International Journal of Foundations of Computer Science


# Decentralized Packet Clustering in Router-Based Networks

Daniel Merkle, Martin Middendorf, Alexander Scheidler
*Department of Computer Science*
*University of Leipzig*
*Augustusplatz 10-11, D-04109 Leipzig, Germany*
{merkle,middendorf}@informatik.uni-leipzig.de

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Different types of decentralized clustering problems have been studied so far for networks and multi-agent systems. In this paper we introduce a new type of a decentralized clustering problem for networks. The so called Decentralized Packet Clustering (DPC) problem is to find for packets that are sent around in a network a clustering. This clustering has to be done by the routers using only few computational power and only a small amount of memory. No direct information transfer between the routers is allowed. We investigate the behavior of new a type of decentralized $k$-means algorithm — called DPClust — for solving the DPC problem. DPClust has some similarities with ant based clustering algorithms. We investigate the behavior of DPClust for different clustering problems and for networks that consist of several subnetworks. The amount of packet exchange between these subnetworks is limited. Networks with different connection topologies for the subnetworks are considered. A dynamic situation where the packet exchange rates between the subnetworks varies over time is also investigated. The proposed DPC problem leads to interesting research problems for network clustering.

## 1. Introduction

Different types of clustering problems have been studied in the literature where the aim is to find solutions with decentralized algorithms. Such problems occur typically in large networks (e.g., [19]) and multi-agent systems (e.g., [16]) where a central control is not available or should not be established. In this paper we investigate a new scenario for clustering that is relevant for distributed applications in networks. We assume that information packets for some distributed application task that runs on several servers are send around between the servers in a network. Thus, we assume that the network contains of router nodes and server nodes (it is possible that routers are also servers). In each server node an application process is running which has to evaluate the information in the packets that are send to it. When an information packet has been processed by the application process it is send back into the network to be transported to some other server node. In order

that the application process can handle the packets appropriately we assume that the packets are clustered and that each packet contains the number of its cluster. The cluster number is used by the application process to apply a corresponding evaluation function that is adapted to work optimally for packets in that cluster. For example the application process might have a learning ability so that it can handle packets from different clusters differently.

The clustering of the information packets is done according to a data vector that each packet contains. Since the application is distributed over several application processes that run in the servers of the network we assume that there is no central process in the network that knows all information packets and can do the clustering. Hence, we are interested in decentralized solutions for the clustering problem. Since the routers are typically visited by many packets we consider the case that the clustering is done by the routers. The corresponding clustering problem is called here the Decentralized Packet Clustering (DPC) problem. Network router typically receive and send so many packets (from different applications) that they will have not much resources available for other tasks. Therefore, we assume that the routers can store information from at most a few information packets for doing the clustering. Moreover, the routers should not need to spend much computational effort to do the necessary operations for clustering.

In this paper we propose a clustering algorithm called DPClust which can be seen as a form of a distributed $k$-means algorithm since estimated centroids play a role for determining the cluster of an information packet. DPClust has also similarities with a clustering method that is called ANTCLUST which is inspired by the chemical recognition system of ants [11]. In this algorithm artificial ants (which correspond to the packets in our algorithm) meet randomly and exchange odor information which they use to determine whether they are in the same colony (cluster). If this is not the case they decide whether they should change to the other ants colony. The odor information that an ant has is considered as its private odor (its data vector). Each ant has an acceptance threshold that is the difference between the average value of the average similarity to other ants it has met and the maximal similarity to these ants.

DPClust uses meetings of information packets (in the following we call information packets simply packets when the context is clear) in the routers for information exchange and for deciding if a packet changes to the other packets cluster. This decision is based on the information contained in both packets. Each packet contains its cluster number, an estimation of the centroid of its cluster (which can be seen as an average colony odor) and its personal data vector. The estimation of the centroid is computed only from information that is exchanged during the meetings with other packets. Since we do not want to hold packets in the routers for a longer time the routers copy and store the relevant information from the last packets that it has seen.

In this paper we investigate the influence of several network parameters and network topologies on the clustering behavior of DPClust. These parameters are the number of routers in the network, a partitioning of the network into different

numbers of subnetworks, and the amount of information exchange between these subnetworks. Moreover, different topologies for the connection of the subnetworks are considered. In addition and differently to most other works on distributed clustering algorithms, we study also the situation that the characteristics of the packet transfer in the network may change.

The outline of the paper is as follows. In the next Section 2 we give an overview over clustering methods that are relevant for this study. The Decentralized Packet Clustering problem and the DPClust algorithm are explained in Section 3. The experiments are described in Section 4 and the results are presented in Section 5. The paper ends with a conclusion and remarks on future work in Section 6.

## 2. Distributed and Ant-Inspired Clustering

In this section we give a short overview over distributed clustering methods and also discuss some nature inspired agent based clustering methods.

### *2.1. General Distributed Approaches*

Many distributed clustering algorithms assume a coarse grained approach where the clustering problem is partitioned into larger subproblems. Typically, the results of these partial clustering problems or subproblems that have been computed distributed over several processors are then collected and have to be combined to obtain the final clustering. This recombination of the results is often done by a central server. The most prominent types of distributed clustering methods are explained in the following:

- Robust Centralized Clustering (RCC) is to combine different clusterings of the same set of objects. The clustering algorithms might be distributed over several processors but all have access to the whole set of objects and all features of the data. The combination of the clusterings may then be done by distributed processes that have no access to the original data features.
- Feature-Distributed Clustering (FDC) is to combine a set of clusterings that are obtained from clustering algorithms that have only a partial view of the data features.
- Object-Distributed Clustering (ODC) is to combine clusterings obtained from clustering algorithms that have access only to a limited number of objects but to all data features of these objects.

There also exist combinations of the above approaches, e.g. a combination of FDC and ODC where the clustering algorithms know only a subset of the objects and only parts of the data features of these objects. All these approaches are different from our approach since they are not fully decentralized and collect information (partial clusterings) in a central server to obtain the final clustering.

### *2.2. Distributed/Parallel k-Means Approaches*

One of the most often used clustering algorithms is the $k$-means algorithm. This is an iterative algorithm that starts with a set of $k$ initial data vectors, called center points. Each object is assigned to its nearest (measured, e.g., with respect to the Euclidean distance) center point. Then all objects that are assigned to the same center point form a cluster. For each cluster its centroid is computed and these centroids form the new center points for the next iteration of the algorithm. The algorithm stops when some convergence criterion has been met, e.g. the center points have not changed or a maximal number of iterations has been done. The selection of the initial center points has a great influence on the results of the algorithm and different methods have been proposed for choosing these initial points (one is simply to select the center points randomly from the given data vectors).

