Ant-based clustering: a comparative study of its relative performance with respect to k-means, average link and 1d-som

J. Handl, J. Knowles and M. Dorigo
IRIDIA, Université Libre de Bruxelles
jhandl@iridia.ulb.ac.be, {jknowles,mdorigo}@ulb.ac.be

Abstract. Ant-based clustering and sorting is a nature-inspired heuristic for general clustering tasks. It has been applied variously, from problems arising in commerce, to circuit design, to text-mining, all with some promise. However, although early results were broadly encouraging, there has been very limited *analytical* evaluation of ant-based clustering. Toward this end, we first propose a scheme that enables unbiased interpretation of the clustering solutions obtained, and then use this to conduct a full evaluation of the algorithm. Our analysis uses three sets each of real and artificial data, and four distinct analytical measures. These results are compared with those obtained using established clustering techniques and we find evidence that ant-based clustering is a robust and viable alternative.

1 Introduction

Ant-based clustering and sorting [4] was inspired by the clustering of corpses and larval-sorting activities observed in real ant colonies [3]. The algorithm's basic principles are straightforward [16]: Ants are modelled by simple agents that randomly move in their environment, a square grid with periodic boundary conditions. Data items that are scattered within this environment can be picked up, transported and dropped by the agents. The picking and dropping operations are biased by the similarity and density of data items within the ants' local neighbourhood: ants are likely to pick up data items that are either isolated or surrounded by dissimilar ones; they tend to drop them in the vicinity of similar ones. In this way, a clustering and sorting of the elements on the grid is obtained. Hence, like ant colony optimisation (ACO, [6]), ant-based clustering and sorting is a distributed process that employs positive feedback. However, in contrast to ACO, no artificial pheromones are used; instead, the environment itself serves as stigmergic variable [5].

One of the algorithm's particular features is that it generates a clustering of a given set of data through the embedding of the high-dimensional data items on a two-dimensional grid; it has been said to perform both a vector quantisation and a topographic mapping at the same time, much as do self-organising feature maps (SOMs, [12]). While this would be an attractive property, (i) there has been little thorough analysis of the algorithm's actual performance in terms of *clustering* and (ii) it is unclear to what degree the *sorting* really is topology-preserving (cf. [10]).

In this paper, we address only the *first* of the above issues: *ant-based clustering*, that is, the algorithm's application to *pure cluster analysis*. Towards this goal we present:

- A method for the determination of suitable parameter settings across different test sets.
- A technique to convert the spatial embedding generated by the ant algorithm, which *implicitly* contains clusters, to an *explicit* partitioning of the data set.
- Results on synthetic and real data sets. Evaluation is done using four different analytical evaluation measures. We compare the outcome to k*-means*, *hierarchical agglomerative clustering* based on the linkage metric of *average link* and *one-dimensional self-organising maps*.

The remainder of this paper is structured as follows. Section 2 briefly introduces the problem domain and Section 3 reviews previous work on ant-based clustering. Section 4 presents the algorithms used within our comparative study, that is, the necessary extensions to ant-based clustering are introduced, and details on the implementation of k-means, the agglomerative clustering scheme and the self-organising map are given. The employed test data and evaluation measures are explained in Section 5. Results are presented and discussed in Section 6, and Section 7 concludes.

2 Clustering

Clustering is concerned with the division of data into homogenous subgroups. Informally, the objective of this division is twofold: data items within one cluster are required to be similar to each other, while those within different clusters should be dissimilar. Problems of this type 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 number of clusters).

The four main classes of clustering algorithms available in the literature are *partitioning methods*, *hierarchical methods*, *density-based clustering* and *grid-based clustering* (see [1] for an extensive survey). For the purpose of our comparative study we select two of the most popular and well-studied algorithms from the literature: k-means, a representative of the class of partitioning methods, and agglomerative average link clustering, which is a hierarchical approach. Additionally, we compare against one-dimensional self-organising maps. All three algorithms are described in more detail in Section 4.

3 Ant-based clustering in the literature

Ant-based clustering and sorting was originally introduced for tasks in robotics by Deneubourg [4]. Lumer and Faieta [16] modified the algorithm to extend to numerical data analysis and it has subsequently been used for data-mining [17], graph-partitioning [15, 14, 13] and text-mining [11, 20, 10].

