A Method for Decentralized Clustering in Large Multi-Agent Systems

Elth Ogston , Benno Overeinder, Maarten van Steen, and Frances Brazier
Department of Computer Science, Vrije Universiteit Amsterdam
{elth,bjo,steen,frances}@cs.vu.nl

ABSTRACT

This paper examines a method of clustering within a fully decentralized multi-agent system. Our goal is to group agents with similar objectives or data, as is done in traditional clustering. However, we add the additional constraint that agents must remain in place on a network, instead of first being collected into a centralized database. To do this we connect agents in a random network and have them search in a peer-to-peer fashion for other similar agents. We thus aim to tackle the basic clustering problem on an Internet scale and create a method by which agents themselves can be grouped, forming coalitions. In order to investigate the feasibility of a decentralized approach, this paper presents a number of simulation experiments involving agents representing two-dimensional points. A comparison between our method's clustering ability and that of the k-means clustering algorithm is presented. Generated data sets containing 2,500 to 160,000 points (agents) grouped in 25 to 1,600 clusters are examined. Results show that our decentralized agent method produces a better clustering than the centralized k-means algorithm, quickly placing 95% to 99% of points correctly. The the time required to find a clustering depends on the quality of solution required; a fairly good solution is quickly converged on, and then slowly improved. Overall, our experiments indicate that the time to find a particular quality of solution increases less than linearly with the number of agents.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; I.5.3 [**Pattern Recognition**]: Clustering

General Terms

Algorithms, Experimentation, Performance

Keywords

Decentralized Systems

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AAMAS'03, July 14–18, 2003, Melbourne, Australia.
Copyright 2003 ACM 1-58113-683-8/03/0007 ...$5.00.

1. INTRODUCTION

Agents that wish to cooperate within a multi-agent system must have a means of finding each other. The straightforward solution to this problem is to create a central directory server that is able to match requests. However, this centrally directed solution limits the autonomy of agents with respect to their choice of partners, and it limits the scalability of the multi-agent system as a whole. Ideally agents would, on their own, be able to group together to form cliques of like minded agents. As a result they would know their potential partners (members of their social circle) and could directly negotiate new partnerships based on more information than a directory server would contain. Grouping agents in this way, based on similar objectives, can be viewed as a clustering problem. Clustering has been studied in a variety of fields, notably statistics, pattern recognition and data mining. These fields have a wide range of purposes in mind, for instance discovering trends, segmenting images, or grouping documents by subject. However, in all of these disciplines the underlying problem is the same; given a number of data items, create a grouping such that items in the same group are more similar to each other than they are to items in other groups [5]. Most algorithms for clustering focus on how to form these groups given a file or database containing the items. Yet, for Internet applications like finding similar web pages or finding agents with similar interests, items can be widely distributed over many machines and the issue of collecting the items in the first place gains importance. Centralized clustering is problematical if data is widely distributed, data sets are volatile, or data items cannot be compactly represented. Decentralization, on the other hand, is a thorny problem. Even in the centralized case where each data item can be compared to every other data item, perfect clusters can be hard to find. Decentralization creates the additional complication that even if a correct classification can be determined with the incomplete information available, the location of items belonging to a class also needs to be discovered.

This paper considers the case where classification is straightforward and focuses on the question of finding potential cluster members in a decentralized fashion. By studying in depth a simplified example of agent grouping we hope to gain insight into dynamics that can be used to create more complex, self-organizing agent communities. With this purpose in mind, we view clustering as a search problem in a multi-agent system in which individual agents have the goal of finding other "similar" agents. Agents aim to form groups among themselves, and these groups constitute a clustering. In large scale Internet systems potentially millions of agents are spread across possibly as many machines. As a result each agent will always have a view of only a very small fraction of the rest of the system. Our research is concerned with the minimal abilities and resources required by such agents. We create an ab-

stract model of simplified agents which have a very small range of actions and act using straightforward decision functions. Furthermore, these agents can generally only communicate in a peer-to-peer manner with a limited amount of additional coordination among small groups.

We study the behavior of this abstract system through simulation experiments. In these experiments agents are each given a two-dimensional point and seek to group themselves based on the Euclidean distance between their points. Initially agents are randomly assigned a small number (5) of neighbor agents. These neighbors are an agent's only view of the system as a whole. Based on these local views, agents form clusters with the closest points they come across. Agents within a cluster coordinate, combining their local views to allow each member to search a broader range of neighbors for better matches. Clusters are limited in size by a user-defined parameter. Once clusters have grown to this size they spilt when better matches are found by their members, allowing stronger new clusters to form. We find that using this method, given an appropriate maximum cluster size, agents are able to quickly form an approximation of an underlying clustering in the data points. For small systems with 2,500 to 40,000 agents in 25 to 400 clusters the system quickly converges to a configuration in which 99% of points are placed in the correct clusters. For a larger system with 160,000 agents and 1,600 clusters the same rapid convergence to a solution with 95% of points placed correctly is proceeded by a tail behavior where the solution slowly improves. Further experimental results indicate that finding a particular quality of solution costs less than linear time as the number of agents and clusters increases.

The remainder of this paper first discusses the application of the clustering problem to multi-agent systems and surveys previous work. Section 3 sketches the model we study followed by a precise description of the simulated procedure. Section 4 presents our methodology and experimental results. A discussion of future directions in which these experiments can be expanded concludes the paper.

2. BACKGROUND AND RELATED WORK

Middle agents or directory services are commonly used in multi-agent systems to enable the location of agents with particular capabilities [1]. Such services, however, add an essentially centralized component to an ideally decentralized agent world. The formation of groups of agents based on like interests provides a potential alternative for very large decentralized systems where maintaining a directory becomes too costly. Such groups place potential partners for collaboration in an agent's immediate local environment [3] [12]. When agents' interests include jointly working on common tasks this process evolves into coalition formation; the negotiation of agreements between agents with complementary skills for the distribution of work and rewards [9] [14] [15]. Clustering, as studied in this paper, is a more basic problem, yet an essential component of the process of forming coalitions. A multi-agent system made up of heterogenous agents that cannot somehow group similar agents, is unable to introduce potential coalition members to each other. Clustering, on the other hand, is usually studied as a centralized problem. This paper surveys previous work on clustering and explore how a decentralized approach can be designed for a multi-agent system. The resulting procedure could be applied as a directory service for multi-agent systems and can enhance our understanding of coalition formation in general.

