

Strategies for the increased robustness of ant-based clustering

J. Handl, J. Knowles and M. Dorigo

IRIDIA, Université Libre de Bruxelles
jhandl@iridia.ulb.ac.be, {jknowles,mdorigo}@ulb.ac.be

Abstract. This paper introduces a set of algorithmic modifications that improve the partitioning results obtained with ant-based clustering. Moreover, general parameter settings and a self-adaptation scheme are devised, which afford the algorithm's robust performance across varying data sets. We study the sensitivity of the resulting algorithm with respect to two distinct, and generally important, features of data sets: (i) unequal-sized clusters and (ii) overlapping clusters. Results are compared to those obtained using k -means, one-dimensional self-organising maps, and average-link agglomerative clustering. The impressive capacity of ant-based clustering to automatically identify the number of clusters in the data is additionally underlined by comparing its performance to that of the Gap statistic.

1 Introduction

Ant-based clustering is a heuristic clustering method that draws its inspiration from the behaviour of ants in nature [1]. In particular, it models the clustering and sorting that can be observed in some ant species: where *real ants* gather corpses to clean up their nest, or transport larvae to order them by size, *artificial ants* transport *data items* that are laid out in an artificial environment, and spatially arrange them in a sorted fashion.

Thus, the working principles of ant-based clustering are quite different from those of ordinary clustering algorithms. While well-known clustering methods like k -means or agglomerative clustering gradually build or refine an *explicit* representation of a data set's partitioning, ant-based clustering uses no such model but only implicitly generates the partitioning: all information on the number of clusters and the cluster memberships of individual data items is contained in the final spatial distribution of the data. Also, this outcome is obtained without the explicit specification of an optimisation criterion or global goal, it emerges in a self-organised fashion as the result of local actions and positive feedback only.

As no a priori assumptions on the number, the shape or the size of the clusters in the data need to be made, the risk of producing solutions that are mere artefacts of such assumptions, and which do not reveal any actual information about true data structures, is reduced. In spite of these favourable features, applications of ant-based clustering are rare, as not much is known about the algorithm's real performance and robustness. In this paper we take a step towards

a wider application of ant-based clustering by introducing an improved version of the algorithm, and devising generalised parameter settings that permit its application across varying data sets. The algorithm’s robustness towards different data properties is then studied using two series of synthetic benchmarks. The obtained results are analysed in terms of the number of clusters identified and in terms of the F-measure; we compare to the number of clusters predicted by the Gap statistic, and to the partitionings generated by each of k -means, average-link agglomerative clustering, and one-dimensional self-organising maps (see Section 4 for details).

The remainder of this paper is structured as follows. Section 2 briefly summarises previous work on ant-based clustering. In Section 3 our new version of the algorithm is described. Section 4 describes the experimental setup, Section 5 discusses results and Section 6 concludes.

2 Ant-based clustering

The first ant-based clustering and sorting algorithm was introduced by Deneubourg et al. [2] in 1990, to produce clustering behaviour in simple groups of robots. In the proposed simulation model, ants are modeled as simple agents that randomly move in a square, toroidal environment. The data items that are to be clustered / sorted are initially randomly scattered in this environment and they can be picked up, transported and dropped by the agents. A clustering and sorting of these items is obtained by introducing a bias for the picking and dropping operations, such that data items that are isolated or surrounded by dissimilar ones are likely to be picked up, and transported data items are likely to be dropped in the vicinity of similar ones.

Deneubourg et al. implement this bias by applying the following probabilities for picking and dropping operations:

$$p_{pick}(i) = \left(\frac{k^+}{k^+ + f(i)}\right)^2$$

$$p_{drop}(i) = \left(\frac{f(i)}{k^- + f(i)}\right)^2$$

Here, k^+ and k^- are parameters, which determine the influence of the *neighbourhood functions* $f(i)$ and which Deneubourg et al. set to 0.1 and 0.3 respectively. $f(i)$ is an estimation of the fraction of data items in the ant’s immediate environment that are similar to the data item i the ant currently considers to pick up / drop. In Deneubourg et al.’s original implementation this estimate is obtained using a short-term memory of each agent, where the contents of the last encountered grid cells are stored, and it therefore only permits the discrimination between a limited number of classes of data items. This limitation has been overcome by Lumer and Faieta [6], who, moving away from the use in robotics, introduced a more general definition of $f(i)$ that permits the algorithm’s application to numerical data. An agent deciding whether to manipulate an item i

now considers the average similarity of i to all items j in its local neighbourhood:

$$f(i) = \max \left(0, \frac{1}{\sigma^2} \sum_j \left(1 - \frac{d(i,j)}{\alpha} \right) \right) \quad (1)$$

Here, $d(i,j) \in [0, 1]$ is a dissimilarity function defined between points in data space, $\alpha \in [0, 1]$ is a data-dependent scaling parameter, and σ^2 is the size of the local neighbourhood (typically, $\sigma^2 \in \{9, 25\}$). The agent is located in the centre of this neighbourhood; its *radius of perception* in each direction is therefore $\frac{\sigma-1}{2}$.