While many of the obtained results ‘look’ promising, there is a lack of knowledge on the actual clustering performance of the algorithm. The evaluation of the results has, to a

large extent, been based on visual observation. Analytical evaluation has mainly been used to track the *progress* of the clustering process, using evaluation functions (grid entropy and local dissimilarity) that provide only very limited information on the *overall quality* of the clustering and the sorting obtained [4, 16, 13, 20]. More enhanced analytical evaluation measures have been employed in [14] and [10] (non-metric stress and Pearson correlation), but their focus is on measuring the degree of *topology-preservation*, not the *clustering quality*. Classical evaluation measures for cluster analysis have been used in a single paper [11] (cluster entropy and purity). Unfortunately, Hoe et al. merely provide results for one single run on a small document collection. Only for the specialised application of graph-partitioning are analytical results (error rate and inter-vertex cuts) on a family of pseudo-random graphs and the corresponding values for the classical Fiduccia-Mattheyses [7] heuristic, available [15]. Additionally, the range of benchmark data sets employed has been very limited. Apart from the pseudo-random graphs used by Kuntz et al. [15], one rather simple synthetic data set has been used in most of the work.

Note that Monmarché has introduced an interesting hybridisation of ant-based clustering and the k-means algorithm and compared it to traditional k-means on various data sets, using the classification error for evaluation purposes [18]. However, the results obtained with this method are not applicable to ordinary ant-based clustering since it differs significantly from the latter.

Looking at the previous evaluations of ant-based clustering, one might be inclined to ask the following two questions:

(1) Why have evaluation measures for cluster analysis found so little application?

Certainly not for a lack of measures, as the data-mining literature provides a large pool of functions for the evaluation of partitionings both when the real cluster structure is known and when it is unknown (cf. [8] for a survey). The main problem is one related to the nature of the algorithm's outcome: it does not generate an *explicit partitioning* but a *spatial distribution* of the data elements. While this may contain clusters 'obvious' to the human observer, an evaluation of the clustering performance requires the retrieval of these clusters, and it is not trivial to do this without human interaction (e.g., 'marking' of the clusters) and without distorting the resulting partitioning (e.g., by assuming that the correct number of clusters has been identified).

(2) Why has evaluation been limited to such a small range of benchmark data?

This might be due, at least partly, to the difficulty of devising appropriate parameter settings for ant-based clustering on different types of data, which makes its application to new data sets rather tedious.

A rigorous evaluation of the performance of ant-based clustering requires a solution to both of the above problems.

4 Algorithms

We will now detail the four algorithms used in our comparative study, starting with a short description of the ant-based algorithm, and the methods developed to automatically determine parameter settings and to retrieve clusters from the grid.

4.1 Ant-based clustering

The ant algorithm is mainly based on the version described in [10]. A number of slight modifications have been introduced that improve the quality of the clustering and, in particular, the spatial separation between clusters on the grid, which is essential for the scheme of cluster retrieval introduced below. Results on the qualitative performance gains afforded by these extensions are provided in [9].

4.2 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.

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

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

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

where $f^*(i)$ is a modified version of Lumer and Faieta's [16] neighbourhood function:

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

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 4.4 below).