There are a large number of centrally controlled algorithms for discovering natural clusters, if they exist, in a data set (see [6] for a general review of the literature). The majority of these algorithms focus on finding clusters given various properties of the data set; clusters of widely differing sizes, odd cluster shapes, little separation between clusters, noise, outliers, high-dimensional data and complex data types for which a similarity function is difficult to define. In general, clustering algorithms focus on creating good compact representations of clusters and appropriate distance functions between data points. To this purpose they generally need a user to provide one or two parameters that indicate the types of clusters expected. Most commonly, algorithms are given the number of clusters into which the data set is to be split or a density value that defines that the expected distance between points within clusters. Since a central representation is available, where each point can be compared to each other point or cluster representation, points are never placed in a cluster with hugely differing members. Mistakes made by these algorithms instead take the form of incorrectly splitting a real data cluster in half or incorrectly combining two neighboring data clusters into a single cluster. The definition of clustering as a whole is imprecise; the creation of clusters in which points have more in common with other cluster members than with members of other clusters. Given the complexities listed above it is usually not entirely clear what the "correct" clustering of a data set is. There is generally no one best algorithm for obtaining good clusters [5]. The appropriate algorithm depends on the peculiarities of the data set considered.

This paper focuses on yet another complexity that must be faced in multi-agent systems: the distribution of the data over many machines. To allow us to clearly separate the issue of decentralization from that of hard to distinguish clusters, basic data sets in which clusters are clearly separable, two-dimensional, of equal size and circular in shape are considered. As these data sets do not contain any of the difficulties addressed by more complex algorithms, the clusters found by our multi-agent system are compared to those found by Forgy k-means clustering, as described in [5]. This is the simplest of the centralized clustering techniques. Nonetheless, it works well on the elementary data sets examined and illustrates the basic abilities and common mistakes of centralized clustering.

An orthogonal line of research to the quality of clustering is the speed of clustering algorithms. The amount of time it takes to find a clustering is crucial when considering large data sets. It is this line of research that we build upon in this paper. K-means clustering has the lowest time complexity. It chooses a set of k cluster centers, called centriods, and compares each of the n points in the data set to each centroid, placing each point in the cluster to which it is closest. It then recalculates centroids based on the resulting cluster memberships and repeats this process l times. Its time complexity is thus $O(nkl)$, however k and l are usually small compared to n and thus it is usually considered to be $O(n)$. K-means is a partitional algorithm, meaning that it creates a single partition of the data. More complex, and thus more accurate, algorithms are generally hierarchical. They build a table of the distances between every pair of points in the data set and use this table to create a series of partitions, each time splitting or combing clusters based on the next largest or smallest distance between points. However, the construction of the distance table costs $O(n^2)$ time, making hierarchical algorithms unsuitable when large amounts of data are involved.

A number of papers have addressed ways in which to deal with large data sets. One way of reducing the execution time is to reduce the space of solutions that a clustering algorithm has to search. CLARANS [10] is a k-medoid based algorithm that uses randomized search. K-medoid algorithms are similar to k-means algorithms except that they represent clusters by their medoids, the most central member of the cluster, instead of their centroids, the mean value point of the cluster. Thus for a given k it is possible to search

all the possible sets of k medoids. There are, however, n choose k of these. CLARANS reduces this search space by using a randomized hill climbing technique. Starting with a set of medoids it checks at random only some of the sets that differ by one medoid for an improvement in the clustering. This random search, when restarted a few times in different places, is shown to produce good clusters while increasing the efficiency over exhaustive search. An alternative way of reducing the problem space is to consider only a sample set of the data points. CLARA [8] makes a random selection of points before running a k-medoid algorithm. CURE [4] also uses random sampling, combined with other techniques, to extend the range of a hierarchical clustering algorithm. Sampling assumes that a large enough sample contains a sufficient number of points from each cluster, and thus needs to be given a known minimum cluster size. BIRCH [16] also performs hierarchical clustering on a reduced sample space. BIRCH however uses a pre-cluster phase in which it first creates a compact summary of data points, called a CF-tree. Nodes of the CF-tree represent high density groups of points that are defined to be in the same cluster based on some threshold distance. CF-tree nodes are then sparse enough to be clustered using a hierarchical algorithm.

A second method of dealing with large data sets is to partition them into a series of smaller problems. CURE, in addition to sampling data, divides the resulting sample into partitions and separately pre-clusters each one. The resulting sub-clusters are then combined to create a final clustering. P-CLUSTER [7] parallelizes k-means clustering by partitioning the data set over a network of workstations. A server workstation determines and distributes an initial set of centroids and combines the information on the resulting clusters in each partition. It then distributes new centroid information for the following k-means pass. Olson [13] surveys methods of parallelizing hierarchical clustering by distributing the calculation and storage of the distance matrix. Alternatively, density based algorithms create very fine partitions of data by placing two points in the same cluster should they be close enough together and in an area with sufficiently many other points. This requires creating a data structure that can efficiently find the nearest spatial neighbors of a point. DBSCAN [2], for example, defines clusters as areas with a threshold density of points within a given radius and grows clusters out from single points based on this density.

