y x-dimensional normal distributions $N(\vec{\mu}, \vec{\sigma})$ from which we sample the data items for the y different clusters in the instance. The sample size s of each normal distribution, the mean vector $\vec{\mu}$ and the vector of the standard deviation $\vec{\sigma}$ are themselves randomly determined using uniform distributions over fixed ranges (with $s \in [50, 450]$, $\mu_i \in [0, 100]$ and $\sigma_i \in [0, 5]$). Consequently, the expected size of instances of xD-$4C$ and xD-$10C$ is 1000 and 2500 data items respectively. During the generation process, cluster centres are rejected if the resulting distributions are likely to have more than 3% overlap. A different instance is used in each individual run of the experiments. Results are presented for four synthetic data sets of this type: these are the sets 2D-4C, 2D-10C and 10D-4C, 10D-10C, 100D-4C and 100D-10C.

Real data: Finally, six real data sets from the Machine Learning Repository [3] are used, which we summarize in Table 2. Note that these data are not necessarily designed for unsupervised (clustering) methods, and are often used as benchmarks for supervised techniques. Therefore, performance on these data is sometimes low for all algorithms, as cluster structure does not necesarily accord with the class labels. Nonetheless, we include these data because they exhibit a great diversity in dimensionality, cluster number and shape, etc., and allow us to be more confident about any general conclusions we might draw from the synthetic data.

4.3.1 Data Processing

The synthetic data requires no preprocessing, and distance computation is done using the Euclidean distance.

The real data is subject to two simple preprocessing steps: missing values are replaced by zero, and the data vectors are then normalized to have mean zero and standard deviation one in each dimension. Dissimilarities between data vectors are computed using the Cosine similarity. Note that this setup may not be the most suitable for all data sets (e.g. the Cosine similarity with no normalization performs better on the Iris data set), but for ease of design we constrict ourselves to the use of just one configuration.

All algorithms in the comparison are provided with the data in exactly the same form.

Table 2: Summary of the used real data sets from the Machine Learning Repository. N is the total number of data items in the data set, N_i is the number of items for cluster i, and D and k give the dimensionality and the number of clusters respectively.

Name	N	N_i	D	k	**Type**
Dermatology	366	112, 61, 72, 49, 52, 20	34	6	Integer
Iris	150	3×50	4	3	Continuous
Pendigits	3498	363, 364, 364, 336, 364 335, 336, 364, 336, 336	16	10	Integer
Wine	178	59, 71, 48	13	3	Continuous
Wisconsin	699	458, 241	9	2	Integer
Yeast	1484	463, 429, 244, 163, 51 44, 37, 30, 20, 5	8	10	Continuous

5 Results

For each algorithm and each data set, we present results over 50 runs. Thereby, MOCK's performance is evaluated on several levels:

- $MOCK$ labels the results obtained using the full functionality of MOCK, that is, if both the best clustering solution and the number of clusters are *automatically* determined.
- $MOCK_k$ labels the results obtained if the number of clusters k is known. In this case the solution with the highest attainment score for k clusters is selected. The performance of MOCK on this level is particularly relevant, as all other clustering algorithms in our comparison are also given the advantage of being provided with the correct number of clusters.
- $MOCK_{ideal}$ labels the results obtained if the *best* solution in the final Pareto front is selected (based on the F-Measure value). These results are interesting for two reasons. On the one hand, they provide information on the performance of MOCK at finding high-quality solutions, and its performance in a scenario where an experimenter has the opportunity to test a set of alternative solutions. On the other hand, they help to put the performance of our method of solution selection into perspective.