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 := random_select(remaining_items)
6:    pick_up(agent(j), i)
7:    g := random_select(remaining_empty_grid_locations)
8:    place_agent(agent(j), g)
9: end for
10: MAIN LOOP
11: for each it_ctr in 1 to #iterations do
12:    j := random_select(all_agents)
13:    step(agent(j), stepsize)
14:    i := carried_item(agent(j))
15:    drop := drop_item?(f*(i)) // see equations 2 and 3
16:    if drop = TRUE then
17:       while pick = FALSE do
18:          i := random_select(free_data_items)
19:          pick := pick_item?(f*(i)) // see equations 1 and 3
20:       end while
21:    end if
22: end for
23: end
```

4.3 Short-term memory

A modified version of the 'short-term memory' introduced by Lumer and Faieta in [16] 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.

4.4 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 [16, 10]. 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 3 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 4.2. 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 1 and 2) 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.

4.5 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 3, 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.

4.6 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 $0.55 \cdot \#iterations$.

4.6.1 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. 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.

4.6.2 Activity-based α-adaptation.

An issue already addressed in [10] 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.

4.6.3 Automatic cluster retrieval

In order to make the clusters identified by ant-based clustering explicit, we apply an agglomerative hierarchical clustering algorithm to the positions of the data items on the grid. The algorithm starts by assigning each data item on the grid to an individual cluster, and proceeds by merging the two least distant clusters (in terms of spatial distance on the grid) in each iteration, until a stopping criteria is met.

Given two clusters C_i and C_j on the grid and, without loss of generality, $|C_i| \leq |C_j|$, we define their spatial distance in grid-space as

$$weighted\_singlelink(C_i, C_j) = singlelink(C_i, C_j) \cdot weight(C_i, C_j).$$

Here, $singlelink(C_i, C_j)$ is the standard *single link* linkage metric, i.e. the minimal distance between all possible pairs of data elements i and j with $i \in C_i$ and $j \in C_j$. In our case, the distance between two data elements is given by the Euclidean distance between their *grid positions*. The term $weight(C_i, C_j)$ is an additional scaling factor taking the relative sizes of the clusters into account:

$$weight(C_i, C_j) = 1.0 + \log_{10}(1.0 + 9.0 \cdot \frac{|C_i|}{|C_j|})$$

Clearly, $weight(C_i, C_j)$ is restricted to the range $(1, 2]$.

The spatial distribution generated by our version of the ant-based clustering algorithm has two interesting features, which are necessary for the robust performance of this retrieval

scheme. These are, firstly, the high compactness of the clusters on the grid and, secondly, the clear spatial separation between the individual clusters. Given these properties, an agglomerative algorithm based on the single link criterion will clearly work very well. However, as data items around the cluster borders are sometimes slightly isolated (so that they are prone to mislead the single link metric by establishing individual clusters or 'bridging' gaps between clusters), we have introduced the additional weighting term $weight(C_i, C_j)$ that encourages the merging of these elements with the 'core' clusters.

The radius used within the last phase of the ant-based algorithm signifies the minimum distance clusters should have on the grid, and therefore provides a suitable stopping criterion for the agglomerative clustering algorithm. The merging of clusters stops once all the distances between the clusters have reached a value larger than this radius.

4.7 k-means

The first algorithm we compare against is the well-known k-means algorithm. Starting from a random partitioning, the algorithm repeatedly (i) computes the current cluster centres (i.e. the average vector of each cluster in data space) and (ii) reassigns each data item to the cluster whose centre 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 centres, is locally minimised. k-means' strength is its runtime, which is linear in the number of data elements, and its ease of implementation. However, the algorithm tends to get stuck in suboptimal solutions (dependent on the initial partitioning and the data ordering) and it works well only for spherically shaped clusters. It requires the number of clusters to be provided or to be determined (semi-) automatically.