All of the above methods can be used to find clusters of similar agents in multi-agent systems. Methods based on partitioning that limit, or eliminate, the global data that needs to be maintained are, nevertheless, best suited for large scale systems. The method we present in this paper is based on the concept of creating a fine partitioning of points. Points are agents, giving us a high level of local decision making capability. Two agents are defined as belonging to the same cluster if their attributes are sufficiently similar. However, unlike density-based methods, we do not assume that agents know of all of their spatial neighbors, as this would require a global view of the system. Instead, agents choose cluster partners based on the closest agents they encounter and later abandon more distant partners in favor of new found closer ones. The decision of when to drop partners can either be based on a minimum distance as in density-based algorithms, or a fixed maximum cluster size corresponding to the k used in k-means clustering. In this paper we fix the maximum cluster size. Overall, this agent approach allows us to find clusters in a fully decentralized manner. Although all data points (or agents) are considered in our analysis of the clusters, individual agents only see a sample of the data points at any one time. Agents work in parallel. For this reason we do not concentrate on using resources efficiently, instead we rely on fine grain parallelization and partitioning to allow us to handle very large data sets.

3. AGENT PROCEDURE

3.1 General Model

In our model agents are characterized by their attributes, and can thus be defined, for the purpose of clustering, as a set of data items. Each agent has a small number of *links* to other agents. These links represent communication channels and thus define the neighborhood of each agent. The aim of the system is for the agents to rearrange these links and to select some of them to form *connections* or *connected links* between agents, generating a graph of connections corresponding to a clustering.

The creation of initial links is a bootstrapping problem; we assume they are derived from the placement of agents in a network, or some other application-dependent source, and model them as a random network. In our simulations each agent starts out a cluster of a single item with links to some other randomly chosen agents. As a simulation progresses, agents pick some of their links to become *matches*, or *matched links* based on the similarity of the agents joined by the link. Clusters choose the best of these matches proposed by their agents to become connections. Agents joined by a path of connected links form a single cluster.

The creation of connected links allows clusters to expand, but the initial clusters formed in this way are very poor. They represent the best clusters agents can see in an extremely limited local view. We give agents two behaviors that are used to improve clusters. First, agents in a cluster combine their individual pools of links, widening their individual neighborhoods. This allows them (still individually) to pick better matched links as candidates for cluster membership. Additionally, to prevent agents conglomerating into one large cluster, a limit is placed on cluster size. A further procedure then allows clusters to break weaker connections, enabling them to upgrade stronger available matches into connections. Since connections are between agents, breaking a connection splits a cluster, but leaves other stronger agent pairs connected in the resulting clusters.

In previous research we concentrated on the matchmaking abilities of this model [12, 11]. Following this work, we represent the search for good matches between similar agents as a matchmaking problem among agents' short-term objectives. Internally each agent is considered to have a main *attribute* that describes its basic characteristics. We would like to cluster agents according to these attributes. Each agent further contains a number of *objectives*, or current goals based on its attribute. In the experiments in this paper an attribute is abstracted as a two-dimensional point. Objectives are also represented by points, chosen as a function of their agent's attribute. Objectives thus form a cloud of points around an agent's central attribute. For two-dimensional points objectives simply extend an agent's range of influence. For higher dimensional data or complex data, on the other hand, objectives can be chosen to reduce dimensionality and thus reducing the cost of checking for matches. For instance, an objective could be only one of many tasks that an agent needs to complete to reach a final goal. Figure 1 shows a diagram of two clustered agents. Figure 2 shows the links between all agents in a small system containing four clusters of ten points each.

Note that agents pick matches, since they are best able to determine how closely related their objectives are to other agents' objectives. Clusters, however have a wider view of relative closeness on the attribute level since they contain a larger number of matched and connected links. Thus clusters have a stronger basis on which to choose connections to make and break.

Multi-agent systems are often described in terms of their agents' individual behaviors. Along these lines, we can summarize the

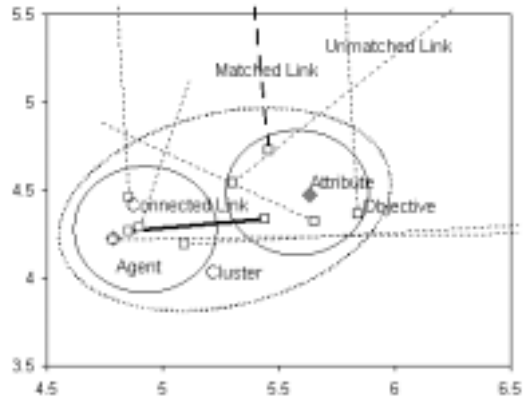


Figure 1: A diagram of two clustered agents.

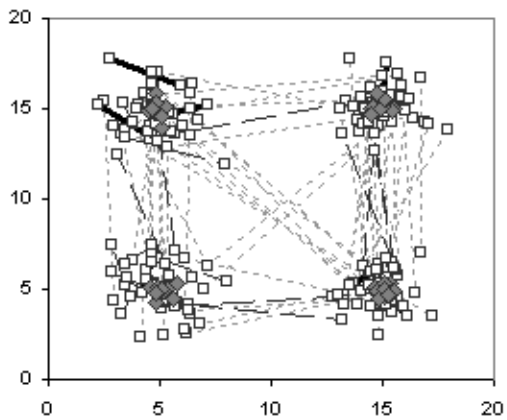


Figure 2: A diagram of a small system.

above agents as having the goal of finding good matches for their objectives. We would like the individual goals of these agents, along with the local coordination provided by cooperation within the clusters, to result in the overall system self-organizing into a grouping of agents based on their attributes. In this paper we focus on the nature of this collective clustering behavior. For this reason agents are limited to having very simple decision processes. This allows us to highlight the basic clustering behavior and gives us a good foundation on which to study more complex agents that might be used in real applications. This paper does not discuss the exact communication required within clusters. We studied this problem for the small clusters in [11], larger clusters require further research.

An example application for which our approach could be used is the classification of text documents spread over the web. Agents could each represent one document, with initial links created by references within the documents, or the storage location of the documents on web servers. Agents could choose matches for their documents based on the number of words they have in common, or on important keywords. These matches could then be presented to clusters along with normalized values indicating the agent's belief in the similarity of the documents. Clusters themselves would then represent queries to find documents similar to one of their members, and could be limited in size based on a suitable number of hits for such a query to return. In a more complex application agents could represent people, links acquaintances, matches how much time two people spend together, and clusters cliques of friends. For now, however, we shall limit ourselves to the more mundane world of 2D spatial data points.