Several distributed or parallel versions of the $k$-means algorithm have been proposed in the literature. The aim of most of these algorithms is to provide a fast parallel or distributed implementation of $k$-means or one of its variants (e.g. the medoid variants where always a given data vector that is near to the centroid is chosen as center point). The problem is then how to exchange the necessary information between the processors (see e.g. [4]). Since this is different to our approach we do not discuss these algorithms in detail but only describe one example that has been proposed for networks of processors. In [5] an iterative distributed $k$-means algorithm is investigated where the data vectors are distributed between the nodes of a networks. In every iteration the center points are distributed starting from an initiator node to all other nodes in the network. The distribution is done along a peer/echo tree that is determined as a subgraph of the network. The results of the clusterings in the nodes are then collected along the tree. In every node of the tree the received results are merged (i.e. each node echoes for each cluster only one centroid that it has computed as a weighted sum of the centroids it has received).

### *2.3. Special Approaches for Networks*

Various clustering algorithms have been proposed recently for Mobile Ad-Hoc Networks. Other approaches for networks seek to cluster clients in a network (e.g. based on IP numbers [2]) and use large data structures in monitor processes for the clustering. Most of all these works are not relevant here because they consider the special problem to cluster objects that are located in a two-dimensional grid world or use a large amount of memory resources. In the following we review two approaches that are interesting for our application.

In [18] a variant of $k$-means called ISODATA was studied in a distributed mobile network. The given objects are assumed to be distributed between a set of worker nodes. In each iteration a master process distributes a set of center points to the worker processes. Then each worker computes a clustering with its data, determines the centroids for this clustering, and sends the set of centroids and the number of objects in each cluster to the master. The master then computes a new set of centers from these data and the next iteration starts. Different strategies were

proposed how to react when the connection to a worker process gets lost (which is likely in mobile networks). One strategy is to ignore the connection loss and another strategy is to use for the objects of the lost worker node the old values for the centroids. Unfortunately, the few results that are given in [18] do not allow to observe significant differences between the strategies.

A fully-decentralized algorithm for clustering agents that are spread across a network of machines has been proposed in [16]. The algorithm is intended to work within large networks and the main focus of the method is that agents find potential cluster members in a decentralized fashion. Differently to our approach where the packets can move within the network the agents are fixed at their machines. Each agent is represented by a two dimensional data point and the agents seek to group themselves based on the Euclidean distance between their data points. Each agent is initially assigned a small number of random neighbor agents. Based on their local neighborhood the agents form clusters with neighbored agents having the closest data points. Agents within a cluster combine their local views to allow their members search a broader range of neighbors. Since clusters are limited in size by a user-defined parameter a mechanism to split clusters that become too large is integrated into the algorithm.

### *2.4. Ant-Based Approaches*

Some clustering algorithms have been proposed that are inspired by the behavior of ants (see e.g. [3]). Ant-based clustering and sorting (see [3, 14]) is inspired by the particular behavior of real ants to cluster corpses and larvae. The basic idea of ant clustering algorithms is to model ants as simple agents that move randomly within a grid environment. Objects (or data items) are placed randomly on the grid points. An ant that does not carry an object and finds an object in its grid point can pick up the object and transport it. Eventually it may drop the object at some grid point that is not occupied by another object. Usually the probability to pick up or drag an object depends on the similarity of the objects on neighboring grid points. The more similar these objects are the smaller is the probability that an ant that does not carry an item picks up an item it finds on the grid point and the larger is the probability that an ant that carries an object drags it when the grid point is not occupied. Thus, the standard ant-based clustering algorithms cluster the objects (implicitly) by their relative positions on the grid. In order to obtain an explicit clustering it is necessary to apply a postprocessing step where the objects are assigned explicitly to clusters. Ant based-clustering has been used for example for graph partitioning [10] and text-mining [6, 17]. In order to speed up ant-based clustering it was proposed to move the ants directly to a grid point where it can pick up or drop an object [15]. Another approach is to use spatial transition probabilities for the ants that depend on pheromone values and guide the ants to interesting regions instead of moving the ants just randomly [1]. A variation of ant-based clustering where the ants can place several objects onto a grid point so that these objects form a heap was proposed in [9]. In this algorithm when an ant wants to pick up an object from a heap containing several objects it chooses the

most dissimilar object in the heap.

In [7, 8] ant-based clustering was compared with $k$-means clustering, a hierarchical agglomeration clustering method that is based on the average link metric, and one-dimensional self-organizing map clustering (SOM). The ant-based clustering algorithm studied in [7, 8] has been improved with respect to the standard algorithm by several features, e.g., the ants are equipped with memory, the radius of perception of the ants is increased during the run, and the ants posses a variable stepsize.

A different approach for ant clustering that is based on the chemical recognition system of ants is proposed in [11, 13]. The idea is to use an artificial odor that is learned by the ants and represents the odor of their nest. In the algorithm called ANTCLUST each object (i.e. an $n$-dimensional data vector) that is to be clustered is represented by an ant. The idea is to assign groups of similar ants to a nest that represents a cluster. Initially all ants are not member of any nest. Then, random meeting between pairs of ants take place. During these meetings the ants collect information that determines their behavior as described in the following. Each ant learns an acceptance threshold for the dissimilarity between its own data vector and the data vector of the ant that it has met in order to decide whether the other ant would be accepted as a nestmate. Moreover, each ant measures i) how well it is integrated in its nest by counting for all meetings with nestmates the difference of the number of acceptances and non acceptances, ii) the size of its nest by counting the relative number of meetings with nestmates, iii) the mean and the maximal similarity that it observed during its meetings with other ants. These latter values are used to adapt the acceptance threshold value.

The assignment of an ant to a nest can change during a meeting as described in the following. When two ants meet that are not member of a nest and they accept each other (with respect to their acceptance thresholds) then they found a new nest. If the ants accept each other but one ant is already member of a nest then the other ant joins this nest. If two ants meet that are members of different nests and do not accept each other then the ant which posses the worst integration within its nest is removed from its nest and thus is not member of a nest anymore. If two ants meet that are members of different nests but accept each other then the ant from the smaller nest (with respect to estimated values) leaves its nest and changes to the other ants nest.

An extension of ANTCLUST called Visual ANTCLUST uses two dimensional vectors as labels for a nest [13]. The values of these labels are chosen so that nests with similar ants are placed nearby within the two dimensional space. The labels of the nests are changed dynamically in the algorithm during meetings of the ants. It was shown in [7, 8] that ANTCLUST and Visual ANTCLUST give good results when compared to $k$-means clustering for a fixed $k$.

