Group Formation Among Peer-to-Peer Agents: Learning Group Characteristics

Elth Ogston, Benno Overeinder, Maarten van Steen, and Frances Brazier

Department of Computer Science, Vrije Universiteit
Amsterdam de Boelelaan 1081a,
1081 HV Amsterdam The Netherlands
{elth, bjo, steen, frances}@cs.vu.nl

Abstract. This paper examines the decentralized formation of groups within a peer-to-peer multi-agent system. More specifically, it frames group formation as a clustering problem, and examines how to determine cluster characteristics such as area and density in the absence of information about the entire data set, such as the number of points, the number of clusters, or the maximum distance between points, that are available to centralized clustering algorithms. We develop a method in which agents individually search for other agents with similar characteristics in a peer-to-peer manner. These agents group into small centrally controlled clusters which learn cluster parameters by examining and improving their internal composition over time. We show through simulation that this method allows us to find clusters of a wide variety of sizes without adjusting agent parameters.

1 Introduction and Background

Forming groups of similar agents can serve many purposes within a multi-agent system. Group membership can provide an alternative to directory services when performing associative matching, and the process of coalition formation requires a manner of identifying preliminary groups. Clustering is the abstract problem of dividing a set of data into groups of like items. The clustering problem, however, has been primarily studied in the centralized case of grouping items that have been gathered together in a database. In the context of multi-agent systems, on the other hand, we consider large groups of distributed agents [5] [8]. These agents ideally require only peer-to-peer interactions to perform their basic operations in order to achieve scalability, flexibility and independence of components. Centralized clustering is a difficult problem because clusters can have many unknown characteristics, such as size and shape, which make them difficult to define. Decentralized clustering is even more problematic because global information such as the size or range of a data set is unavailable. In previous work [6] we discussed a method by which agents could perform decentralized clustering, provided that we could predefine the desired number of items in a cluster. In this paper we consider a manner in which agents can learn cluster area and density, thus allowing them to perform on a much broader range of data sets without having to adjust parameters. We demonstrate, through simulation experiments, collections of agents representing two dimensional points grouping themselves. Initial experimental results show that these agents, without changing experimental parameters, can identify clusters or varying density with between 20 and 250 points.

The clustering problem has been widely studied [2], however it is usually assumed that points in the data set can be compared by some central "clusterer". In peer-to-peer systems, on the other hand, data is distributed widely over a network and this assumption no longer holds. Further, in multi-agent systems the notion of agent autonomy can exclude the use of a single central comparison function. By decentralizing clustering we thus exacerbate the problem of how to tell a computer the characteristics of the clusters that we wish to find. One of the great difficulties encountered when designing clustering algorithms is defining the term "cluster" in a way that a computer can interpret. Intuitively a cluster can be seen as a connected dense region of points, surrounded by a less dense region. Finding clusters is thus finding boundaries between density regions. Because these boundaries are generally fuzzy, computer algorithms need a more concrete definition. To achieve this additional restrictions are used to make clusters calculable.

The K-means algorithm [4], for instance, fixes the number of clusters in the data set and, hidden in its total error squared measure, makes the assumption that clusters are roughly spherical. Given this information, clusters will extend to their natural boundaries, provided that centoids are chosen correctly. The k-means passes and various starting heuristics are methods of estimating the "correct" centroids.

Hierarchical algorithms [2], (we will use the top down minimal spanning tree algorithm as an example), split clusters along the largest existing edge between points to obtain each level. By doing this they in essence say "if there is a density boundary, this edge must cross it". This method identifies boundaries nicely, but leaves the problem of choosing which level of the tree contains the correct clustering. By basing this on some statistic, like the total error squared or the number of clusters, additional assumptions are introduced that are, again, dependant on the data set.

Density based clustering specifically looks for density boundaries by walking through a data set from point to nearest point. Here it is clear to see that a definition of a boundary is required. DBSCAN [1] specifically says that clusters are areas with at least a given density. DBSCLADS [9] makes the assumption that clusters are of roughly uniform density, with a parameter to define "roughly."

Overall, the standard data set dependant "clues" used by clustering algorithms include:

– the number of clusters, or analogously, the expected cluster size,
– the minimum gap between clusters,
– the minimum density of clusters.

Each of these gives a global standard for the data set, but can be used in decision functions locally. This allows centralized clustering algorithms to be parallelized, provided that global information about the data set can be shared between processing units [7]. P-CLUSTER [3] for instance parallelized k-means by distributing each k-means pass and synchronizing centroid information between passes.