The resulting algorithm still suffers from convergence problems and an unfavourable runtime behaviour, and several attempts to overcome these limitations have therefore been proposed [6, 4].

3 Our algorithm

In this section, we build upon the work of [2, 6, 4] to develop a general and robust version of ant-based clustering. In particular, we describe how parameter settings for the algorithm can be automatically derived from the data, and we introduce a number of modifications that improve the quality of the clustering solutions generated by the algorithm. In this context, our criteria for ‘quality’ are twofold: first, in the distribution generated by the ant algorithm on the grid, we desire a clear *spatial separation* between clusters, as this is a requirement both for unambiguously interpreting the solutions and for evaluating them; second, we are interested in a high accuracy of the resulting classification.

Results on the effect of individual modifications are not provided in this paper, but can be found in [3]. Here, we focus on a study of the algorithm’s overall performance, in particular its sensitivity with respect to different data properties.

3.1 Basics

The basic ant algorithm (see Algorithm 1) starts with an initialisation phase, in which (i) all data items are randomly scattered on the toroidal grid; (ii) each agent randomly picks up one data item; and (iii) each agent is placed at a random position on the grid. Subsequently, the sorting phase starts: this is a simple loop, in which (i) one agent is randomly selected; (ii) the agent performs a step of a given *stepsize* (in a randomly determined direction) on the grid; and (iii) the agent (probabilistically) decides whether to drop its data item. In the case of a ‘drop’-decision, the agent drops the data item at its current grid position (if this grid cell is not occupied by another data item), or in the immediate neighbourhood of it (it locates a nearby free grid cell by means of a random search). It then *immediately* searches for a new data item to pick up. This is done using an index that stores the positions of all ‘free’ data items on the grid: the agent randomly selects one data item i out of the index, proceeds to its position on the grid, evaluates the neighbourhood function $f^*(i)$, and

(probabilistically) decides whether to pick up the data item. It continues this search until a successful picking operation occurs. Only then the loop is repeated with another agent.

Algorithm 1 *basic_ant*

```

1: begin
2: INITIALISATION PHASE
3: Randomly scatter data items on the toroidal grid
4: for each  $j$  in 1 to  $\#agents$  do
5:    $i := \text{random\_select}(\text{remaining\_items})$ 
6:    $\text{pick\_up}(\text{agent}(j), i)$ 
7:    $g := \text{random\_select}(\text{remaining\_empty\_grid\_locations})$ 
8:    $\text{place\_agent}(\text{agent}(j), g)$ 
9: end for
10: MAIN LOOP
11: for each  $it\_ctr$  in 1 to  $\#iterations$  do
12:    $j := \text{random\_select}(\text{all\_agents})$ 
13:    $\text{step}(\text{agent}(j), \text{stepsize})$ 
14:    $i := \text{carried\_item}(\text{agent}(j))$ 
15:    $\text{drop} := \text{drop\_item?}(f^*(i))$  // see equations 3 and 4
16:   if  $\text{drop} = \text{TRUE}$  then
17:     while  $\text{pick} = \text{FALSE}$  do
18:        $i := \text{random\_select}(\text{free\_data\_items})$ 
19:        $\text{pick} := \text{pick\_item?}(f^*(i))$  // see equations 2 and 4
20:     end while
21:   end if
22: end for
23: end

```

For the picking and dropping decisions the following threshold formulae are used:

$$p_{pick}^*(i) = \begin{cases} 1.0 & \text{if } f^*(i) \leq 1.0 \\ \frac{1}{f^*(i)^2} & \text{else} \end{cases} \quad (2)$$

$$p_{drop}^*(i) = \begin{cases} 1.0 & \text{if } f^*(i) \geq 1.0 \\ f^*(i)^4 & \text{else,} \end{cases} \quad (3)$$

where $f^*(i)$ is a modified version of Lumer and Faieta's [6] neighbourhood function (Equation 1):

$$f^*(i) = \begin{cases} \max\left(0, \frac{1}{\sigma^2} \sum_j \left(1 - \frac{d(i,j)}{\alpha}\right)\right) & \text{if } \forall j \left(1 - \frac{d(i,j)}{\alpha} > 0\right) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

This definition of $f^*(i)$ combines two important properties. First, as in the original neighbourhood function $f(i)$, the division by the neighbourhood size σ^2 penalises empty grid cells, thus inducing a tight clustering (rather than just a

loose sorting). Secondly, the additional constraint $\forall j (1 - \frac{d(i,j)}{\alpha}) > 0$ serves the purpose of heavily penalising high dissimilarities, which significantly improves spatial separation between clusters.