## 3. Decentralized Packet Clustering

In this section we describe the decentralized packet clustering problem and propose an algorithm for solving this problem by employing the routers. We concen-

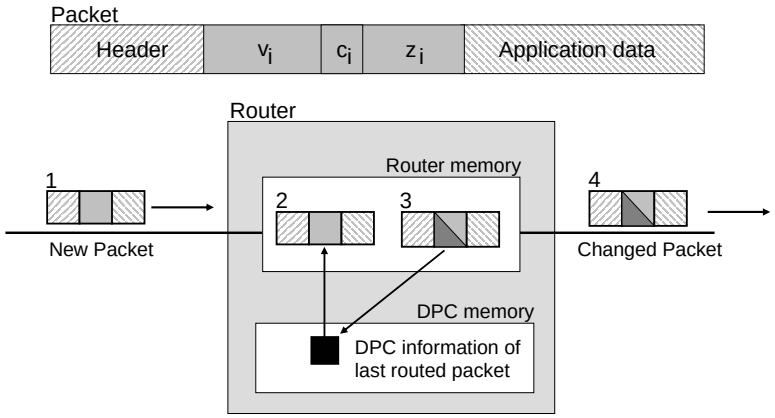


Figure 1: Scheme of DPClust in a router; structure of a packet (above); router with packet at different stages below: 1) incoming new packet, 2) DPC information of the new packet is compared with the copy of DPC information from the stored packet, 3) DPC information of the packet is possibly changed and then the DPC information is copied into the DPC memory, 4) packet with possibly changed DPC information leaves the router

trate on the clustering problem but do not discuss the connected problem of how the clustering information in the packets is used by the application processes that run on the servers (cmp. Section 1). The latter problem depends on the specific application task. Hence, we do not model the servers in the problem formulation.

Consider a network where the nodes are routers. The Decentralized Packet Clustering (DPC) problem is to cluster a set of packets $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ that are send around in the network. Each packet $P_i \in \mathcal{P}$ contains a data vector $v_i$ that is used for clustering. Our aim is to design an algorithm for DCP problem that runs in the routers of the network and satisfies the following requirements:

1. The computational effort of the routers to do the necessary operations for clustering is small.

2. The memory requirements of the algorithm in the routers should be small, so that each router can not store more than a constant number of packets for doing the clustering.

3. Each packet should not store much additional information.

4. The algorithm should not establish a control protocol that requires communication between the routers.

In the following we describe our algorithm DPClust for solving the DPC problem. We assume that each packet $P_i$ has in addition to its data vector, a cluster number $c_i$ and a vector $z_i$ that is an estimation of the centroid of its cluster (altogether this is called the DPC information of the packet). A packet might contain additional information, e.g., header information and application data but this is not relevant

for the clustering algorithm. Thus, a packet $P_i$ can be characterized by its DPC information, i.e., $P_i = (v_i, c_i, z_i)$ for $i \in [1 : n]$. For the clustering algorithm the only information that a router stores is a copy of DPC information of the last packet that is has seen. The main idea of DPClust is that a packet $P_i$ that arrives in a router is compared with respect to the DCP information with the corresponding information $P = (v, c, z)$ that was copied from the predecessor packet. If both packets are in the same cluster (i.e. they have the same cluster number $c_i = c$) the new packet updates the estimation of the centroid of its cluster. If both packet numbers are different then the new packet decides whether it should change to the cluster of the other packet. This is the case when the distance between the data vector $v_i$ of the packet and the estimation $z_i$ of the centroid of its cluster is larger than the distance to the estimation $z$ of the centroid of packet $P$. Formally and in more detail (see also Figure 1) DPClust is described in Algorithm 1.

**Algorithm 1** DPCLust

Let $v, c, z$ be copies of the data vector, the cluster number, and the estimate of the centroid of the last packet that was processed in router $R_j$ respectively. Let $P_i$ be a newly arriving packet.

**if** $P_i$ is in the same cluster as the last packet that was processed in the router, i.e. $c_i = c$,

**then** update the estimate of the centroid of $P_i$ by

$$z_i := (1 - \beta) \cdot z_i + \beta \cdot v$$

where $0 < \beta \leq 1$ is a parameter that determines the relative influence of the other packets data vector and the old estimate $c_i$ of packet $P_i$

**else if** the distance of $v_i$ to the centroid $z$ is smaller than to its own centroid $z_i$, i.e. $v_i - z_i > v_i - z$ (we use Euclidean distance in this paper)

**then** $P_i$ changes its cluster, i.e. $c_i := c$, $z_i := z$

Note that a large value of parameter $\beta$ has the effect that the estimation of the centroid of a packet depends mainly on the last few packets from the same cluster that have been "met" in the routers. For our experiments we assume that all packets to be clustered already exist in the network from the start and that all packets have an unlimited life time. But our methods will in general work also when new packets arrive or old packets are removed from the network (a similar situation is modelled in this paper by connecting subnetworks that were not connected before). Especially for strongly dynamic such situations it is important that $\beta$ is not too small so that old information does not influence the centroid estimation for too long. On the other hand for more static situations $\beta$ should not be too large in order to increase the accuracy of the centroid estimation.

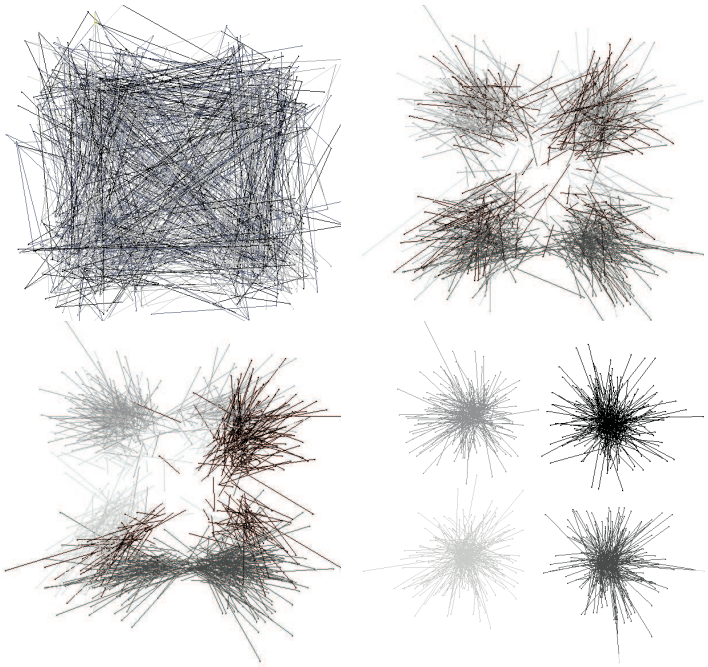

Figure 2: Behavior of DPClust on *Square1* for a network (with one subnetwork) at steps 0 (upper left), 40 (upper right), 80 (lower left), and 120 (lower right); for each packet $P_i$ an arrow connects the data vector $v_i$ with the estimation $z_i$ of the centroid of the cluster; the grey value of an arrow indicates the cluster number