5.1 Pure clustering performance

Figure 11 visualizes the distribution of F-Measure values obtained by the different algorithms. Overall, MOCK performs most robustly compared to its rivals. In particular, $MOCK_{ideal}$ reliably generates solutions that are comparable or better than *the best* of the other contestant methods, showing that the algorithm consistently explores high-quality solutions. Evidently, it manages to cope with a whole range of different data properties, whereas all other clustering methods tested fail on certain data sets: as can be expected, single link performs poorly in the absence of spatial separation between clusters, whereas average link and k-means fail for elongated cluster shapes. On the *Long* series, the *Longsquare* and the *Spiralsquare* data set, none of these three traditional methods succeeds in generating good solutions reliably. The

cluster ensemble method shows an impressively robust performance over a range of synthetic data sets exhibiting various different features. In particular, its performance on the Longsquare data set (where none of its input methods on its own retrieves the correct cluster structure) is notable. However, the *Sizes5*, *Triangle 2* and the *xD-yC* data sets reveal a major drawback of the algorithm: these data sets contain unevenly-sized clusters, which poses a major problem to the ensemble technique.[5] The results on the real data shows that this impairment is not insignificant, given that unevenly sized clusters are a common feature also in real data sets: four out of our six real data sets contain (mildly) unevenly sized clusters, and on each one of these, the cluster ensemble turns out as the worst or second worst performer.

On the synthetic data sets, the performance of $MOCK$ and $MOCK_K$ is comparable to $MOCK_{ideal}$ with only one exception: on the *Spiralsquare* data $MOCK$ primarily identifies the 'upper-level' two-cluster-solution,which results in a low F-Measure value on this data set. On the real data sets, the performance of $MOCK$, $MOCK_{ideal}$ and $MOCK_K$ becomes more variable: while the performance of $MOCK_k$ and $MOCK_{ideal}$ is again favourable over the entire range of real data sets (with $MOCK_{ideal}$ being the best performer on all six data sets), the F-Measure values obtained by $MOCK$ are significantly lower on the *Iris*, the *Dermatology*, the *Yeast* and *Pendigits* data sets. This is due to the fact that in these data sets, the actual cluster borders (according to the true class labels) are not consistently discernible from the structure, making it impossible or very hard to correctly identify the number of clusters.[6] While the *Iris* data set, for example, contains three classes, two of them are highly overlapping. Without the presence of a significant density gradient (as on the *Square* series of data sets) our method of k-determination will therefore prefer the two-cluster solution (see Figure 10e). As a consequence, the resulting F-Measure values obtained for $MOCK$ are significantly lower than those for $MOCK_k$ and $MOCK_{ideal}$. Note however, that on the *Wine* and the *Wisconsin* data, where a clear cluster structure is present, $MOCK$ consistently determines the correct number of clusters.

5.2 Pareto fronts reveal structure

Figure 10 shows six sample solutions fronts and the corresponding reference fronts. For each set of such fronts, the F-Measure values corresponding to the individual solutions are additionally given. While the 'knee' in the front of the *Square1* data set is clearly visible, its identification is much harder for the *Longsquare*, *Spiral* and the *Triangle2* data sets. This is because the Pareto front may contain several significant 'knees' (as in the case of the *Triangle2* data) and the 'knee' may be very small (see the plots for the *Spiral* and the *Longsquare* data). The 'knees' may also be situated in very different regions of the Pareto front, with those in the very left revealing the presence of well-separated clusters and 'knees' towards the right signifying an increasing degree of overlap. The last two plots in Figure 10 give examples of real data sets where $MOCK$ determines a too low number of clusters. On the plot for the *Iris* data the 'knees' for both the two and the three cluster solution are clearly dicernible. The former is the more pronounced and is therefore selected by $MOCK$. The solution for the *Yeast* data reveals the low degree of structure exhibited by the data set. The presence of several coequal 'knees' is the reason why $MOCK$'s estimate of k is less consistent for this data set.

5.3 Automatic determination of k

In order to analyze $MOCK$'s actual performance at determining the number of clusters on both the synthetic and real data sets, we compare it against the Gap statistic in Figure 12.

[5] This limitation has been mentioned in [48, 21], and is due to the use of hypergraph-partitioning techniques in each CSPA, HGPA and MCLA. It is a standard constraint in graph-partitioning to require partitions to be comparably sized.

[6] In particular on the *Yeast* data sets, the low F-Measure values obtained by *all* clustering algorithms indicate that the data reveals very little structure with respect to the true class labels. While this problem may be alleviated by the use of different normalization strategies, distance functions and, in particular, feature selection, an investigation of these issues is beyond the scope of this paper.