Note that the above given threshold formulae are quite different from the ones suggested by Deneubourg et al. and are *not* applicable to the basic ant algorithm. They have been experimentally derived for the use with our enhanced version (for which they significantly speed up the clustering process) and have to be seen in light of the shift of the range of attainable values $f^*(i)$ resulting from our increase of the radius of perception (see Section 3.3 below).

3.2 Short-term memory

A modified version of the ‘short-term memory’ introduced by Lumer and Faieta in [6] is employed. In their approach, each agent remembers the last few carried data items and their respective dropping positions. When a new data item is picked up, the position of the ‘best matching’ memorised data item is used to bias the direction of the agent’s random walk. Here, the ‘best matching’ item is the one of *minimal dissimilarity* $d(i, j)$ to the currently carried data item i . We have extended this idea as follows.

In a multi-agent system the items stored in the memory might already have been removed from the remembered position. In order to determine more robustly the direction of bias, we therefore permit each ant to exploit its memory as follows: An ant situated at grid cell p , and carrying a data item i , uses its memory to proceed to all remembered positions, one after the other. Each of them is evaluated using the neighbourhood function $f^*(i)$, that is, the suitability of each of them as a dropping site for the currently carried data item i is examined. Subsequently, the ant returns to its starting point p .

Out of all evaluated positions, the one of ‘best match’ is the grid cell for which the *neighbourhood function yields the highest value*. For the following step of the ant on the grid, we replace the use of a biased random walk with an agent ‘jump’ directly to the position of ‘best match’. However, this jump is only made with some probability, dependent on the quality of the match; the same probability threshold that we use for a dropping operation $p_{drop}^*(i)$ is used for this purpose. If the jump is not made, the agent’s memory is de-activated, and in future iterations it reverts to trying random dropping positions until it successfully drops the item.

3.3 Increasing radius of perception

The size of the local neighbourhood perceived by the ants limits the information used during the sorting process. It is therefore attractive to employ larger neighbourhoods in order to improve the quality of the clustering and sorting on the grid. However, the use of a larger neighbourhood is not only more expensive (as the number of cells to be considered for each action grows quadratically with the radius of perception), but it also inhibits the quick formation of clusters during the initial sorting phase.

We therefore use a radius of perception that gradually increases over time. This saves computations in the first stage of the clustering process and prevents difficulties with the initial cluster formation. At the same time it accelerates the dissolution of preliminary small clusters, a problem that has already been addressed in [6, 4]. In the current implementation, we start with an initial perceptive radius of 1 and linearly increase it to be 5 in the end. While doing so, we leave the scaling parameter $\frac{1}{\sigma^2}$ in Equation 4 unchanged, as its increase results in a loss of spatial separation.

This brings about the gradual shift in the range of attainable values $f^*(i)$ that we have mentioned in Section 3.1. In the starting phase of the algorithm, $f^*(i)$ is limited to the interval $[0, 1]$; the upper bound, however, increases with each increment of the neighbourhood radius, such that, in our implementation, $f^*(i)$ can yield values within the interval $[0, 15]$ after the last increment. Consequently, the picking operation is purely deterministic in the beginning, and, at this stage, it is the dropping operation solely that favours dense and similar neighbourhoods. Gradually, with the rise of $f^*(i)$, an additional bias towards the picking of misplaced data items is introduced. The shift of the values of $f^*(i)$ combined with the use of the threshold functions (Equations 2 and 3) has the effect of decreasing the impact of density for the dropping threshold while, simultaneously, increasing it for the picking threshold. This results in an improved spatial separation between clusters.

3.4 Spatial separation

As stated above, the spatial separation of clusters on the grid is crucial in order for individual clusters to be well-defined. Spatial closeness, when it occurs, is, to a large degree, an artefact of early cluster formation. This is because, early on in a run, clusters will tend to form wherever there are locally dense regions of similar data items; and thereafter these clusters tend to drift only very slowly on the grid. After an initial clustering phase, we therefore use a short interlude (from time t_{start} to t_{end}) with a modified neighbourhood function, which replaces the scaling parameter $\frac{1}{\sigma^2}$ by $\frac{1}{N_{occ}}$ in Equation 4, where N_{occ} is the *actual observed number* of *occupied* grid cells within the local neighbourhood. Hence only similarity, not density, is taken into account, which has the effect of spreading out data items on the grid again, but in a sorted fashion; the data items belonging to different clusters will now occupy individual ‘regions’ on the grid. Subsequently, we turn back to using the traditional neighbourhood function. Once again, clear clusters are formed, but they now have a high likelihood of being generated along the centres of these ‘regions’, due to the lower neighbourhood quality at their boundaries.