## 4. Experiments

Since this is a first study on the DPC problem we consider for the experiments simple types of networks. Each network $\mathcal{N}$ consists of a set of $r \geq 1$ subnetworks $N_1, N_2, \ldots, N_r$. Each subnetwork contains $k \geq 1$ routers and each router is assigned to a subnetwork. Thus when $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ is the set of routers and $\mathcal{R}_i$ is the set of routers assigned to subnetwork $N_i$ then $(\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_r)$ is a partition of $\mathcal{R}$. We assume that the routers within a subnetwork are fully connected. For the connection of the subnetworks we study three types of topologies namely ring networks, fully connected networks, and star networks. A ring network $N$ consists of a directed ring of subnetworks $N_1, N_2, \ldots, N_r$ so that $N_{i+1 \bmod r}$ is the successor of $N_i$. In the fully connected network each subnetwork is directly connected to every other subnetwork. In the star network the subnetwork $N_1$ is connected to every other subnetwork and vice versa. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be the set of packets in the network. Each packet is assigned to a subnetwork. Let $f(i)$ be the index of the subnetwork packet $P_i$ is assigned to.

**Algorithm 2** Test Scenario for Ring Networks

Initialization (see details below)

**repeat**

i) Randomly choose a packet $P_i \in \mathcal{P}$ with uniform probability.

ii) With probability $\alpha > 0$ set $f(i) := f(i) + 1 \bmod r$, i.e., assign $P_i$ to the successor subnetwork of its actual subnetwork.

iii) Randomly choose a router $R_j \in N_{f(i)}$ with uniform probability.

iv) Apply DPClust in router $R_j$ to packet $P_i$.

**until** stopping criterion is met

To investigate the behavior of DPClust we describe the test scenario for the ring networks using Algorithm 2. Basically, the algorithm describes how packets move in the network. Parameter $0 \leq \alpha \leq 1$ in the algorithm is called the exchange parameter since it determines the probability of packet exchanges between a subnetwork and its successor subnetwork in the ring. Note, that the description of the algorithm can easily be modified for other behaviors of the packets and other network topologies. E.g., for fully connected networks Step ii) is: With probability $\alpha$ set $f(i) := f(j)$ with $j \neq i$ uniformly chosen from $1, 2, \ldots, i-1, i+1, \ldots r$, i.e., assign $P_i$ to one random neighbored subnetwork. An similar definition of Step ii) has been used for star networks so that for the same value $\alpha$ and same total number of packets the expected number of packets that change their subnetworks is the same as for the ring network. Thus, in a star network with $r$ subnetworks the exchange rate for the inner subnetwork is $(\alpha \cdot r)/2$ and for each outer subnetwork the exchange rate is $(\alpha \cdot r)/(2r-2)$.

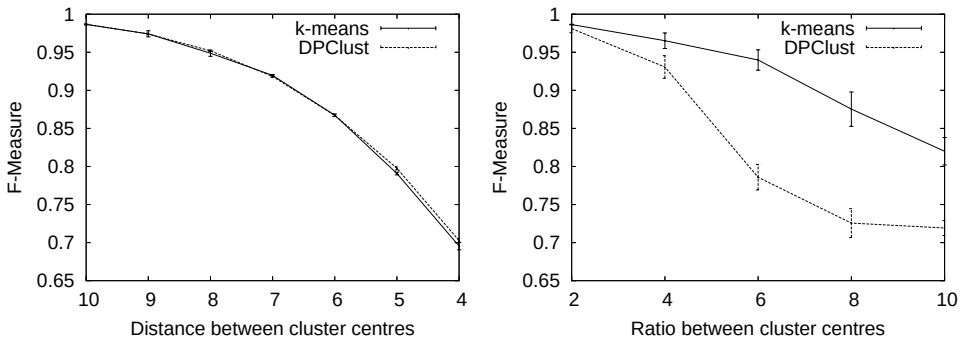


Figure 3: Performance of DPClust and $k$-means on *Square1* to *Square7* with different distances between cluster centers (left) and on *Sizes1* to *Sizes5* with different ratios between cluster sizes (right); $\beta = 0.05$, 2000 steps; vertical bars show the standard error

The initialization is done so that, i.) each packet $P_i \in \mathcal{P}$ is assigned randomly to a subnetwork, i.e., $f(i)$ is chosen uniformly in $[1 : r]$, ii.) each packet $P_i \in \mathcal{P}$ is assigned randomly to a cluster, i.e., $c_i$ is chosen uniformly from $[1 : k]$, and iii.) for each packet $P_i \in \mathcal{P}$ the estimate of the centroid is set to the value of a data vector of a random packet, i.e., $z_i := v_h$ where $h$ is chosen uniformly from $[1 : n]$.

To test DPClust we used the same type of clustering instances for the data vectors as have been used also in other papers on ant-based clustering (e.g. [7]). There are two types of instances both of which consist of two-dimensional data vectors from four classes (clusters). One data set is defined for investigating the influence of class (cluster) overlaps and the other data set for investigating the influence of different (class) cluster sizes.

For the first type of instances called *Square* each of the four data classes (clusters) contains 250 data vectors. The data vectors are generated by a two-dimensional normal distribution with standard deviation 2 in both dimensions. The centers of the normal distributions of the four clusters are arranged in a square. The test data sets *Square1* to *Square7* differ by the distance between the class (cluster) centers which is $10, 9, \ldots, 4$ respectively (an example of a test instance of type Square1 is shown in Figure 2). The second type of instances called *Sizes* is similar to the *Square1* data but the size of the classes (clusters) is different. For *Sizes1* to *Sizes5* the ratio between the size of the three small classes (cluster) (which are of equal size) and the size of the large class (cluster) is $2, 4, \ldots, 10$ respectively.