In our experiments, we run k-means using the correct cluster number k. Random initialisation is used and the best result out of 10 runs (in terms of minimum variance) is selected in order to reduce the effects of local optima.

4.8 Average link

As a second method, an agglomerative hierarchical clustering algorithm based on the *average link* linkage metric is used. The algorithm starts with the finest partitioning possible (i.e. singletons) and, in each iteration, merges the two least distant clusters. 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 $j \in C_j$. Hierarchical clustering methods are thought to give higher quality solutions than partitioning methods. However, their runtime scales quadratically and results heavily depend on the linkage metric used. The derivation of appropriate stopping criteria can be difficult, if the correct cluster number is not known.

As for k-means, we provide the correct cluster number. The Ward updating formula [19] is used to efficiently recompute cluster distances.

4.9 One-dimensional self-organising maps

The implementation of 1D-SOM is based on the guidelines given in the description of the SOM Toolbox [21]. The correct number of clusters k is used to set the grid resolution to $1 \times k$ (rectangular grid cells); the weight vectors are uniformly randomly initialised. The SOM is

trained in two training phases, a first 'coarse' approximation phase and a second fine-tuning phase. The first phase starts with a neighbourhood size of $ns_1^{start} = \max(1.0, \frac{1}{4}k)$, which is exponentially decreased to $ns_1^{end} = \max(1.0, \frac{1}{4}ns_1^{start})$. The learning rate during this phase is $lr_1 = 0.5$. The second phase starts with the final neighbourhood size of phase 1, that is, $ns_2^{start} = ns_1^{end}$, and continues to decrease it to $ns_2^{end} = 1.0$. The learning rate in phase 2 is $lr_2 = 0.05$. The number of iterations for each phase are $it_1 = 10$ and $it_2 = 40$ respectively, and, in each iteration, all data items are presented to the SOM in random order. Finally, in the classification step all data items are assigned to the best matching output neuron. Each output neuron is interpreted as one cluster.

5 Evaluation

Comparative results are presented for three synthetic test sets and three real data collections taken from the Machine Learning Repository [2], which we describe below. In a preprocessing step, the data vectors are normalised in each dimension, and, for the ant-based algorithm and agglomerative clustering, all pairwise dissimilarities are precomputed. The employed distance function is the Euclidean distance[1] for the synthetic data sets, and the Cosine measure[2] for the real data sets.

5.1 Synthetic data

The *Square1* data set is the type of data most frequently used within previous work on ant-based clustering. It is two-dimensional and consists of four clusters arranged as a square. They are generated by the Normal Distributions $(N(-5,2), N(-5,2))$, $(N(5,2), N(5,2))$, $(N(-5,2), N(5,2))$ and $(N(5,2), N(-5,2))$, and are each of size 250. Hence, *Square1* contains four spherically shaped clusters with equal spread. In the experiments, a new sample is generated from these distributions in each run.

$2D$-$4C$ and $100D$-$10C$ are two examples of a range of benchmark test sets generated and used for our comparative study. Every set xD-yC (where x describes the dimensionality of the data and y gives the number of clusters) consists of 50 different instances, and we generate each individual instance 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 $2D$-$4C$ and $100D$-$10C$ is 1000 and 2500 data items respectively. During the generation process, cluster centres are rejected if the resulting distributions would have more than 3% overlap. A different instance is used in each individual run of the experiments.

[1] The synthetic test sets are generated in Euclidean space and the Euclidean distance is therefore the appropriate measure to use.

[2] On the real data sets taken from the Machine Learning Repository the Cosine measure outperforms the Euclidean distance and is commonly used.

5.2 *Real data*

The *Iris* data contains 150 items described by 4 attributes. Its 3 clusters are each of size 50. Two of them are linearly non-separable. The *Breast Cancer Wisconsin* data contains 699 items described by 9 attributes. Its 2 clusters are of size 458 and 241 respectively. The *Yeast* data contains 1484 data elements described by 8 attributes. Its 10 clusters are of size 463, 429, 244, 163, 51, 44, 35, 30, 200 and 5. The real data sets are arbitrarily permuted in each run.

5.3 *Evaluation functions*