3.2 Simulation Definition

Our objective is to examine the ability of a multi-agent system to find clusters in a set of points $P = \{x_1, \ldots, x_n\}$. A set of n agents $A = \{a_1, \ldots, a_n\}$ is created. Each agent a_i has as its attribute the point x_i in P. The agents are connected by links, forming a graph $G = (V, E)$ where the nodes are agents and the edges are links. We stipulate that each node in the graph G has the same degree δ. The interaction of the agents will change the edges in G, and eventually yield a new graph $G^* = (V, E^*)$. Connected components in G^* will correspond to clusters of P.

The procedure is as follows. Each agent a_i is given a set W_i of δ objectives, which are points (not necessarily in P) chosen as a function of x_i. Note that in the experiments presented in this paper we use $\delta = 5$ and all objectives of an agent a_i are simply given the value of the agent's attribute, x_i. To initiate the system we chose for each objective $\omega_i \in W_i$ an objective $\omega_j \in W_j$ of a different agent uniformly at random in such a way that no objective is paired twice. This pairing of objectives leads to an initial set of *unmatched links*, denoted as E_0^-. The initial set of *matched links*, denoted as E_0^+, is set equal to the empty set. The initial set of *connected links*, E_0^*, is also empty, indicating that to begin with each agent forms a *cluster* of size 1. From this position we proceed in turns, each turn t consisting of the following four steps. Some of these steps contain functions, which will be defined later.

Step 1 (Connecting): Clusters choose some of their matched links from E_t^+ to become connected links using a rule r_c. Together with all links from E_t^* this forms the edge set E_{t+1}^*. Note that a connected link remains in E_t^+.

Step 2 (Mixing): Each cluster C_i has a set of unmatched links adjacent to it, given by

$$E_t^-(C_i) = \{(\omega, \omega') \in E_t^- : \omega \text{ is an objective of an agent in } C_i\}.$$

Each cluster mixes its objectives that are adjacent to an unmatched link in $E_t^-(C_i)$, using a random permutation. After each cluster has completed this mixing procedure, a new set of unmatched links is obtained which is denoted by E_{t+1}^-.

Step 3 (Matching): All agents test their unmatched links (from E_{t+1}^-) using a turn-dependent matching probability $p_t^+(\omega, \omega')$. More precisely, an unmatched link (ω, ω') will become a matched link with probability $p_t^+(\omega, \omega')$ and will remain unmatched with probability $1 - p_t^+(\omega, \omega')$. The new matched links together with E_t^+ form E_{t+1}^+, and are taken out of E_{t+1}^-.

Step 4 (Breaking): Clusters choose some of their matched links from E_{t+1}^+ to be broken, downgrading them to unmatched links and adding them to E_{t+1}^-, according to a breaking probability p_b. Each broken link is then removed from the set E_{t+1}^+. If a link to be broken is also a connected link, it is also taken out of E_{t+1}^*.

The connecting, mixing and breaking steps must be done collaboratively by a cluster as a whole, while the matching step can be done separately by each of a cluster's agents. In our simulations, to simplify operations within in a cluster, we elect one cluster agent to perform the collaborative steps. Over many turns the mixing and matching steps above create a search for matches among the objectives of neighboring clusters. The connecting and breaking steps result in clusters forming and changing over time.

To determine $p_t^+(\omega, \omega')$ each agent maintains a *range*, which it continuously adjusts as follows. Let R_t^i denote the range of a_i

for turn t. The agent a_i considers M distances between objectives $\omega \in W_i$ and $\omega' \in W_j$ presented to it in the matching step. This might take several turns. After the M distances have been observed let σ be the smallest observed value. If $R_t^i \geq \sigma$, the agent forgets its distances and starts collecting M new distances. Meanwhile R_t^i stays the same. On the other hand, if $R_t^i < \sigma$, the range is gradually increased in the next M turns by a fixed fraction $(\sigma - R_t^i)/M$. However, if after say m turns, the agent is presented with a distance σ' smaller than the current σ, it repeats the test $R_i^t \geq \sigma'$ and follow the above procedure from that point on. The above procedure increases the range of the agent a_i. To decrease it, when a match is made the range is set to the distance of this match. In our experiments, $M = 100$.

For each turn t, we now let $p_t^+(\omega, \omega') = 1 - d(\omega, \omega')/R_t$. Here, $d(\omega, \omega')$ denotes the Euclidean distance between ω and ω', and $R_t = \max\{R_t^i, R_t^j\}$, if $\omega \in W_i$ and $\omega' \in W_j$.

The rule r_c is defined as follows. Let the *strength* of a matched link (ω, ω') be defined as $1/d(\omega, \omega')$. All current matched links in the cluster, that are not connected, are first ordered according to their strength. We then proceed to create connections, starting with the strongest. A connection is created if the resulting cluster is not larger than a size limit L. Once a connection cannot be formed because of this size limit no more connections are formed. In the experiments in this paper $L = 150$.

To define the breaking probability we need a speed parameter λ. Consider a cluster C consisting of N_C agents. Each turn the cluster has a probability of breaking *one* of its links given by: $p_b(C) = \lambda N_C/L$. The cluster chooses which link to break out of its set of matched links, $E^+(C)$, according to the following formula. Let $s(l)$ denote the strength of the link l, and let $s_{\max}^C$ be the maximal strength of a matched link in C. The weight of a link is defined as:

$$w(l) = (\frac{1}{s(l)} - \frac{1}{s_{\max}^C})^2.$$

The probability of the cluster choosing a link l is then given by:

$$p_b(l) = \frac{w(l)}{\sum_{l' \in E^+(C)} w(l')}.$$

4. EXPERIMENTAL RESULTS

4.1 Experimental Methodology

The following experiments cluster data sets of varying numbers of points. These data sets are generated according to the procedure described in [16]. Each data set consists of K clusters of 2-dimensional data points. A cluster is characterized by the number of points per cluster ($n_{\text{low}} = n_{\text{high}} = 100$) and the cluster radius ($r_{\text{low}} = r_{\text{high}} = \sqrt{2}$). The *grid* pattern is used, which places the cluster centers on a $\sqrt{K} \times \sqrt{K}$ grid. The distance between the clusters is controlled by k_g, which is set to 8. The noise parameter is set to 0. This creates a grid of well separated, circular, 2D clusters with 100 points each and equal density.