3.5 Parameter settings

Ant-based clustering requires a number of different parameters to be set, some of which have been experimentally observed to be independent of the data. These include the number of agents, which we set to be 10, the size of the agents’

short-term memory, which we equally set to 10, and t_{start} and t_{end} , which we set to $0.45 \cdot \#iterations$ and $0.55 \cdot \#iterations$.

Parameters to be set as a function of the size of the data set. Several other parameters should however be selected in dependence of the size of the data set tackled, as they otherwise impair convergence speed. Given a set of N_{items} items, the grid (comprising a total of N_{cells} cells) should offer a sufficient amount of ‘free’ space to permit the quick dropping of data items (note that each grid cell can only be occupied by one data item). This can be achieved by keeping the ratio $r_{occupied} = \frac{N_{items}}{N_{cells}}$ constant and sufficiently low. A good value, found experimentally, is $r_{occupied} = \frac{1}{10}$. We obtain this by using a square grid with a resolution of $\sqrt{10N_{items}} \times \sqrt{10N_{items}}$ grid cells. The stepsize should permit sampling of each possible grid position within one move, which is obtained by setting it to $stepsize = \sqrt{20N_{items}}$. The total number of iterations has to grow with the size of the data set. Linear growth proves to be sufficient, as this keeps the average number of times each grid cell is visited constant. Here, $\#iterations = 2000 \cdot N_{items}$, with a minimal number of 1 million iterations imposed.

Activity-based α -adaptation. An issue already addressed in [4] is the automatic determination of the parameter α (recall that α is the parameter scaling the dissimilarities within the neighbourhood function $f^*(i)$), which the functioning of the algorithm crucially depends on. During the sorting process, α determines the percentage of data items on the grid that are classified as similar, such that: a too small choice of α prevents the formation of clusters on the grid; on the other hand, a too large choice of α results in the fusion of individual clusters, and in the limit, all data items would be gathered within one cluster.

Unfortunately, a suitable choice of the parameter α depends on the distribution of pairwise dissimilarities within the collection and, hence, cannot be fixed without regard to the data. However, a mismatch of α is reflected by an excessive or extremely low sorting activity on the grid. Therefore, an automatic adaptation of α can be obtained through the tracking of the amount of activity, which is reflected by the frequency of the agents’ successful picking and dropping operations. The scheme for α -adaptation used in our experiments is described below.

A heterogenous population of agents is used, that is, each agent makes use of its own parameter α . All agents start with an α parameter randomly selected from the interval $[0, 1]$. An agent considers an adaptation of its own parameter after it has performed N_{active} moves. During this time, it keeps track of the failed dropping operations N_{fail} . The rate of failure is determined as $r_{fail} = \frac{N_{fail}}{N_{active}}$ where N_{active} is fixed to 100. The agent’s individual parameter α is then updated using the rule

$$\alpha \leftarrow \begin{cases} \alpha + 0.01 & \text{if } r_{fail} > 0.99 \\ \alpha - 0.01 & \text{if } r_{fail} \leq 0.99 \end{cases}$$

which has been experimentally derived. α is kept adaptive during the entire sorting process. This makes the approach more robust than an adaptation method with a fixed stopping criterion. Also, it permits for the specific adaptation of α within different phases of the sorting process.

4 Evaluation

In the following we briefly describe the main experimental setup used for the evaluation of the described algorithm.

Comparison. The performance of a clustering algorithm can best be judged with respect to its relative performance when compared to other algorithms. We therefore choose three popular clustering methods from the literature, the partitioning method *k-means* [7], the hierarchical clustering algorithm *average-link agglomerative clustering* [12], and *one-dimensional self-organising maps* (1D-SOM, [5]).

All three of these algorithms require the correct number of clusters as an input parameter. While automatic and semi-automatic methods for the determination of the number of clusters within a data set exist (cf. [8] for a survey), none of these is infallible. In order to avoid the introduction of an additional source of error we therefore provide the correct number of clusters to *k-means*, average link agglomerative clustering and 1D-SOM, thus giving the same advantage to all three algorithms.

Other implementation details are as follows: The standard version of average-link agglomerative clustering is used. *k-means* is implemented using batch training and random initialisation, and only the best result out of 20 runs (in terms of the minimal intra-cluster variance) is returned. 1D-SOM is implemented in accordance with the description given in [11]: Sequential training, uniform initialisation of the weight vectors, a rectangular grid and a ‘bubble’ neighbourhood are used. The training consists of two phases: a first ‘coarse’ approximation phase of 10 iterations, with a learning rate of $lr = 0.5$ and the neighbourhood size decreasing exponentially from $\max(1.0, \frac{k}{4})$ to $\max(1.0, \frac{k}{16})$, and a second fine-tuning phase of 40 iterations, with a learning rate of $lr = 0.05$ and the neighbourhood size decreasing exponentially from $\max(1.0, \frac{k}{16})$ to 1.0 (here, k is again the number of clusters). In each iteration all data elements are presented to the SOM.