The clustering results of the different algorithms on the test sets are compared using four different evaluation measures. The first two of them make use of the correct classification, which is known for all six data sets.

- The *F-Measure* uses 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-value 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 maximised.

- The *Rand Index* determines the degree of similarity (in terms of pairwise co-assignments) between the known correct classification U and the solution V generated by a clustering algorithm. It is defined as

  $$R = \frac{a+d}{a+b+c+d},$$

  where a, b, c and d are computed for all possible pairs of data points i and j and their respective cluster assignments $c_U(i), c_U(j), c_V(i)$ and $c_V(j)$:

  $$a = |\{i,j | c_U(i) = c_U(j) \wedge c_V(i) = c_V(j)\}|, \quad b = |\{i,j | c_U(i) = c_U(j) \wedge c_V(i) \neq c_V(j)\}|,$$

  $$c = |\{i,j | c_U(i) \neq c_U(j) \wedge c_V(i) = c_V(j)\}|, \quad d = |\{i,j | c_U(i) \neq c_U(j) \wedge c_V(i) \neq c_V(j)\}|.$$

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

- The *Inner Cluster Variance* computes the sum of squared deviations between all data items and their associated cluster centre, which reflects the common agreement that data elements within individual clusters must be similar. It is given by

$$I = \sum_{c \in C} \sum_{i \in c} \delta(i, \mu_c)^2,$$

  where C is the set of all clusters, μ_c is the centroid of cluster c and $\delta(i, \mu_c)$ is the distance function employed to compute the deviation between each data item i and its associated cluster centre. It is to be minimised.

- The *Dunn Index* determines the minimal ratio between cluster diameter and inter cluster distance for a given partitioning. Thus, it captures the notion that, in a good clustering solution, data elements within one cluster should be much closer than those within different clusters. It is defined as

$$D = \min_{c,d \in C} \left[\frac{\delta(\mu_c, \mu_d)}{\max_{e \in C} [diam(e)]} \right],$$

  where the diameter $diam(c)$ of a cluster c is computed as the maximum inner cluster distance, and $\delta(\mu_c, \mu_d)$ is the distance between the centroids of clusters c and d. It is to be maximised.

6 Results

Table 1 gives the means and standard deviations (over 50 runs) obtained for each of these measures. Additionally, it shows the average number of clusters (which is automatically determined only for the ant algorithm) and the algorithms' average runtimes. Note that the timings reported for the ant algorithm do *not* include the time required for cluster retrieval. This is justified as the cluster retrieval is a tool *only* necessary for the analysis process. In a typical application, the user would be presented with the visual representation and, in an interactive setting, clusters could then be identified with hardly any computational overhead.

6.1 Number of clusters

The results demonstrate that, if clear cluster structures exist within the data, the ant algorithm is quite reliable at identifying the correct number of clusters. This is the case both if the clusters are spatially well separated (as is the case in the test sets $2D$-$4C$, and $10D$-$4C$) and also if they touch but show clear density gradients (in *Square1*). Only on the *Yeast* data the detected number of clusters is far too low. However, the F-Measure reveals that all four algorithms perform very badly on this data, showing that k-means, agglomerative clustering and 1D-SOM equally fail to correctly identify the clusters in spite of the fact that they have a priori knowledge of the correct number. This is an indication that the structure within the data is not very pronounced.

Table 1: Results for k-means, hierarchical agglomerative average-link clustering, one-dimensional self-organising maps and ant-based clustering on three synthetic and three real data sets. The quality of the partitioning is evaluated using the F-Measure, the Rand Index, the Inner Cluster Variance and the Dunn Index. Runtimes and the number of identified clusters (which is automatically determined only for the ant algorithm) are additionally provided. The table shows means and standard deviations (in brackets) for 50 independent runs. Underlined bold face indicates the best, bold face the second best and italic face the third best result out of the four algorithms.

Square1	k-means	average link	1D-SOM	ant-based clustering
#Clusters	4 (0)	4 (0)	4 (0)	4 (0)
F-Measure	**0.987059** (0.00409363)	*0.980654* (0.0076673)	0.977711 (0.0047729)	**0.982561** (0.00518902)
Rand Index	**0.987212** (0.00398614)	*0.981053* (0.00730242)	0.978157 (0.00459841)	**0.982828** (0.00503838)
Variance	**0.461473** (0.0060167)	0.465201 (0.0086531)	*0.464669* (0.00675606)	**0.463618** (0.00883593)