Four data sets are generated with 25, 100, 400 and 1600 clusters labelled 5×5, 10×10, 20×20, and 40×40 respectively. A corner of the 20×20 data set is shown in Fig. 3. We compare the quality of clusters found by our method to the generated clusters (perfect case), a set of clusters of the correct size but with randomly assigned points (random case) and the clusters generated by the Forgy k-means algorithm as described in [5], given the correct value of k, initial centers picked uniformly at random from all the points, and run until no further improvement in clustering is found.

Several measures of cluster quality are compared. First, the total square error metric, E^2, which is used by the k-means algorithm. Given k clusters $C_1, \ldots, C_k$, where C_i has a mean value m_i for $1 \leq i \leq k$,

$$E^2 = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - m_i\|^2.$$

Total square error gives an easily computed measure of the compactness of clusters, but in doing so favors small clusters. In fact, clusters with a single point have a square error of zero and thus the total square error alone cannot be used to compare clusterings of a data set with different numbers of clusters. Total square error also cannot be used to compare data sets of different sizes. The more points there are in a data set and the larger the range over which the data set is spread, the larger the total square error. The *weighted average cluster diameter* used by Zhang *et al.* [16], gives an alternative measure of the compactness of clusters. The *average pairwise distance* for a cluster C with points $P = \{x_1, \ldots, x_n\}$ is given by:

$$D = \frac{\sum_{i=1}^{n} \sum_{j=i}^{n} d(x_i, x_j)}{n(n-1)/2},$$

where $d(x_i, x_j)$ is the Euclidean distance between x_i and x_j. The weighted average cluster diameter for k clusters is then give by:

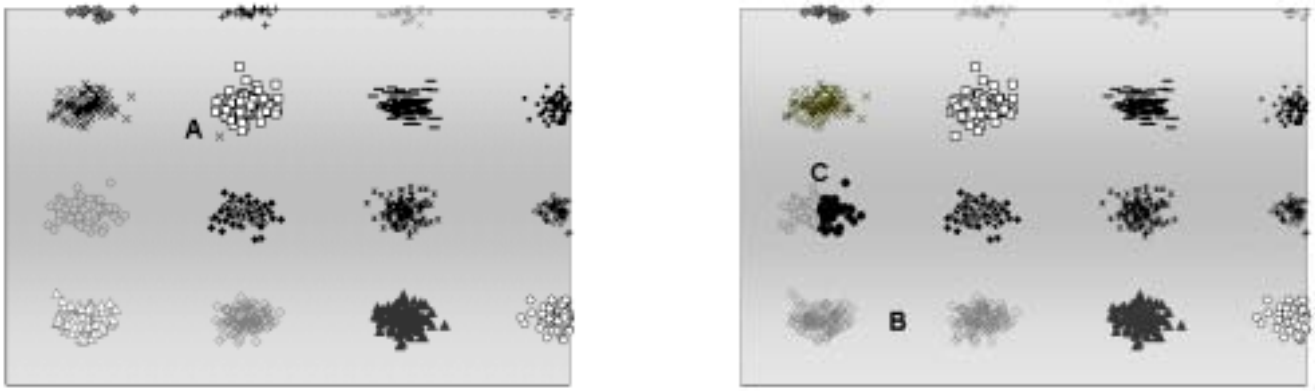
$$\overline{D} = \frac{\sum_{i=1}^{k} n_i(n_i - 1)D_i}{\sum_{i=1}^{k} n_i(n_i - 1)}.$$

While this measures scales with the number of data points and the number of clusters it also favors smaller clusters and does not account for singleton clusters.

Jain and Dubbes [5] describe several measures for comparing two clusterings of the same data by creating a contingency matrix which lists the number of points in common between each pair of clusters, between the two clusterings. These measures consider cluster membership, rather than the distance over which clusters are spread. We use this method to compare our experimental clusterings to the perfect clusterings produced by the generator. The Rand statistic [5] sums the number of pairs of points that are correctly placed in the same cluster and the number of pairs of points that are correctly placed in different clusters and normalizes by the total number of possible pairs. However, for our data sets which have many small clusters, the number of pairs correctly placed in different clusters dominates. Thus we also use the contingency matrix to calculate the number of points incorrectly placed by associating with each real cluster the found cluster with which it has the most points in common. We sum the number of common points over all real clusters and subtract from the total number of points to get the points that are out of place. This gives a clearer distinction between clusterings that are close to, but not quite, correct. On the other hand it does not distinguish clusters that are incorrectly grouped into a single cluster. It also can count up to half of the points in a real cluster as incorrectly placed if that cluster is simply split in two.

4.2 Types of Clusters Found

Figure 3(a) shows the clusters found by our agent procedure in a sample run, for a section of the 10×10 cluster grid. Figure 3(b) shows, for comparison, the clusters found by an example k-means run. Both methods were given the correct input parameters. The agent-based procedure used a maximum cluster size, L, of 150 and $\lambda = 0.3$ The k-means algorithm was run with $k = 100$. Overall both methods found the correct number of clusters, meaning that k-means did not loose any, which is possible, and that our agents adjusted to the correct cluster size of 100 instead of staying at the maximum size of 150. Total square error for the k-means run was 126,889 versus 20,372 for the agent-based method.



(a) Clusters produced by the agent-based procedure.

(b) Clusters produced by the *k*-means algorithm.

Figure 3: Agent-based and *k*-means clustering results.