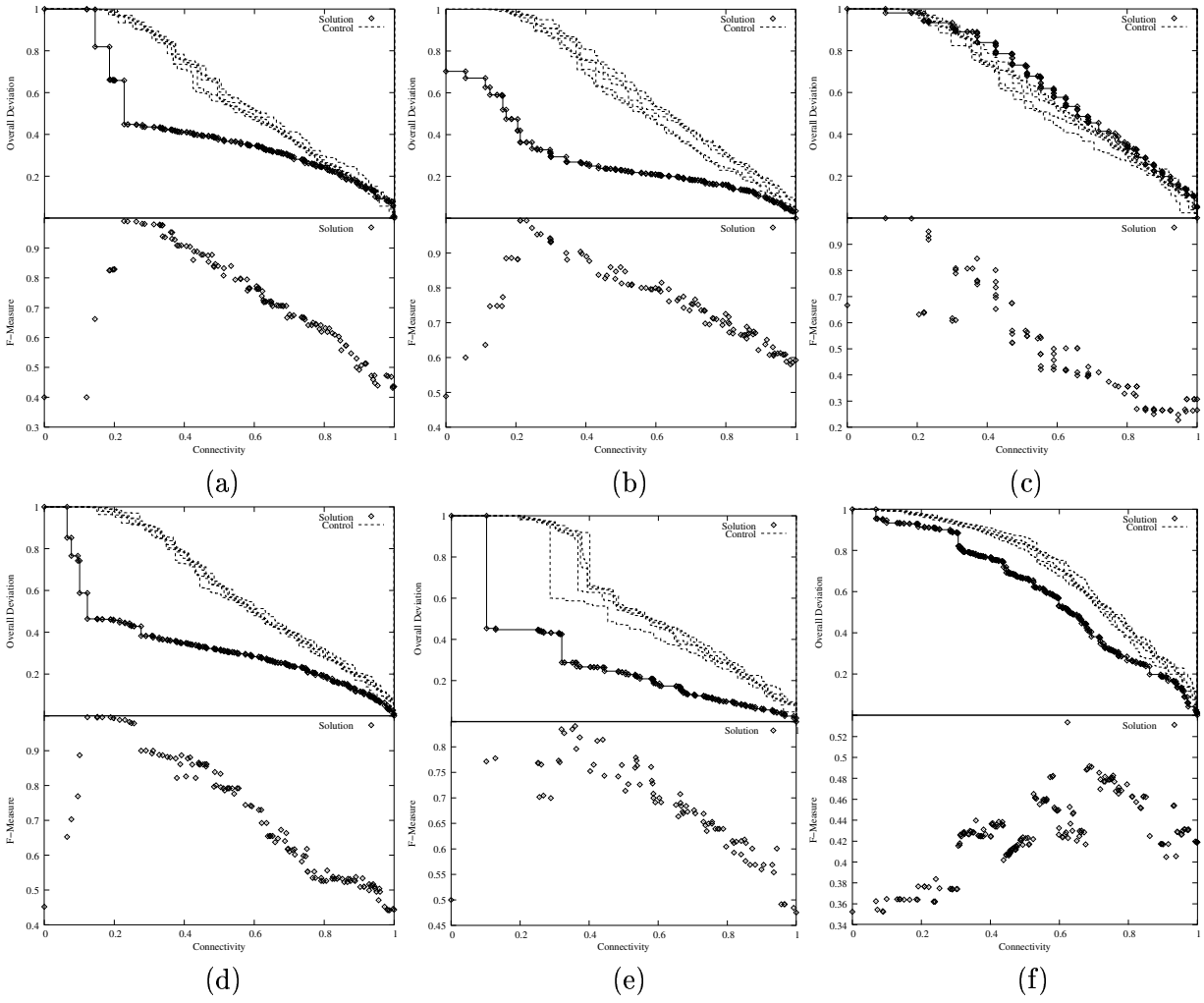

Figure 10: Normalized solution and reference fronts for four synthetic and two real data sets. (a) *Square1* (b) *Longsquare* (c) *Spiral* (d) *Triangle2* (e) *Iris* (f) *Yeast*. The lower part of the figure gives the corresponding F-Measure values as a function of connectivity. Note how, dependent on the structure of the data set, the knees corresponding to the best solution are located in different parts of the Pareto front. Also, the knees are pronounced to different degrees and deceptive knees may occur.