For the evaluation of ant-based clustering’s performance at identifying the correct number of clusters in the data, we additionally compare against the results returned by the Gap statistic, a recently proposed automated method for the determination of the number of clusters in a data set [9]. This statistic is based on the expectation that the most suitable number of clusters appears as a significant ‘knee’ in a plot of the performance of a clustering algorithm versus the number of clusters, k . For this purpose, the clustering problem is solved for a range of different values of k and, for each k , the resulting partitioning

$C = \{C_1, \dots, C_k\}$ is evaluated by means of the intra-cluster variance, which is given by

$$V(k) = \sum_{C_i \in C} \sum_{j \in C_i} (d(j, \mu_i))^2.$$

Here C_i is the i th cluster in the partitioning, μ_i is the corresponding cluster centre, and $d(j, \mu_i)$ gives the dissimilarity between data item j and μ_i . The intra-cluster variance is affected by the number of clusters, such that a plot of $V(k)$ 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 normalisation of the performance curve. B reference curves $R_b(k)$ (with $b \in \{1, \dots, B\}$) are computed, which are the performance curves obtained with the same clustering algorithm for uniform random reference distributions. Using these, the normalised performance curve ('Gap curve') for $V(k)$ is then given as

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

The most suitable number of clusters is determined by finding the first significant local maximum of $Gap(k)$.

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

Benchmark data. The benchmarks used in our experiments are synthetic data sets with each cluster generated by a two-dimensional normal distribution $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$. The number of clusters, the sizes of the individual clusters, and the mean vector $\boldsymbol{\mu}$ and vector of the standard deviation $\boldsymbol{\sigma}$, for each normal distribution, are manually fixed. In the experiments, each algorithm is run 50 times on each type of benchmark, and for every individual run, the actual data items are newly sampled from the normal distributions.

Table 1 gives the definition of the benchmarks, and Figure 1 shows four sample instances. The benchmarks are variations of the *Square* data set, a data set that has been frequently employed in the literature on ant-based clustering. It is two-dimensional and consists of four clusters of equal size (250 data items each), which are generated by normal distributions with a standard deviation of 2 in both dimensions and are arranged in a square.

The data sets *Square1* to *Square7* only differ by the distance between the individual clusters (i.e., the length of the edges of the square), which is 10, 9, 8, 7, 6, 5 and 4 respectively. They were generated in order to study the relative sensitivity of the algorithms to increasing overlap between clusters.

In the *Sizes1* to *Sizes5* data sets, edge length and standard deviation are kept constant, and, instead, they differ in the sizes of the individual clusters. In particular, the ratio between the smallest and the largest cluster increases from

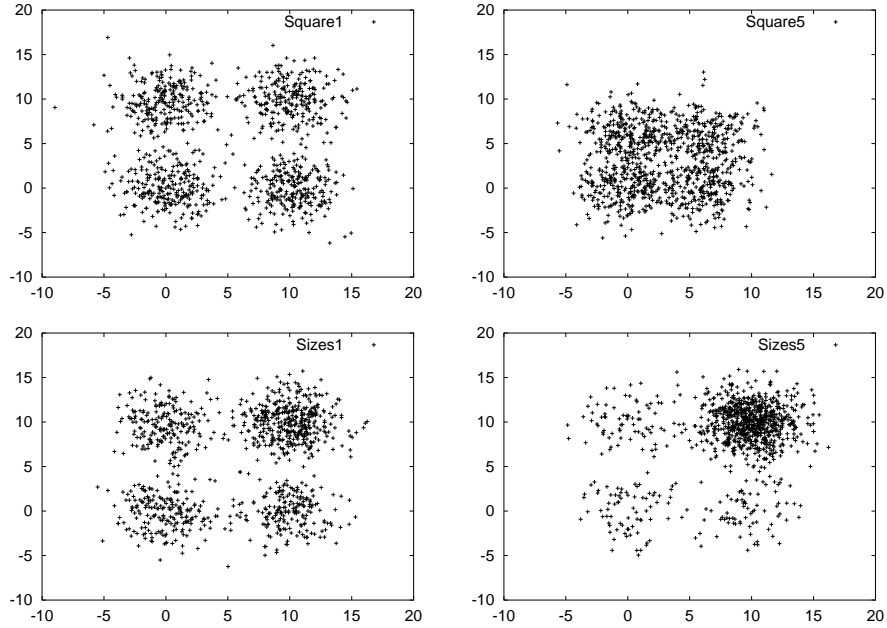


Fig. 1. Four sample instances of the *Square1*, the *Square5*, the *Sizes1* and the *Sizes5* benchmark data.

the *Sizes1* (where it is 2) to the *Sizes5* data (where it is 10). This is used to investigate the algorithms' sensitivity to unequally-sized clusters.