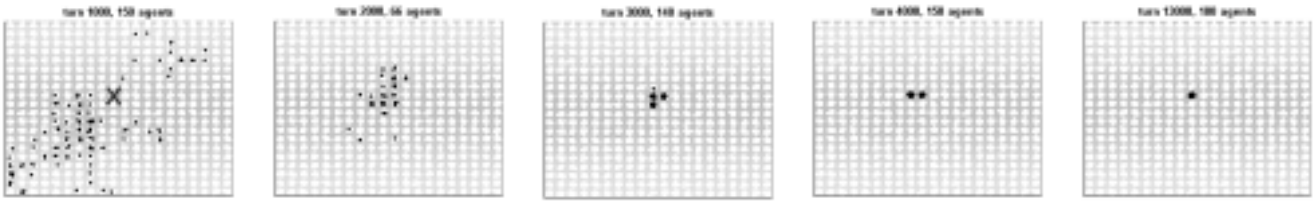


Figure 4: Clusters containing the point X (shown in turn 1000) over time.

The graphs in Figure 3 show the typical mistakes made by each method. At point A our method has associated a point with a neighboring cluster instead of with the correct cluster. This can happen when a point has joined the correct cluster, but has been broken off again due to the randomness of the breaking function. Generally these points reattach, though they can spend some time as a member of a neighboring cluster. Thus at any point in time after clusters have been found, several such mistakes are likely to exist. This type of mistake can also occur when a point simply does not find its correct cluster. This results in clusters with one or a small number of agents that can take a long time to discover their correct group. This problem is related to the speed factor, λ, in the breaking function, as discussed in Section 4.3. We see in Figure 3a that the *k*-means algorithm, by contrast, makes very different types of mistakes. At label B it has incorrectly joined two clusters into one, and at label C it has incorrectly split a cluster in two. This can occur when two initial centroids are chosen from the same cluster. This cluster then becomes split, but somewhere else two clusters need to be joined to maintain *k*. When there are large numbers of clusters it is likely that two initial centroids will be chosen from the same cluster. There are heuristic methods of choosing better initial centriods, however a perfect choice would amount to knowing the correct clustering a-priori.

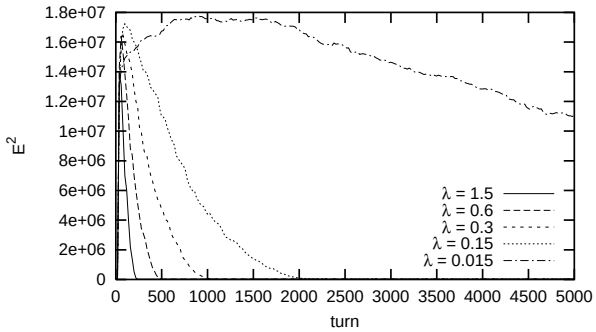
Figure 4 shows how the cluster containing the point X develops over time for the 20×20 cluster data set. Notice how the cluster is initially wide spread and contracts over time. Also notice that the cluster when expanded stays near its maximum size of 150 agents but that once it has contracted it adjusts to the correct size of 100 agents. In other experiments we found that agent clusters will adjust to the largest real cluster size under their maximum size. Thus when L=199 points, agents can correctly find clusters of 100 points but a size limit of 200 points will result in the combination of neighboring real clusters to create 200 point agent clusters. If the maximum cluster size is too small, data clusters become split into two or more neighboring areas, in much the same way as clusters are accidentally split in the k-means algorithm.

4.3 Rate of Finding Clusters

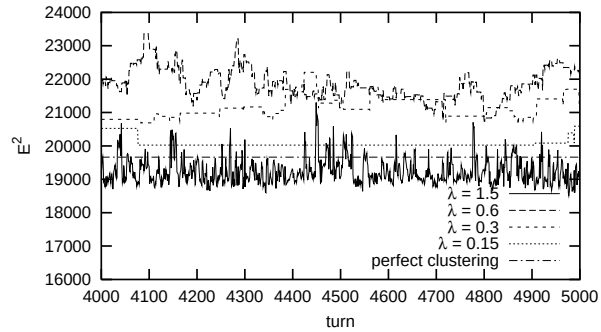
Figure 5 shows the total square error as a function of time for several example runs on the 10×10 data set. Time is measured in turns as defined in Section 3.2. The total square error, E^2, begins high when clusters are spread out, (though it is initially 0 as all clusters begin at size 1) but rapidly converges to near its optimal value. This convergence is followed by a tail behavior where clusters improve slowly until an equilibrium is reached. Note that E^2 does not precisely correspond to the quality of clusters. A better clustering can have a higher E^2 than a worse one, and during the tail there is a period where E^2 remains approximately constant while the clustering is actually improving slowly.

Figure 5 depicts the convergence for several values of the speed parameter, λ, used in the breaking function. Figure 5(a) shows that increasing λ increases the rate of convergence. However Fig. 5(b) shows that λ also effects the stability of the equilibrium found in the tail. A slower speed results in more stability. $\lambda = 1.5$ results in the lowest E^2 value, but this is because many small clusters are created. The run with the most stable equilibrium, $\lambda = 0.15$, is also the one with the best clustering. This can be seen by comparing the distribution of cluster sizes for $\lambda = 1.5$ and $\lambda = 0.15$ in Fig. 6.

From Fig. 6, which shows the distribution of cluster sizes at turn 5000 for two sample runs, it is apparent that the solution found when $\lambda = 1.5$ has a large number of very small clusters. This occurs when initial clusters contract too quickly. It is then possible for small sets of agents to become isolated. Since these sets of agents are quickly broken off of any neighboring cluster they join, they never gets the benefit of the wider view available in a large cluster. Thus, they can take a very long time to find their parent clusters. For the remainder of our experiments we use $\lambda = 0.3$ as it represents a good balance between rapid convergence and quality of the end solution.



(a) Square error over time.



(b) Tail of Fig 5(a).

Figure 5: Square error over time for different speeds (λ).