The obtained results make clear that *MOCK* benefits from the use of two objectives during k-determination. It reliably identifies the correct number of clusters on all but one synthetic data set (*Spiralsquare*), and on two real data sets. On two further real data sets (*Iris* and *Dermatology*) it underestimates the number of clusters by one, but is consistent in this prediction, which is justified by the actual structure of the data. Only on the *Pendigits* and the *Yeast* data is *MOCK*'s performance less good.

The Gap statistic, in contrast, which is limited to just one clustering objective, is far less robust with respect to different data properties: the results on the synthetic data demonstrate that it copes with neither highly overlapping clusters, differing cluster sizes nor elongated cluster shapes. On the real data, the Gap statistic shows a very varying performance: it is highly unstable for the *Dermatology*, the *Wine* and the *Pendigits* data sets, but apparently outperforms *MOCK* on the *Iris* and the *Yeast* data.

The reader should note that the obvious difficulty of determining the number of clusters demonstrated here, strongly puts the performance of k-means and the hierarchical clustering algorithms into perspective: in our experiments they clearly profited from being provided the correct number of clusters. However, in a real-world scenario, they would (like *MOCK*) depend on the automatic determination of the number of clusters.

5.4 Execution time

For sake of completeness, and in order to give an idea of MOCK's time complexity, Table 3 provides runtimes on a Pentium 4, 2.8 GHz PC for a number of sample data sets. The reader should note that the bulk of the computational overhead is spent during the evaluation of individual solutions. In this context it is highly favourable that both objective functions are only linear in the number of data items N. Due to the objective of *overall deviation* the algorithm's execution time also (linearly) increases with higher dimensionality D. The number of clusters k has no significant impact; however, in order to optimally search for clustering solutions with a high number of clusters, k (e.g. larger than 15), it may be necessary to increase the number of iterations.

Table 3: Runtime in minutes of *MOCK* with the parameter settings given in Table 1, for different N, D and k. The mean (and standard deviation) over 10 runs is given.

N	D	k	*Runtime (mins)*
100	2	4	0.78180 (0.0135579)
1000	2	4	1.36306 (0.032032)
1000	2	20	1.21782 (0.0389146)
1000	100	4	4.31813 (0.090189)

6 Conclusion

Existing clustering algorithms are limited to optimizing (explicitly or otherwise) one single clustering objective. This can lead to a lack of robustness with respect to different data properties, a limitation which we have suggested can be overcome by the use of several complementary objectives. Their *simultaneous* optimization does not only permit the resolution of clustering problems that otherwise require conceptually different clustering algorithms, but also affords solutions that correspond to trade-offs between different clustering objectives, and which would therefore be hard to identify by *any* traditional clustering algorithm.

In this paper we have described a new multiobjective clustering algorithm, MOCK, which optimizes two clustering objectives and attempts to automatically estimate the number of clus-