For the evaluation of our method we use the F-Measure [20]. This measure combines measures for the purity and the completeness of the clusters. The F-measure assumes that the real partition of packets into different classes is known. Thus data vectors of the packets are assumed to be partitioned into $k$ classes. Let $s_i$ be the number of members in class $i$. The clustering algorithm generates a partition

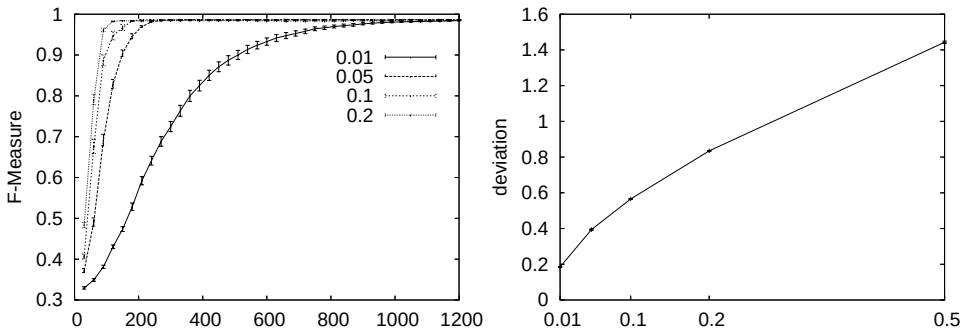


Figure 4: Convergence behavior of DPClust for $\beta \in \{0.01, 0.05, 0.1, 0.2\}$ (left); mean deviation between estimated centroids and true centroids for $\beta \in \{0.01, 0.05, 0.1, 0.2, 0.5\}$ (right); 1 router; vertical bars show the standard error

of the packets into $k$ clusters. Let $n_j$ be the number of members in cluster $j$. Let $n_{ij}$ be the number of members of class $i$ that are also member of cluster $j$. For each class $i$ and cluster $j$ the precision and recall are defined as $p(i,j) = n_{ij}/n_j$ and $r_{ij} = n_{ij}/s_i$, respectively. Let $F(i,j) = ((b^2+1)\cdot p(i,j)\cdot r(i,j))/(b^2\cdot p(i,j)+r(i,j))$. Here we chose $b = 1$ which gives $p(i,j)$ and $r(i,j)$ an equal influence in order to compare us with the results of other works. The F-Measure for the clustering with respect to the given partition into classes is defined as $F = \sum_i (s_i/n) \max_j \{F(i,j)\}$.

If not stated otherwise, for parameter $\beta$ the value 0.1 was used and the test instance was Square1. All the results are averaged over 50 test runs. In the following section a step of an algorithm means that number of packets many iterations were done (for most experiments 1000 packets were used).

## 5. Results

To illustrate the behavior of DPClust we show four snap-shots from a run of DPClust on an instance of *Square1* (see Figure 2). In the figure each packet $P_i$ is depicted by an arrow that connects its data vector $v_i$ with the actual estimation $z_i$ of the centroid of its cluster. The cluster number $c_i$ is indicated by the grey value of the arrow. The upper left part of the figure shows the random situation at the start of the test run. This random situation is followed by a situation where the centroids estimations become smaller but reasonably well formed clusters have not been found (upper right part of the figure). It can be seen that estimates of the centroids do not approximate the real centroids well (very different centroids with the same grey value occur). This is not surprising because at this stage of a run the packets are often changing their clusters. In later stages the quality of the clusters increases and the clusters have most of their packets from only one or two classes (lower left part of the figure). Finally well formed cluster are found which correspond to the classes (lower right part of the figure).

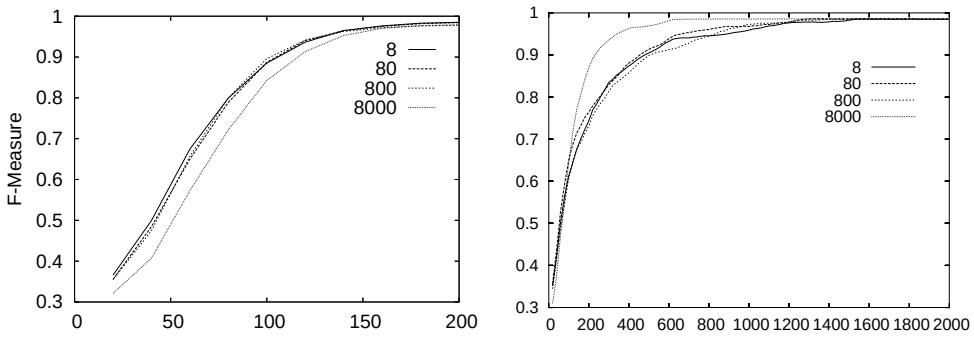


Figure 5: Influence of the number of routers (8, 80, 800 or 8000 routers) on convergence speed of DPClust in a ring network with subnetworks; 1 subnetwork (left); 8 subnetworks with $\alpha = 0.032$ (right)

Before we study different aspects of the clustering behavior of the decentralized DPClust algorithm we compare it with the $k$-means algorithm (see Figure 3). For DPClust we used one router (it is shown later that the results of DPClust are very stable with respect to a changing number of routers). The results show that both algorithms perform equally well on the *Square* instances (see left part of Figure 3). Clearly, an increasing overlap between the clusters leads to decreasing F-Measure values for the clusterings of both algorithms. On the *Size* instances both algorithms have difficulties for large size differences between the clusters (see right part of Figure 3). But $k$-means performs better than DPClust for size differences that are larger than 2. It has to be mentioned here that the initialization of the clusters is an important factor for the performance of both algorithms. Since we used the data vectors of random packets for the initial centroids most of these centroids will be from the larger cluster which is a difficult situation for the algorithms. It is to be expected that a different initialization method where the initial centroids are more regularly distributed in the two-dimensional plane can improve the results for the *Size* type instances.

The influence of parameter $\beta$ is shown in the left part of Figure 4. Recall, that $\beta$ determines the relative influence of a packets own centroid estimation to the copy of the data vector of the packet that is stored in the router when a new centroid estimate is computed. It can be seen in the figure that for all tested values of $\beta$ ($\beta = 0.01$, ..., $\beta = 0.2$) the clusterings that are obtained at the end of the runs are of the same quality with respect to the F-Measure. But $\beta$ has an influence on the speed of improvement of the clustering when starting with the initial random clustering. The right part of Figure 4 shows the average deviation of the estimated centroids of the packets from the true centroids of the clusters. Clearly, the larger the value of $\beta$ is the larger is the deviation of the estimated centroid from the true centroid. For all other tests we use $\beta = 0.1$ so that DPClust finds a good cluster

fast and the difference between the estimated and the real centroids are not too large.

Figure 5 shows the influence of the number of routers on the clustering behavior of DPClust. It seems surprising that the influence of the number of routers on the quality of clustering is so small for a network which consists of only one subnetwork (see left part of the figure). This indicates that DPClust will work successfully in large networks with many routers working in parallel. In other words, DPClust obtains very good speedup values even for a large number of routers. Note, that it takes some time until all of 8000 routers have received at least one packet. This leads to the smaller value of the F-Measure after only 20 steps where the number of meetings between packets in the 8000 routers is effectively nearly the same as after only 12 steps for 8 routers. Since for large networks the situation that all routers are connected directly is not realistic we have also studied the influence of the number of routers for a ring network with 8 subnetworks (see right part of Figure 5). Here a large number of routers is even an advantage leading to a superlinear speedup when the aim is to obtain a clustering with a certain good F-Measure value.