Data preprocessing. Prior to clustering, the data is normalised in each dimension. The Euclidean distance is used as a distance measure between individual data items, and, for average-link agglomerative clustering and ant-based clustering the complete dissimilarity matrix is precomputed.¹ The entries of the matrix are normalised to lie within the interval $[0, 1]$.

Analytical evaluation. The analytical evaluation of the performance of ant-based clustering requires that the solution generated by the algorithm be made explicit, that is, that the spatial distribution produced by the algorithm be converted to an explicit partitioning of the data. In our experiments this was done using the automated method described in [3].

We evaluate the obtained partitioning using the F -Measure [10], which combines information on the purity and the completeness of the generated clusters

¹ Note that this is not possible for k -means and 1D-SOM, as these work with cluster representatives that do not necessarily correspond to actual data items within the collection and can change in each iteration.

Table 1. Summary of the used data sets. dim is the dimensionality, k gives the number of clusters, and n_j gives the number of data elements for cluster C_j . The test sets are generated by multidimensional normal distributions $N(\boldsymbol{\mu}, \boldsymbol{\sigma})$, where $\boldsymbol{\mu}$ is the vector of means and $\boldsymbol{\sigma}$ is the vector of the standard deviations.

Name	k	n_j	dim	Source
Square1	4	4×250	2	$N([0, 0], [2, 2])$, $N([10, 10], [2, 2])$ $N([0, 10], [2, 2])$, $N([10, 0], [2, 2])$
Square2	4	4×250	2	$N([0, 0], [2, 2])$, $N([9, 9], [2, 2])$ $N([0, 9], [2, 2])$, $N([9, 0], [2, 2])$
Square3	4	4×250	2	$N([0, 0], [2, 2])$, $N([8, 8], [2, 2])$ $N([0, 8], [2, 2])$, $N([8, 0], [2, 2])$
Square4	4	4×250	2	$N([0, 0], [2, 2])$, $N([7, 7], [2, 2])$ $N([0, 7], [2, 2])$, $N([7, 0], [2, 2])$
Square5	4	4×250	2	$N([0, 0], [2, 2])$, $N([6, 6], [2, 2])$ $N([0, 6], [2, 2])$, $N([6, 0], [2, 2])$
Square6	4	4×250	2	$N([0, 0], [2, 2])$, $N([5, 5], [2, 2])$ $N([0, 5], [2, 2])$, $N([5, 0], [2, 2])$
Square7	4	4×250	2	$N([0, 0], [2, 2])$, $N([4, 4], [2, 2])$ $N([0, 4], [2, 2])$, $N([4, 0], [2, 2])$
Sizes1	4	400, 200, 200, 200	2	$N([0, 0], [2, 2])$, $N([10, 10], [2, 2])$ $N([0, 10], [2, 2])$, $N([10, 0], [2, 2])$
Sizes2	4	571, 143, 143, 143	2	$N([0, 0], [2, 2])$, $N([10, 10], [2, 2])$ $N([0, 10], [2, 2])$, $N([10, 0], [2, 2])$
Sizes3	4	667, 111, 111, 111	2	$N([0, 0], [2, 2])$, $N([10, 10], [2, 2])$ $N([0, 10], [2, 2])$, $N([10, 0], [2, 2])$
Sizes4	4	727, 91, 91, 91	2	$N([0, 0], [2, 2])$, $N([10, 10], [2, 2])$ $N([0, 10], [2, 2])$, $N([10, 0], [2, 2])$
Sizes5	4	769, 77, 77, 77	2	$N([0, 0], [2, 2])$, $N([10, 10], [2, 2])$ $N([0, 10], [2, 2])$, $N([10, 0], [2, 2])$

with respect to the real class memberships. In particular, it adopts the ideas of precision and recall from information retrieval: Each *class* T_i (inherent to the data) is regarded as the set of n_i items desired for a query; each *cluster* C_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 T_i within cluster C_j . For each class T_i and cluster C_j precision and recall are then defined as $p(i, j) = \frac{n_{ij}}{n_j}$ and $r(i, j) = \frac{n_{ij}}{n_i}$, respectively, 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 equal weighting for $p(i, j)$ and $r(i, j)$ is obtained if $b = 1$. The overall F-value for the partitioning is computed as

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

It is limited to the interval $[0, 1]$ and should be maximised.