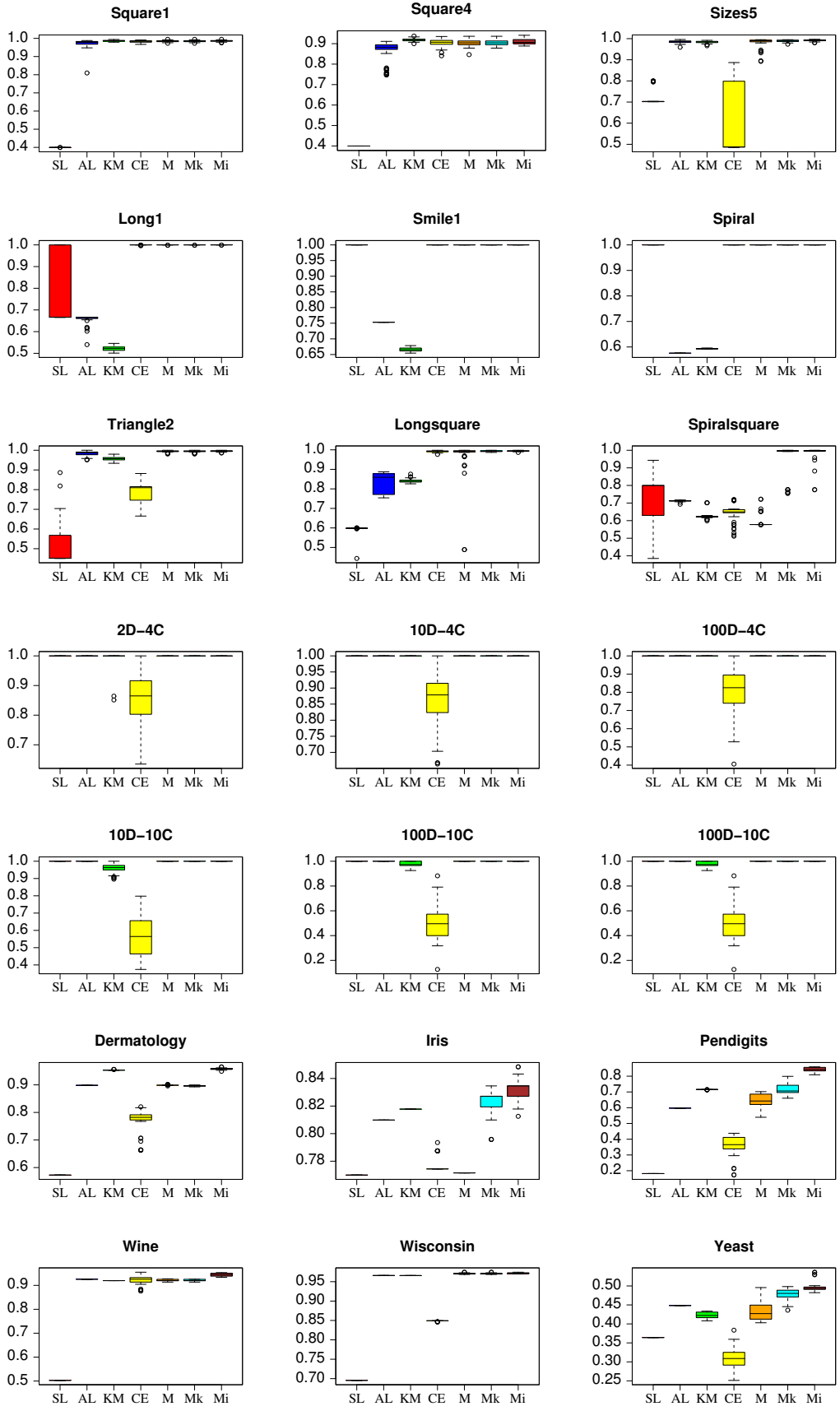

Figure 11: Boxplots [55] giving the distribution of F measure values achieved for 50 runs of each algorithm on 21 representative data sets. Key: SL = single link agglomerative, AL = average link agglomerative clustering, clustering, KM = k-means, CE = cluster ensemble, M = $MOCK$, Mi= $MOCK_{ideal}$ (best solution in terms of the F-Measure sekected), Mk = $MOCK_k$ (number of clusters k known). Median and IQR values for these and additional data sets have also been tabulated and can be found at [1].