For packet clustering in networks it is interesting to investigate scenarios where the network consists of subnetworks that are only loosely connected. Figure 6 shows the results for DPClust for ring networks that are divided into different numbers subnetworks. For each number of subnetworks various values of the exchange parameter $\alpha$ were tested. The results show that the algorithm has difficulties to find a good (global) clustering when the packet exchange rate between the subnetworks is very small (e.g., $\alpha \leq 0.002$ for 4 subnetworks). Moreover, it seems that for these small exchange probabilities the values of the F-Measure converge to a small value. This value becomes smaller when the ring network has more subnetworks. The F-Measure converges to approximately 0.65 (0.55, 0.43) for 2 (4, 16, respectively) subnetworks with $\alpha = 0.0005$). For 32 subnetworks the F-Measure was still improving after 10000 evaluations. The reason for this behavior is that the clusterings which are established in the subnetworks during the first iterations might be similar but the clusters are numbered differently. Hence, when a packet changes its subnetwork it might be that the corresponding cluster in the new subnetwork has a different number. Thus, to cope with such a situation it might be advantageous to enforce some explicit information exchange between the subnetworks about the numbering of clusters. The good message from the results is that DPClust finds a consistent numbering of the clusters implicitly when the packet exchange rate between the subnetworks is not too small (for $\alpha = 0.016$ the final large value of the F-Measure is reached after about 400 steps for up to 8 subnetworks). Not surprisingly, the larger the amount of packet exchange is the faster is the increase of the F-Measure values. In general, it can be seen that the larger the number of subnetworks is the slower is the increase of the F-Measure.

Since in the experiment corresponding to Figure 6 the total number of packets was the same for all tests it follows that the networks with a large number of subnetworks have less packets in each subnetwork. In order to study the influence of the number of packets we also compared DPClust for ring networks with total

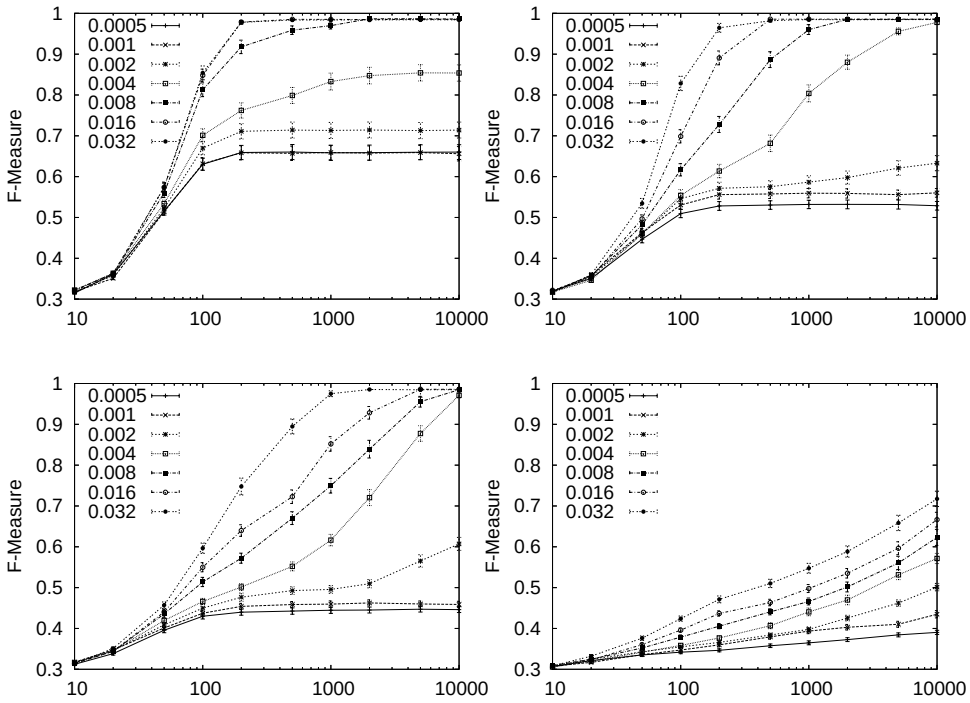


Figure 6: Performance of DPClust on the ring network with 2 (upper left), 4 (upper right), 16 (lower left), and 32 (lower right) subnetworks and 1000 packets as a function of the number of packet evaluations; packet exchange parameter $\alpha \in \{0.0005, \ldots, 0.032\}$, 1 router in each subnetwork; vertical bars show the standard error

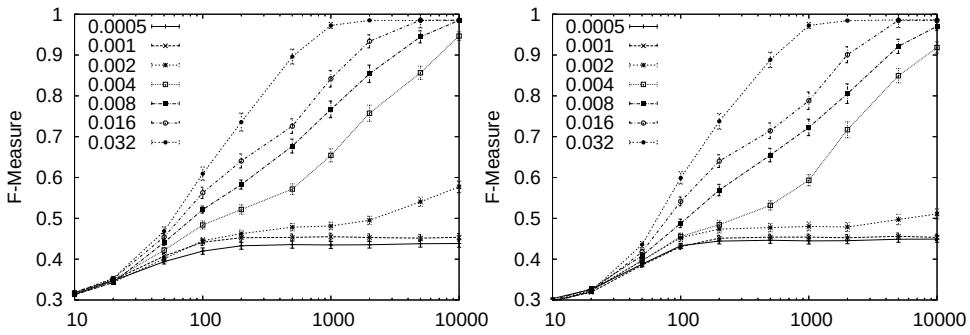


Figure 7: Performance of DPClust on ring network with 8 subnetworks and different number of packets per subnetwork as a function of number of packet evaluations (125 packets per subnetwork (left), 250 packets per subnetwork (right)); packet exchange parameter $\alpha \in \{0.0005, \ldots, 0.032\}$, 1 router in each subnetwork; vertical bars show the standard error

numbers of 1000 and 2000 packets. The results are shown in Figure 7. While the general appearance of the curves is very similar it can be seen that it takes slightly longer for the larger number of packets and smaller exchange rates to reach the same F-Measure values as with half as much packets.

Results on the influence of the connection topology between the subnetworks is shown in Figure 8. The figure shows the performance of DPClust for a fully connected and a star network, both having 8 subnetworks. Compared to the ring network with 8 subnetworks (see left part of Figure 7) the F-Measure in the fully connected network increases faster. This is different for the star network where the F-Measure values increase slower than for the ring network. A possible reason is that it is more difficult for a subnetwork in the fully connected network to establish a clustering with a different numbering than in the other subnetworks. In the star network it seems to be difficult to establish the same numbering in all subnetworks because the outer subnetworks are only connected to the center subnetwork and in the center it is difficult to establish a good clustering because it has much packet exchange with all other networks.