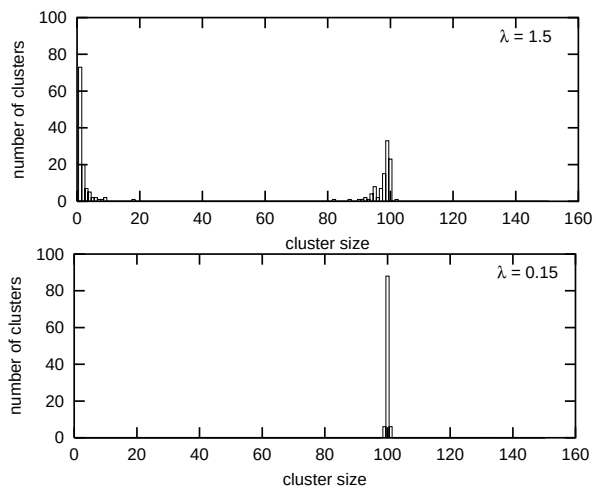


Figure 6: Distribution of cluster sizes: $\lambda = 1.5$ and $\lambda = 0.15$.

4.4 Increasing System Size

Figure 7 shows how the speed of convergence changes as system size increases. In this figure we plot, per time, square error normalized by the number of data points, N, and the area, A over which those data points are spread. Each data set represents an averaged value over 50 trials. The different test data sets are generated with 5×5, 10×10, 20×20, and 40×40 clusters, and thus with 2,500, 10,000, 40,000 and 160,000 points. We consider the smallest data set, 5×5, to have and area of 1. The larger data sets, 10×10, 20×20, and 40×40, thus have an $A = 4$, 16, and 32 respectively. The number of points in a cluster is kept constant when increasing system size. Preliminary experimentation suggests that agent-based clustering of systems with the same number of points but different numbers of clusters, possibly produces different behavior. A few large clusters are much easier to find than a large number of small clusters.

Figure 7 shows that while larger systems continue to converge rapidly, their improvement phase becomes more drawn out. Figure 8 shows the tail behavior of the data sets in Figure 7. Here E^2 is normalized only by N, hence $\langle E^2 \rangle = E^2/N$, since in all of the data sets real clusters are equally far apart and thus $\langle E^2 \rangle$ for the perfect clusters is about the same. In Fig. 8, the average square error, $\langle E^2 \rangle$, arrived at its equilibrium value for the 2,500, 10,000, and 40,000 agent data sets, while for the 160,000 agent data set $\langle E^2 \rangle$ is still slowly improving. There is a small increase in the equilibrium value of $\langle E^2 \rangle$ for larger data sets.

Figure 9 graphs the time it takes to reach certain values of $\langle E^2 \rangle$, again averaging $\langle E^2 \rangle$ over 50 trials for each point. This shows more clearly that better quality solutions take increasingly more time to find as the number of points increases. In general, however, the cost of reaching a particular solution quality grows less than linearly with system size. This is an improvement on the linear costs of the k-means algorithm. Table 1 lists all the statistics for the different system sizes, run for 5000 turns with 50 trials for each data set, compared to the k-means algorithm run 100 times for each data set, the values for perfect clusterings, and the values for random clusterings. While the agent-based clustering system usually does not find the perfect clusterings, it consistently improves upon the clusters found by the k-means algorithm. For smaller systems it places more than 99% of the points correctly and for the larger system, which was still improving, it places at least 95% of points correctly.

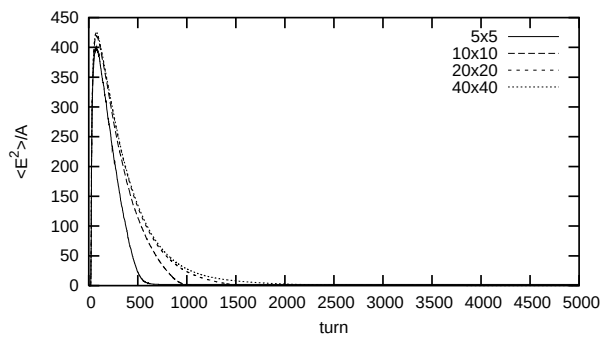


Figure 7: $\langle E^2 \rangle / A$ for different system sizes.

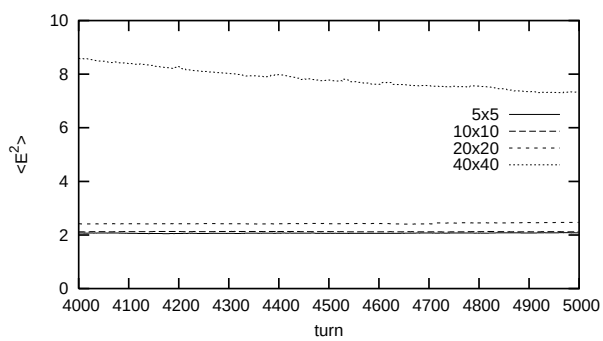


Figure 8: $\langle E^2 \rangle$ for the tail of Figure 7.

5. CONCLUSION AND FUTURE WORK

The experiments reported in this paper indicate that decentralized agent systems can indeed be used to find clusterings of large data sets in a reasonable amount of time, and with surprisingly good quality. Our method contains two parameters, the maximum