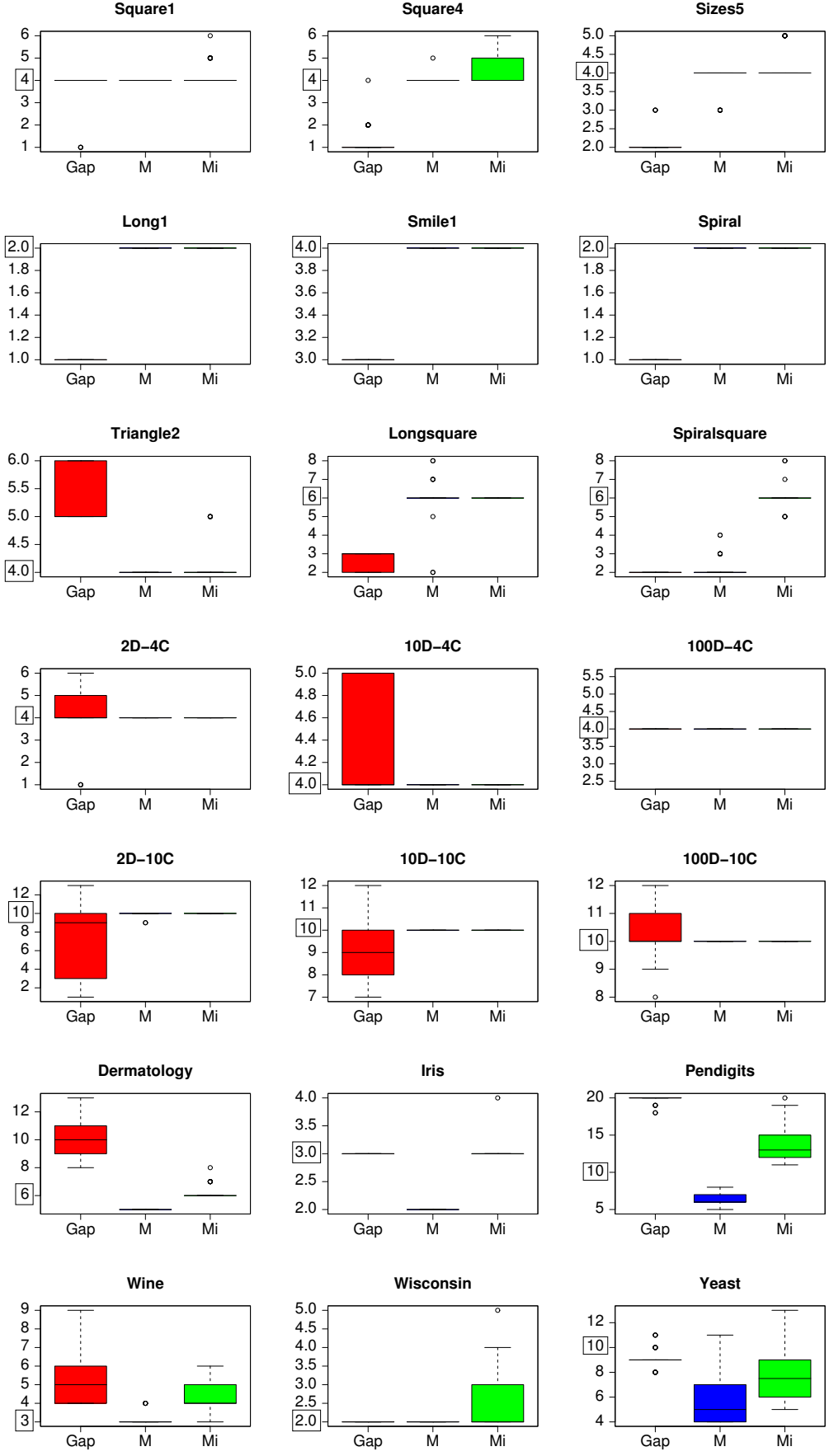

Figure 12: Boxplots [55] giving the distribution of the cluster number determined for 50 runs of each algorithm on 21 representative data sets. The correct number of clusters is framed. Key: Gap = Gap statistic, M = $MOCK$, Mi= $MOCK_{ideal}$ (the number of clusters in the solution with the highest F-measure value). Median and IQR values for these and additional data sets have also been tabulated and can be found at [1].

ters in a data set. In a comparison of the algorithm to a set of conceptually diverse clustering algorithms, MOCK has shown an impressively robust performance. The algorithm reliably generates high quality solutions, which are comparable to or better than those identified by the best of four other contestant methods. Moreover, it considerably outperforms the Gap statistic with its estimates of the number of clusters in the data. Importantly, MOCK's performance was achieved *without any adjustment of its parameters* between the test problems. Thus, we tentatively suggest that MOCK is ready for use by 'field practitioners' in an off-the-shelf configuration (with the parameter choices as stated herein), and we make this freely available at [1]. In this respect, MOCK would seem a more straightforward choice than some other advanced clustering techniques, such as ensemble methods, where greater expertise is needed for effective use. However, we look forward to more direct comparisons between these conceptually different approaches in future.