In order to study the behavior of DPClust in a dynamic situation where it may happen that new packets arrive from subnetworks that were disconnected before we studied the following scenario. For a network with 4 subnetworks DPClust was run for 400 steps in each subnetwork separately ($\alpha$ was set to zero). Then the subnetworks were connected by setting $\alpha = 0.016$ or $\alpha = 0.032$ respectively. Since it is not relevant for the application processes in a subnetwork how the clustering is done in a different subnetwork that is disconnected from it we do not only measure the global F-Measure (over all 1000 packets) but also the average local F-Measure. The local F-Measure is measured with respect to the packets that are actually contained in the considered subnetwork. The results are shown in Figure 9 for

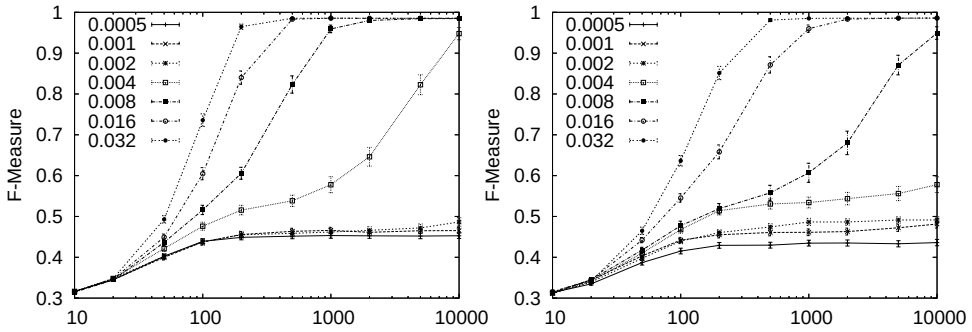


Figure 8: Performance of DPClust in a fully connected network (left) and a star network with 8 subnetworks as a function of number of packet evaluations; packet exchange parameter $\alpha \in \{0.0005, \ldots, 0.032\}$, 1 router and 1000 packets in each subnetwork; vertical bars show the standard error

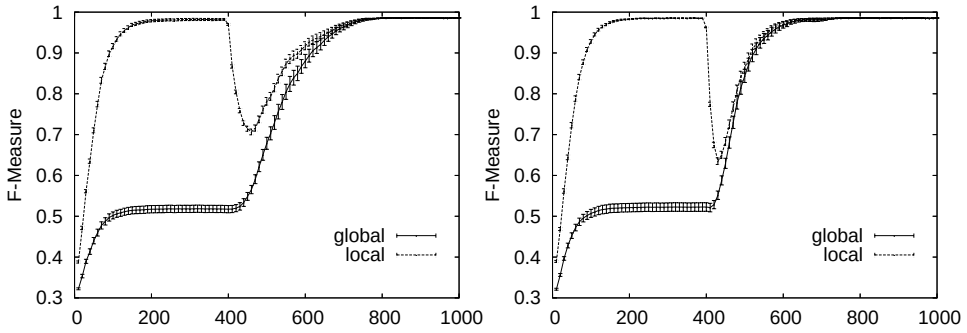


Figure 9: Average local and global F-Measure for DPClust in a ring network with 4 subnetworks: no packet exchange ($\alpha = 0.0$) was done for the first 400 steps, then $\alpha$ was set to 0.016 (left) 0.032 (right)

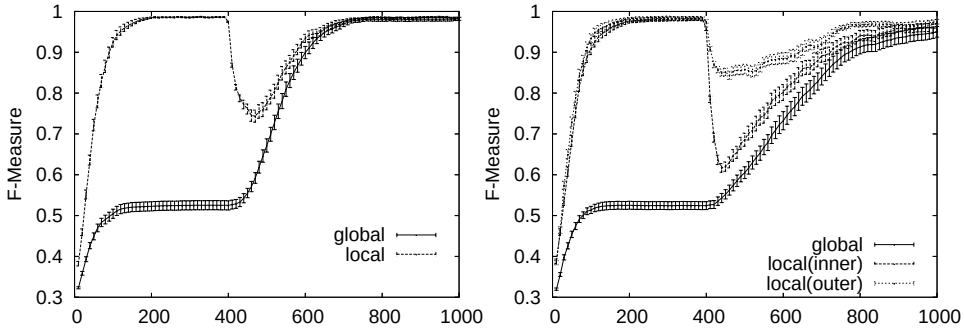


Figure 10: Average local F-Measure for DPClust in a fully connected network (left) and a star network (right) each with 4 subnetworks: no packet exchange ($\alpha = 0.0$) was done for the first 400 steps, then $\alpha$ was set to 0.016; for the star network the local F-measure is shown separately for the inner subnetwork and an outer subnetwork

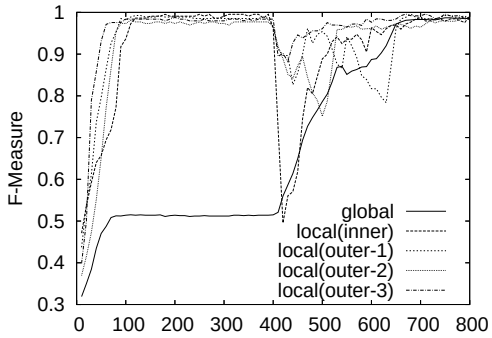


Figure 11: Single run of DPClust in a star network with 4 subnetworks: shown are the local F-Measures for the inner subnetwork and the three outer subnetworks: no packet exchange ($\alpha = 0.0$) was done for the first 400 steps, then $\alpha$ was set to 0.016

a ring network with 4 subnetworks. The figure shows that the average local F-Measure has already high values before the subnetworks are connected. This is in contrast to the global F-Measure which has only small values. This is because even when the clustering is essentially the same in all subnetworks the clusters will usually be numbered differently in the subnetworks. After the connection of the subnetworks the average local F-Measure decreases because packets from other subnetworks enter. It is encouraging how fast the local F-Measure increases again to the old values. This means that a kind of renumbering of the clusters has been done successfully by the algorithm without any central control.

Figure 10 shows the average local F-Measure and the global F-Measure for the same scenario as above for the star network and the fully connected network (subnetworks were connected with $\alpha = 0.016$). The curves for the fully connected network are quite similar to the corresponding curves for the ring network (see left part of Figure 9). However for the star network the global F-Measure increases slower after opening the connection. This is in accordance with the observations that have been made for the static scenarios. It is interesting that there is a clear difference between the inner subnetwork and the outer subnetworks of the star network. For the inner subnetwork the local F-Measure decreases stronger than for the local F-Measures of the ring and fully connected network. Whereas the decrease is much less for the outer subnetworks. This is because the inner network receives more packets from other subnetworks than the outer networks (Recall that the exchange rate is $(\alpha \cdot r)/2 = 0.032$ for the inner and $(\alpha \cdot r)/(2r-2) = 0.012$ for each outer subnetwork).