5 Results

We now summarise the results obtained in our comparison of the ant-based clustering algorithm with the standard clustering techniques k -means, average-link agglomerative clustering and 1D-SOM. While, in this paper, we limit the discussion to the qualitative performance of ant-based clustering on the two types of synthetic data presented in the above section, results for more general synthetic and real data sets (including runtimes) can be found in [3].

Sensitivity to overlapping clusters. We study the sensitivity to overlapping clusters using the *Square1* to *Square7* data sets. It is clear that the performance of all four algorithms necessarily has to decrease with a shrinking distance between the clusters, as points within the region of overlap cannot be correctly classified. It is however interesting to see whether the performance of the individual algorithms degrades gracefully or more catastrophically, as a graceful degradation would indicate that the main cluster structures are still correctly identified.

Figure 2a shows a plot of the algorithms' performance (as reflected by the F-measure) versus the distance between neighbouring clusters. A number of trends can be observed in this graph. There is the very strong performance of k -means, which performs best on the first four data sets. The 1D-SOM starts on a lower quality level, but its relative drop in performance is less than that of k -means: it clearly profits from its topology preserving behaviour, which makes it less susceptible to noise. Average-link agglomerative clustering, in contrast, has trouble in identifying the principal clusters and performs quite badly, especially on the data sets with a lower inter-cluster distance.

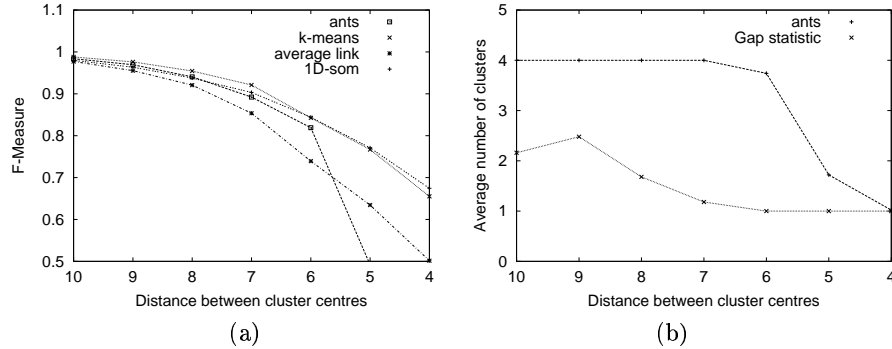


Fig. 2. Performance as a function of the distance between the cluster centres on the *Square* data sets. (a) F-Measure (b) Number of identified clusters.

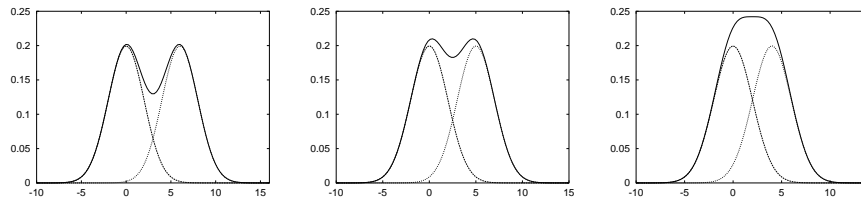


Fig. 3. Theoretical density distribution along the connecting line between two cluster centres in the *Square5*, the *Square6* and the *Square7* test set (from left to right). Constructed as the superimposition of two one-dimensional normal distributions with standard deviation 2 and a distance of 6, 5 and 4 respectively.

The results of ant-based clustering are very close to those for *k*-means on the simplest data set, *Square1*, but its performance drops slightly more quickly. Still, it performs significantly better than average-link agglomerative clustering on the first five test sets. Also, in spite of the fact that the clusters ‘touch’, the ant-algorithm reliably identifies the correct number of clusters on the first five test sets, and it can thus be concluded that the algorithm does not rely on the *spatial separation* between clusters, but that distinct changes in the *density distribution* are sufficient for it to detect the clusters.

For the *Square6* and *Square7* test data, the performance of ant-based clustering drops significantly, as it fails to reliably detect the four clusters. For the *Square6* test set the number of identified clusters varies between 1 and 4, for the *Square7* only 1 cluster is identified (see Figure 2b). However, a plot of the theoretical density distribution along the ‘edge’ between two neighbouring clusters in this data set, puts this failure into perspective: Figure 3 makes clear, that, due