The use of a multiobjective framework for clustering offers a great amount of flexibility that we hope to exploit in further work. In particular, we are currently investigating the integration of feature selection and semi-supervised learning by means of additional objectives.

Acknowledgments

MOCK is built from David Corne's original PESA-II code. We are grateful to Alexander Strehl for making available the ClusterPack MATLAB toolbox. Thanks to Douglas Kell for help with the preparation of this manuscript and general advice. JH 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.

References

[1] Supporting material. http://dbk.ch.umist.ac.uk/handl/mock/.

[2] M. Ankerst, M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the 1999 International Conference on Management of Data*, pages 49–60. ACM Press, New York, NY, 1999.

[3] C. Blake and C. Merz. UCI repository of machine learning databases. Department of Information and Computer Sciences, University of California, Irvine. http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[4] J. Branke, K. Deb, H. Dierolf, and M. Osswald. Finding knees in multi-objective optimization. In *Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature*, Birmingham, UK, 2004. Springer. To appear.

[5] Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Kluwer Academic Publishers, New York, May 2002.

[6] 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.

[7] David W. Corne, Joshua D. Knowles, and Martin J. Oates. The Pareto envelope-based selection algorithm for multiobjective optimization. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 839–848, Paris, France, 2000. Springer.

[8] I. Das. On characterizing the 'knee' of the Pareto curve based on normal-boundary intersection. *Structural Optimization*, 18:107–115, 1999.

[9] K. Deb. *Multi-objective evolutionary algorithms: Introducing bias among Pareto-optimal solutions*, pages 263–292. Springer-Verlag, London, UK, 2003.

[10] Kalyanmoy Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001.

[11] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[12] Ch. Ding and X. He. K-nearest-neighbour consistency in data clustering: incorporating local information into global optimization. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 584–589, New York, NY, 2004. ACM Press.

[13] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999.

[14] S. Dudoit and J.A. Fridlyand. A prediction-based resampling method for estimating the number of clusters in a dataset. *Genome Biology 25*, 3, 2002.

[15] M. Ester, H.-P. Kriegel, and J. Sander. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data-Mining*, pages 226–231. AAAI Press, Menlo Park, CA, 1996.

[16] V. Estivill-Castro. Why so many clustering algorithms: A position paper. *ACM SIGKDD Explorations Newsletter Archive*, 4:65–75, 2002.

[17] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley & Son Ltd, New York, NY, 1998.

[18] G. Fleurya, A. Hero, S. Zareparsi, and A. Swaroop. Gene discovery using Pareto depth sampling distributions. *Special Issue on Genomics, Signal Processing and Statistics, Journ. of Franklin Institute*, 341:55–75, 2004.

[19] C. M. Fonseca and P. J. Fleming. On the performance assessment and comparison of stochastic multiobjective optimizers. In *Proceedings of the Fourth International Conference onParallel Problem Solving from Nature*, pages 584–593. Springer-Verlag, Berlin, Germany, 1996.

[20] C. Fraley and A. E. Raftery. How many clusters? Which clustering method? Answers via model-based cluster analysis. Technical Report 329, Department of Statistics, University of Washington, 1998.

[21] J. Ghosh, A. Strehl, and S. Merugu. A consensus framework for integrating distributed clusterings under limited kowledge sharing. In *Proceedings of the NSF Workshop Next Generation Data Mining*, pages 99–108, Baltimore, MD, 2002. AAAI/MIT Press.

[22] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5:533–549, 1986.

[23] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17:107–145, 2001.

[24] J. Handl and J. Knowles. Evolutionary multiobjective clustering. In *Proceedings of the Eighth International Conference on Parallel Problem Solving from Nature*, Birmingham, UK, 2004. Springer. To appear.