Dunn Index	**3.78352** (0.105684)	3.22921 (0.298599)	*3.30064* (0.319073)	**3.64224** (0.249045)
Runtime	**1.26** (0.795236)	**4.24** (0.427083)	*5.48* (0.4996)	10.14 (1.74367)
2D-4C	k-means	average link	1D-SOM	ant-based clustering
#Clusters	4 (0)	4 (0)	4 (0)	4 (0.282843)
F-Measure	*0.972734* (0.0740772)	**0.997365** (0.018445)	0.953895 (0.0560186)	**0.990371** (0.0354898)
Rand Index	*0.983066* (0.0464064)	**0.998241** (0.012311)	0.971536 (0.036236)	**0.99155** (0.031725)
Variance	*0.177598* (0.16193)	**0.127346** (0.0372185)	0.199594 (0.103416)	**0.133784** (0.062165)
Dunn Index	*4.45335* (2.09255)	**5.10569** (1.61428)	3.68615 (2.12704)	**5.02245** (1.78885)
Runtime	**0.74** (1.09197)	**6.8** (4.36807)	*7.64* (3.39859)	15.68 (6.60739)
10D-10C	k-means	average link	1D-SOM	ant-based clustering
#Clusters	10 (0)	10 (0)	10 (0)	10 (0)
F-Measure	*0.971745* (0.0410038)	**1.0** (0.0)	0.956233 (0.0283843)	**0.999986** (0.0000960888)
Rand Index	*0.993207* (0.0118778)	**1.0** (0.0)	0.990942 (0.00765172)	**0.999995** (0.0000318648)
Variance	*0.411021* (0.160722)	**0.331714** (0.0305796)	0.517608 (0.119397)	**0.331777** (0.0305041)
Dunn Index	*8.22036* (6.67596)	**13.6787** 1.90435)	4.18333 (4.70773)	**13.445** (2.36789)
Runtime	**46.04** (13.1559)	83.26 (33.2955)	*54.64* (19.1141)	**19.2** (5.5715)
IRIS	k-means	average link	1D-SOM	ant-based clustering
#Clusters	3 (0)	3 (0)	3 (0)	3.02 (0.14)
F-Measure	**0.824521** (0.0848664)	0.809857 (0.0)	**0.861415** (0.00773914)	*0.816812* (0.0148461)
Rand Index	0.816599 (0.101288)	*0.822311* (0.0)	**0.857342** (0.00554881)	**0.825422** (0.00804509)
Variance	0.922221 (0.221044)	*0.900175* (0.0)	**0.894906** (0.00147313)	**0.880333** (0.00405812)
Dunn Index	**2.65093** (0.4201)	*2.5186* (0.0)	2.07881 (0.253375)	**2.9215** (0.298826)
Runtime	*0.16* (0.366606)	**0.02** (0.14)	**0.08** (0.271293)	3.36 (0.48)
WISCONSIN	k-means	average link	1D-SOM	ant-based clustering
#Clusters	2 (0)	2 (0)	2 (0)	2 (0)
F-Measure	*0.965825* (0.0)	**0.965966** (0.0)	0.965766 (0.000861166)	**0.967604** (0.00144665)
Rand Index	**0.933688** (0.065321)	**0.933688** (0.0)	0.933583 (0.00159676)	**0.93711** (0.00273506)
Variance	**1.61493** (0.0)	1.63441 (0.0)	*1.61494* (0.000870785)	**1.61257** (0.000838131)
Dunn Index	**5.47121**(0.000178411)	4.91649 (0.000000284355)	**5.45811** (0.015952)	*5.4424* (0.0957218)
Runtime	**0.06** (0.237487)	**1.44** (0.496387)	*2.32* (0.466476)	10.54 (0.498397)
YEAST	k-means	average link	1D-SOM	ant-based clustering
#Clusters	10 (0)	10 (0)	10 (0)	5.36 (1.17915)
F-Measure	*0.431505* (0.00443954)	**0.448316** (0.0)	0.406728 (0.0174483)	**0.435396** (0.0345797)
Rand Index	**0.750657** (0.00124985)	*0.742682* (0.0)	**0.75227** (0.00267426)	0.678131 (0.0752791)
Variance	**1.53798,** (0.001611)	**1.60028** (0.00000000652216)	*1.69317* (0.0201931)	1.89537 (0.115468)
Dunn Index	**1.69692** (0.109608)	*1.55563* (0.000000120366)	1.22087 (0.154187)	**1.88049** (0.207959)
Runtime	**1.7** (1.0247)	14.04 (0.0)	*12.16* (0.417612)	**9.22** (0.54)

6.2 Solution quality

With the exception of the *Yeast* data set, ant-based clustering performs very well under all four measures. On the synthetic benchmarks, it always comes second best and is very close to the best solution. On the first two real data benchmarks it even shows the highest performance under some measures.

It is interesting to note that on both the $2D$-$4C$ and the $10D$-$10C$ test data, where both agglomerative clustering and ant-based clustering perform consistently well, 1D-SOM and k-means repeatedly generate very bad solutions. To k-means this happens in spite of the fact that its solution is already the best selected out of 20 runs.

6.2.1 Runtime

For the small data collections, the ant algorithm's runtimes are considerably higher than those of k-means, hierarchical clustering and 1D-SOM. However, it is worth observing that they scale linearly, such that ant-based clustering already outperforms the agglomerative scheme on the *Yeast* data set. As both k-means and 1D-SOM are affected by the rise in dimensionality, ant-based clustering even becomes the fastest method on the $10D$-$2D$ data.

7 Conclusion

In contrast to previous studies on ant-based clustering and sorting, we have studied the algorithm's clustering solutions *in isolation*, in order to obtain an improved understanding of its performance relative to other methods for clustering. Nonetheless, it should not be forgotten that the mapping *simultaneously* provided by the algorithm is one of its most attractive features. To what extent this mapping really is (or can be improved to be) topology-preserving is investigated in [9].