Figure 11 shows the F-Measures for a single typical run of DPClust in a star network with 4 subnetworks. It can be seen that the local F-Measure of the inner subnetwork decreases much stronger after opening the connection than for the outer subnetworks. This can be explained because the inner subnetwork receives three times as much packets from outside than each outer subnetwork. Moreover it can be seen that for one of the outer subnetworks the local F-measure remains significantly higher than for the other two outer subnetworks. The reason is that for this run all three subnetworks used different cluster numberings for the packets, i.e., packets from clusters that correspond to the same class have different numbers. The subnetwork with the heigh local F-measure was in this run the "winner" with respect to the renumbering of the local clusterings, i.e., its numbering was finally adopted by the other subnetworks. Clearly it can also happen that two or three of the subnetworks use the same numbering (or partially equal numberings). In this case it can typically be observed that for only one or none of the outer subnetworks the local F-Measure decreases strongly after the connection.

## 6. Conclusion

We have proposed a new type of a decentralized clustering problem for networks — called the Decentralized Packet Clustering (DPC) problem. This problem is to find for a set of packets that are send around in a network of routers a good clustering so that no central process is involved. The clustering has to be done by the

routers without direct information transfer and with only minimal computational and routing resources. We have proposed a heuristic algorithm for the DPC problem which is called DPClust. It was shown that the decentralized DPClust algorithm has similar performance as $k$-means on some standard benchmark problems while it is worse on others. However, our main focus was to investigate whether DPClust is robust and successful for networks of different topologies. Moreover we have studied whether DPClust can handle difficult situations which can occur in large networks, namely that parts of the network are more or less disconnected from each other or that the quality of connections between subnetworks changes. The results are promising and show that different clusterings that might have been established in the subnetworks are combined nicely by DPClust in the sense that the resulting clustering has a good global F-Measure after the packet exchange between the subnetworks has been increased.

Our future work is to study changing network topologies and more complicated dynamic situations in networks (e.g. the birth and dead of packets). In addition we plan to use control packets that transfer control information and which use behavioral mechanisms of ant-based algorithms for networks. It is also interesting to include other information in the information transfer of the meetings in the routers in order to solve other type of clustering problems (e.g., where clusters have different shapes).

## References


1. A. Abraham, V. Ramos: Web Usage Mining Using Artificial Ant Colony Clustering and Genetic Programming. Proc. of the Congress on Evolutionary Computation (CEC03), IEEE Press (2003).
2. M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane: Clustering and server selection using passive monitoring. Proc. IEEE INFOCOM 2002, IEEE Press (2002).
3. J.L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, L. Chretien: The dynamics of collective sorting: robot-like ants and ant-like robots. In: J.-A. Meyer et al. (Eds.) Proc. Simulation of Adaptive Behavior: From Animal to Animats, 356–365 (1991).
4. I. S. Dhillon, D. S. Modha: A Data-clustering Algorithm on Distributed Memory Multiprocessors. Proc. Large-Scale Par. Data Mining, LNAI 1759, 245–260 (2000).
5. M. Eisenhardt, A. Henrich: Classifying Documents by Distributed P2P Clustering. In: Dittrich et al. (Eds), Proceedings Informatik 2003, GI Lecture Notes in Informatics, Frankfurt, (2003).
6. J. Handl, B. Meyer: Improved Ant-based Clustering and Sorting in a Document Retrieval Interface. Proceedings of the Seventh international Conference on Parallel Problem Solving from Nature (PPSN VII). LNCS 2439, 913–923 (2002).
7. J. Handl, J. Knowles, and M. Dorigo: Strategies for the increased robustness of ant-based clustering. Postproceedings of the First International Workshop on Engineering Self-Organising Applications (ESOA 2003), LNCS 2977, 90–104 (2003).
8. J. Handl, J. Knowles, and M. Dorigo: On the performance of ant-based clustering. In: Proc. 3rd Int. Conf. on Hybrid Intell. Systems (HIS 2003), IOS Press, (2003).
9. P. M. Kanade, L. O. Hall: Fuzzy Ants as a Clustering Concept. Proceedings 22nd

International Conference of the North American Fuzzy Information Processing Society NAFIPS, 227–232 (2003).

10. P. Kuntz, D. Snyers: Emergent colonization and graph partitioning. In: D. Cliff et al. (Eds.), Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats, MIT Press, 494–500 (1994).
11. N. Labroche, N. Monmarch, G. Venturini . A new clustering algorithm based on the chemical recognition system of ants. Proc. European Conf. on AI, IOS Press, 345–349 (2002).
12. N. Labroche, N. Monmarch, G. Venturini: AntClust: Ant Clustering and Web Usage Mining. Proc. of GECCO-2003, Springer, LNCS 2723, 25–36 (2003).
13. N. Labroche, N. Monmarch, G. Venturini. Visual clustering with artificial ants colonies. Proc. 7th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES 2003), Springer, LNCS 2773, 332–338 (2003).
14. E. D. Lumer, B. Faieta: Diversity and Adaptation in Populations of Clustering Ants. Proc. 3rd Conf. on Simulation of Adaptive Behavior: From Animal to Animats (SAB94), MIT Press (1994).
15. N. Monmarche, M. Slimane, and G. Venturini: AntClass: discovery of clusters in numeric data by an hybridization of an ant colony with the kmeans algorithm. Technical Report 213, Laboratoire d'Informatique de l'Université de Tours (1999).
16. E. Ogston, B. Overeinder, M. van Steen, and B. Brazier: A Method for Decentralized Clustering in Large Multi-Agent Systems. In: Proceedings of the Second International Joint Conference on Autonomous Agent and Multi Agent Systems (AAMAS03), ACM Press, 798–796 (2003).
17. V. Ramos, J. J. Merelo: Self-Organized Stigmergic Document Maps: Environment as a Mechanism for Context Learning. Proc. 1st Spanish Conf. on Evolutionary and Bio-Inspired Algorithms AEB2002, 284–293 (2002).
18. O.B.V.Ramanaiah, H. Mohanty: Adapting a Distributed Data Clustering Algorithm for Mobile Environment. Proc. 4th Int. Conference on Information Technology, Gopalpur-on-Sea, India, (2001).
19. L. Ramaswamy, B. Gedik, and L. Liu: Connectivity Based Node Clustering in Decentralized Peer-to-Peer Networks. Proc. of the 3rd International Conference on Peer-to-Peer Computing (P2P 2003), 66–73 (2003).
20. C. J. van Rijsbergen: Information retrieval. 2nd edition, Butterworths, London, UK, (1979).