[25] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge MA, 1992.

[26] Jeffrey Horn. *The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations.* PhD thesis, University of Illinois at Urbana Champaign, Urbana, IL, 1997.

[27] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31:264–323, 1999.

[28] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data.* John Wiley & and Sons Inc., New York, NY, 1990.

[29] Y. Kim, W. N. Street, and F. Menczer. Evolutionary model selection in unsupervised learning. *Intelligent Data Analysis*, 6:531–556, 2002.

[30] S. Kirkpatrick, C.D.Jr. Gerlatt, and M.P. Vecchi. Optimization by simulated annealing. *Science 220*, pages 671–680, 1983.

[31] J. Kleinberg. An impossibility theorem for clustering. In *Proceedings of the 15th Conference on Neural Information Processing Systems*, Vancouver, Canada, 2002.

[32] M. H.C Law. Multiobjective data clustering. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, 2004. To appear.

[33] 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.

[34] C. A. Mattson, A. A. Mullur, and A. Messac. Minimal representation of multiobjective design space using a smart Pareto filter. In *AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, 2002. AIAA Press.

[35] U. Maulik and S. Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33:1455–1465, 2000.

[36] G. McLachlan and T. Krishnan. The EM algorithm and extensions. In *Wiley Series in Probability and Statistics*, NEy York, NY, 1997. John Wiley & Sons.

[37] G. W. Milligan and M. C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179, 1985.

[38] B. Minaei-Bidgoli, A. Topchy, and W. F. Punch. Ensembles of partitions via data resampling. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, Las Vegas, NV, 2004. IEEE Press. To appear.

[39] M. Morita, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Unsupervised feature selection using multi-objective genetic algorithms for handwritten word recognition. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, pages 666–671, Edinburgh, UK, 2003. IEEE Press.

[40] H. Pan, J. Zhu, and D. Han. Genetic algorithms applied to multi-class clustering for gene expression data. *Genomics, Proteomics & Bioinformatics*, 1, 2003.

[41] 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, Madison, WI, 1998. Morgan Kaufmann.

[42] J. M. Pena, J. A. Lozana, and P. Larranaga. An empirical comparison of four initialization methods for the k -means algorithm. *Pattern Recognition Letters*, 20:1027–1040, 1999.

[43] J. Quackenbush. Computational analysis of microarray data. *Nature Reviews Genetics*, 2:418–427, 2001.

[44] O. Sasson and M. Linial. Protein clustering and classification. *To appear in Bioinformatics*, 2004.

[45] D. K. Slonim. From patterns to pathways: gene expression data analysis comes of age. *Nature Genetics*, 32:502–508, 2002.

[46] G. Stehr, H. Graeb, and K. Antreich. Performance trade-off analysis of analog circuits by normal-boundary intersection. In *40th Design Automation Conference*, Anaheim, CA, 2003. IEEE Press.

[47] A. Strehl. ClusterPack Matlab Toolbox. http://www.lans.ece.utexas.edu/~strehl/soft.html.

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

[49] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

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

[51] A. Topchy, A. K. Jain, and W. Punch. Clustering ensembles: Models of consensus and weak partitions. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004.

[52] A. Topchy, A. K. Jain, and W. Punch. A mixture model for clustering ensembles. In *Proceedings of the SIAM International Conference on Data Mining*, Lake Buena Vista, FL, 2004. SIAM. To appear.

[53] A. Topchy, B. Minaei, A. K. Jain, and W. Punch. Adaptive clustering ensembles. In *Proceedings of the International Conference on Pattern Recognition*, Cambridge, UK, 2004. To appear.

[54] E. Vorhees. *The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval.* PhD thesis, Department of Computer Science, Cornell University, 1985.

[55] E. W. Weisstein. Box-and-whisker plot. From MathWorld — A Wolfram Web Resource, http://mathworld.wolfram.com/Box-and-WhiskerPlot.html.

[56] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[57] Robin J. Wilson and John J. Watkins. *Graphs: an introductory approach: a first course in discrete mathematics.* John Wiley and Sons, New York, NY, 1990.