	number of clusters			E^2			$\langle E^2 \rangle$	$\overline{D}$			Rand	points out of place			% total
number of points (agents)	min	avg	max	min	avg	max	avg	min	avg	max	avg	min	avg	max	avg
PERFECT CLUSTERINGS															
2,500		25			5044.03		2.02		1.79		1		0		0
10,000		100			19662.99		1.97		1.77		1		0		0
40,000		400			79350.99		1.98		1.77		1		0		0
160,000		1600			315993.13		1.97		1.77		1		0		0
RANDOM CLUSTERINGS															
2,500	25	25	25	1.27E+06	1.27E+06	1.28E+06	509.64	28.86	28.96	29.06	0.92354	2272	2289.61	2304	91.58
10,000	100	100	100	2.09E+07	2.09E+07	2.09E+07	2090.38	58.58	58.69	58.78	0.98030	9560	9576.37	9590	95.76
40,000	400	400	400	3.37E+08	3.37E+08	3.38E+08	8429.88	117.76	117.86	117.97	0.99504	38920	38943.87	38978	97.36
160,000	1600	1600	1600	5.40E+09	5.40E+09	5.41E+09	33773.95	235.79	235.90	236.01	0.99876	156744	156772.57	156811	97.98
KMEANS: k=100															
2,500	23	24.87	25	17465.77	29131.18	43853.31	**11.65**	3.14	4.13	5.75	0.98508	57	146.30	235	**5.85**
10,000	97	99.22	100	88574.80	119873.85	162271.79	**11.99**	3.55	4.20	5.13	0.99617	368	576.88	857	**5.77**
40,000	392	396.78	400	412300.88	485795.34	564576.17	**12.14**	3.87	4.25	4.66	0.99903	1939	2327.25	2730	**5.82**
160,000	1580	1588.54	1598	1778912.79	1923502.03	2083301.02	**12.02**	4.02	4.22	4.43	0.99976	31369	34398.64	37667	**21.50**
AGENTS: L = 150, λ =0.3															
2,500	25	25.04	26	5044.03	5204.59	5553.44	**2.08**	1.79	1.80	1.82	0.99987	0	2.02	7	**0.08**
10,000	100	100.02	101	20214.65	21204.09	23030.86	**2.12**	1.78	1.79	1.81	0.99995	6	12.42	24	**0.12**
40,000	399	400.3396226	407	87979.07	98732.89	172885.77	**2.47**	1.80	1.83	1.95	0.99998	46	104.13	216	**0.26**
160,000	1610	1625.613636	1647	1057818.08	1172274.08	1368215.61	**7.33**	2.59	2.70	2.82	0.99990	6320	7123.93	7996	**4.45**

Table 1: Experimental data summary.

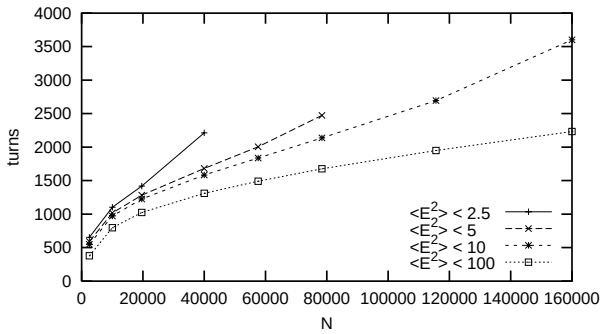


Figure 9: Time to reach various $\langle E^2 \rangle$ values with increasing systems size N.

cluster size and the speed at which clusters break, that can be adjusted to trade off knowledge of the data set and time for cluster quality. Agent clusters show an ability to adjust their size to the size of underlying data clusters, and to learn the appropriate range for matching. This demonstrates how using rational autonomous agents, which can modify their decision making criteria over time, can be advantageous in the clustering problem. It might also be possible to program agents to learn appropriate cluster sizes by watching the series of links they make and break, or to adjust the rate at which clusters change over time based on the current cluster quality. In this manner, agents might be able to find clusters of varying size, shape and density, thus tackling the issues addressed in more advance clustering algorithms.

The agents studied are extremely simple and clustered straightforward data sets. However, they demonstrate the basic dynamics of forming clusters based on similarities between agent attributes. For more complex data types, such as text or agent personalities we need only replace the matching function. Exact distance functions may not exist in these cases, however the probabilistic nature of the agents' behavior should tolerate a less precise definition of a "match". How well this works in practice must be determined by future research. However, some such form of decentralized agent grouping may in the future provide the basis for peer-to-peer directory services.

Acknowledgments

This research is supported by NLnet Foundation, http://www.nlnet.nl.

6. REFERENCES

[1] Decker, K., Sycara, K., and Williamson, M.: Middle-Agents for the Internet. Proceedings of the 15th International Joint Conference on Artificial Intelligence (1997). 578-583

[2] Ester, M., Kriegel, H., Sander, J., and Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Nose. Second International Conference on Knowledge Discovery and Data Mining (1996) 226–231

[3] Forner, L.: Yenta: A Multi-Agent, Referral-Based Matchmaking System. The First International Conference on Autonomous Agents (1997) 301-307

[4] Guha, S., Rastogi, R., and Shim, K.: CURE: An efficient clustering algorithm for large databases. Information System Journal, Vol 26, No. 1, (2001) 35-58

[5] Jain, A., and Dubes, R.: Algorithms for Clustering Data. Pentice-Hall advanced reference series. Prentice-Hall, (1988)

[6] Jain, A., Murty M., Flynn, P.: Data Clustering: A Review. ACM Computing Surveys, Vol 31, No. 3, September (1999). 264-322

[7] Judd, D., McKinley, P., and Jain A.: Large-Scale Parallel Data Clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol 20, No. 8. August (1998). 871-876

[8] Kaufman, L. and Rousseeuw, P.: Finding Groups in Data: an Introduction to Cluster Analysis, John Wiley and Sons (1990)

[9] Klusch, M. and Gerber, A.: Dynamic Coalition Formation Among Rational Agents. IEEE Intelligent Systems. Vol 17, No. 3 (2002) 42-47

[10] Ng, R., and Han, J.: Efficient and Effective Clustering Methods for Spatial Data Mining. Proceedings of the 20th VLDB Conference, (1994) 144-155

[11] Ogston, E.,Vassiliadis, S.: A Peer-to-peer agent auction. First Int. Joint Conference on Autonomous Agents and Multi-Agent Systems (2002) 151-159

[12] Ogston, E.,Vassiliadis, S.: Matchmaking Among Minimal Agents Without a Facilitator. Proc. of the 5th Int. Conference on Autonomous Agents.(2001) 608–615

[13] Olson, C.: Parallel Algorithms for Hierarchical Clustering. Parallel Computing 21, (1995) 1313-1325

[14] Sandholm, T. and Lesser, V.: Coalition Formation among Bounded Rational Agents. 14th International Joint Conference on Articial Intelligence (1995) 662-669

[15] Shehory, O. and Kraus, S.: Task Allocation via Coalition Formation among Autonomous Agents. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (1995) 655-661

[16] Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH:A New Data Clustering Algorithm and Its Applications. Data Mining and Knowledge Discovery, 1(2) (1997) 141–182