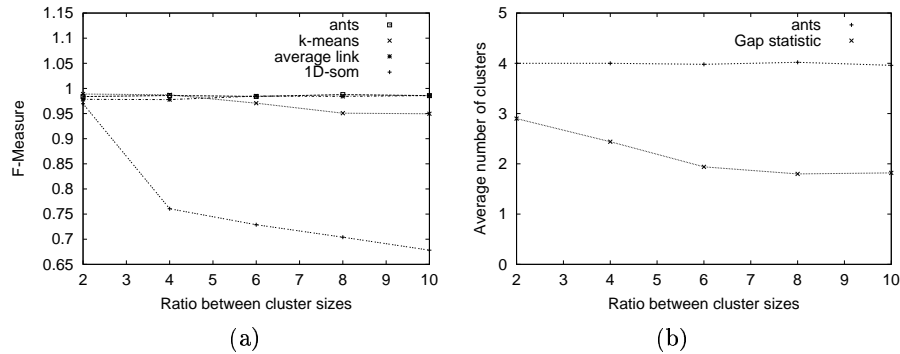


Fig. 4. Performance as a function of the ratio between cluster sizes. (a) F-Measure (b) Number of identified clusters.

to the closeness of the clusters, the density gradient is very weak for the *Square6* data, and the distribution of data items is nearly uniform for the *Square7* data.

The reader should keep in mind that, different from its competitors, ant-based clustering has not been *provided* with the correct number of clusters. In order to get a more precise idea of the performance of ant-based clustering, we therefore additionally analyse its success at identifying the correct number of clusters in the data. The comparison in Figure 2b shows that ant-based clustering performs very well, it is much less affected by the lack of spatial separation between the clusters than the Gap statistic.

Sensitivity to unequally-sized clusters. The sensitivity to unequally-sized clusters is studied using the *Sizes1* to *Sizes5* data sets. Again, we show the algorithms' performance on these data sets as reflected by the F-Measure (Figure 4).

Ant-based clustering performs very well on all five test sets, in fact it is hardly affected at all by the increasing deviations between cluster sizes. Out of its three contestants, only average-link agglomerative clustering performs similarly robustly. 1D-SOM is very strongly affected by the increase of the ratio between cluster sizes, and the performance of *k*-means also suffers. The performance of the Gap statistic is again very weak when compared to ant-based clustering.

6 Conclusion

In this paper we have introduced algorithmic modifications and parameter settings for ant-based clustering that permit its direct application to arbitrary numerical data sets. While the robust performance of the algorithm across a wide range of test data has been demonstrated elsewhere [3], our analysis in this paper has focused on studying two particular data properties that can pose problems to clustering algorithms.

The results presented demonstrate the robust performance of ant-based clustering. The algorithm is largely unaffected by data sets in which the clusters are unequally sized, and it succeeds at reliably separating clusters up to a high degree of overlap. In both cases, it clearly outperforms the Gap statistic at identifying the correct number of clusters.

Acknowledgements

Julia Handl is supported by a DAAD scholarship, and thanks Prof. Günther Görz, Universität Erlangen-Nürnberg, for his supervision. Joshua Knowles gratefully acknowledges the support of a CEC Marie Curie Fellowship, contract: HPMF-CT-2000-00992 (to September 2003) and a BBSRC David Phillips Fellowship (from October 2003). Marco Dorigo acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate, and from the “ANTS” project, an “Action de Recherche Concertée” funded by the Scientific Research Directorate of the French Community of Belgium.

References

1. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence – From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.
2. J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In J.-A. Meyer and S. Wilson, editors, *Proceedings of the First International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 1*, pages 356–365. MIT Press, Cambridge, MA, 1991.
3. J. Handl. Ant-based methods for tasks of clustering and topographic mapping: extensions, analysis and comparison with alternative methods. Masters thesis. Chair of Artificial Intelligence, University of Erlangen-Nuremberg, Germany. November 2003. <http://www.handl.julia.de>.
4. J. Handl and B. Meyer. Improved ant-based clustering and sorting in a document retrieval interface. In *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature*, volume 2439 of *LNCS*, pages 913–923. Springer-Verlag, Berlin, Germany, 2002.
5. T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Germany, 1995.
6. E. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3*, pages 501–508. MIT Press, Cambridge, MA, 1994.
7. L. MacQueen. Some methods for classification and analysis of multivariate observations. In L. LeCam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, Berkeley, 1967.
8. G.W. Milligan. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179, 1985.
9. R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a dataset via the Gap statistic. Technical Report 208, Department of Statistics, Stanford University, 2000. <http://citeseer.nj.nec.com/tibshirani00estimating.html>.

10. C. van Rijsbergen. *Information Retrieval, 2nd edition*. Butterworths, London, UK, 1979.
11. J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parkankangas. SOM Toolbox for Matlab 5. Technical Report A57, Neural Networks Research Centre, Helsinki University of Technology, Espoo, Finland, April 2000.
12. E. Vorhees. *The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval*. PhD thesis, Department of Computer Science, Cornell University, UK, 1985.