Seen *purely* as a clustering algorithm, ant-based clustering performs well in our comparison to the popular methods of k-means, agglomerative hierarchical clustering and one-dimensional self-organising maps. In addition to that, the algorithm has a number of features that make it an interesting candidate for cluster analysis. Firstly, there is its linear scaling behaviour, which is attractive for use on large data sets as are frequently encountered today, e.g., in information retrieval. Also, the nature of the algorithm makes it fairly robust to the effects of outliers within the data. In addition, ant-based clustering has the capacity to work with any kind of data that can be described in terms of symmetric dissimilarities, and it imposes no assumption on the shape of the clusters it works with. Finally, an important strength of the algorithm is its ability to automatically determine the number of clusters within the data. The adaptation-scheme proposed in this paper *will* identify an appropriate α-parameter, *if* clear structures exist within the data. While this is an interesting aspect, it is of course not infallible: the scaling parameter α plays a role similar to the density threshold used in density-based clustering, which *implicitly* determines the cluster number. If clusters on several hierarchical levels exist, this version of ant-based clustering will identify the top level ones. The generated clusters could then however be recursively processed.

Acknowledgements

JH is supported by a DAAD scholarship, and thanks Prof. Günther Görz, Universität Erlangen-Nürnberg,

for his supervision. JK 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). MD acknowledges support from the Belgian FNRS, of which he is a Senior Research Associate.

References

[1] P. Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, California, 2002. http://citeseer.nj.nec.com/berkhin02survey.html.

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

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence – From Natural to Artificial Systems*. Oxford University Press, New York, NY, 1999.

[4] 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.

[5] M. Dorigo, E. Bonabeau, and G. Theraulaz. Ant algorithms and stigmery. *Future Generation Computer Systems*, 16(8):851–871, 2000.

[6] M. Dorigo and G. Di Caro. Ant Colony Optimization: A new meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, UK, 1999.

[7] C. Fiduccia and M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the Nineteenth IEEE Design Automation Conference*, pages 175–181. IEEE Press, Piscataway, NJ, 1982.

[8] M. Halkidi, M. Vazirgiannis, and I. Batistakis. Quality scheme assessment in the clustering process. In *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery*, volume 1910 of *LNCS*, pages 265–267. Springer-Verlag, Heidelberg, Germany, 2000.

[9] 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.

[10] 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.

[11] K. Hoe, W. Lai, and T. Tai. Homogeneous ants for web document similarity modeling and categorization. In *Proceedings of the Third International Workshop on Ant Algorithms*, volume 2463 of *LNCS*, pages 256–261. Springer-Verlag, Heidelberg, Germany, 2002.

[12] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Germany, 1995.

[13] P. Kuntz and D. Snyers. Emergent colonization and graph partitioning. In *Proceedings of the Third International Conference on Simulation of Adaptive Behaviour: From Animals to Animats 3*, pages 494–500. MIT Press, Cambridge, MA, 1994.

[14] P. Kuntz and D. Snyers. New results on an ant-based heuristic for highlighting the organization of large graphs. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1451–1458. IEEE Press, Piscataway, NJ, 1999.

[15] P. Kuntz, D. Snyers, and P. Layzell. A stochastic heuristic for visualising graph clusters in a bi-dimensional space prior to partitioning. *Journal of Heuristics*, 5(3):327–351, 1998.

[16] 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.

[17] E. Lumer and B. Faieta. Exploratory database analysis via self-organization, 1995. Unpublished manuscript. Results summarized in [3].

[18] N. Monmarché. *Algorithmes de fourmis artificielles: applications á la classification et á l'optimisation*. PhD thesis, Laboratoire d'Informatique, Université de Tours, France, December 2000.

[19] C. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313–1325, 1995.

[20] V. Ramos and J.J. Merelo. Self-organized stigmergic document maps: Environments as a mechanism for context learning. In *Proceedings of the First Spanish Conference on Evolutionary and Bio-Inspired Algorithms*, pages 284–293. Centro Univ. Mérida, Mérida, Spain, 2002.

[21] 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.