In [6] we showed how clustering could be performed by a *decentralized* multi-agent system, provided that we could set beforehand the maximum size of the clusters we wished to obtain. In fact, knowing any of the above clues makes the form of decentralized clustering that we studied fairly easy, as it gives clusters an indication of when they have reached their boundaries. The real challenge occurs when these clues are absent, or do not hold for all clusters in the data set.

In this paper we show how clusters that lack any central coordination between them can learn cluster parameters by watching their internal composition over time. This allows them to find clusters with varying cluster sizes and densities, and even to identify clusters with widely differing characteristics within the same data set. Whereas in [6] we study the problem of clustering in a fully decentralized way across possibly large networks and the speed of convergence of this clustering, in this paper we study the problem of dynamically determining when to continue or to stop clustering.

Section 2 of this paper summarizes our base peer-to-peer clustering algorithm from previous work. Section 3 discusses the information available to an agent cluster when attempting to discover correct data cluster parameters. We then describe how this can be used, given that clusters themselves are centrally controlled and assuming that a cluster can correctly detect when it internally contains more than one data cluster. Section 4 then describes a method for doing this detection without needing to know the complete details of the data held by the agents in the cluster, and provides some experimental results exploring the range of data sets that the resulting method can handle.

2 The Basic Decentralized Agent Clustering Method

When defining our agent grouping method we consider very large systems with at least thousands of agents, like the internet, which must be distributed because their data cannot be collected centrally for reasons of size, privacy or dynamics. We thus specify that our agents must communicate in a decentralized manner, though we allow some cooperative group operations to be done by a central elected member of the group. For this reason we consider agent groups that are much smaller then the system as a whole, containing tens to hundreds of agents. We assume a global addressing system exists, allowing agents to send a message to any other agent, but limit the memory available to agents to store addresses so that they only know of the existence of a few neighbors at any one time.

The problem of grouping similar agents is, in the abstract, a clustering problem. Clustering in general considers a set of data items, S that must be partitioned into subsets, s_1, s_2... s_k, such that items within each subset have more in common with each other then they do with points in other subsets. In the agent domain we consider each agent as an item with particular characteristics, and with the goal of finding other like agents. We depict these agents as having a main "attribute" which describes its overall properties, and a number of "objectives" which describe its current short term goals for which a "matching" objective in a partner agent must be found. For instance, our agents could represent documents where an agent's attribute would be the document text, its objectives could be what it currently considers to be keywords for that text, and matches could be based on the distance between keywords in a given ontology. To group our agents we allow them to independently find matches using a peer-to-peer search protocol. We then choose some of the stronger matches that are formed to link agents into clusters of cooperating agents. Over time these clusters update their agents' choice of matches and their own membership, based on better matches that are encountered through the search

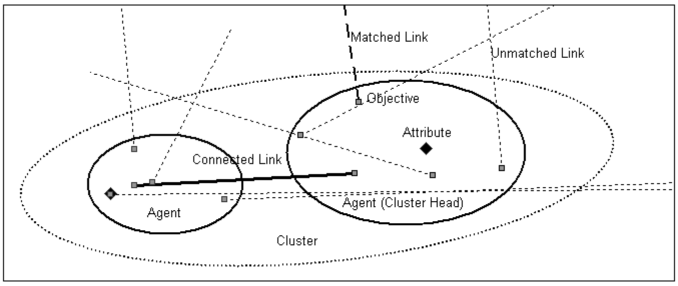


Fig. 1. A diagram of two clustered agents.

protocol. This process results in agents self organizing into clusters that approximate underlying clusters within their data. Figure 1 diagrams two clustered agents.

To create measurable experiments with these agents we consider the simple case where agents represent a set of two-dimensional points, generated to contain underlying data clusters. We initiate an experiment by creating one agent for each data point in the set. Each agent is given five elementary objectives, that are simply the data point. These objectives each use a matching function that measures the Euclidean distance between their point and that of a potentially matching objective. This distance is compared to a value learned over time indicating how small a distance represents a good match. To initiate an experiment we randomly pair up objectives, creating a graph in which the agents are nodes, each with five random edges to other agents. We call these edges "links." Links represent communication paths between agents. They can be of one of three types: "unmatched links" between two objectives that have nothing in common, "matched links" between two objectives that have agreed that they are similar, and "connected links" between strongly matched objectives. Initially each agent is considered to be an individual cluster.

Clusters simultaneously carry on two cycles of operations. The first is shuffling unmatched links to give objectives new potential matches. The second is making matched links into connections and breaking weaker connections so as to be able to replace them with better ones. Groups of agents that are joined by a path of connected links are defined as being a single cluster. They elect one member of the cluster as the cluster head, which keeps a map of the cluster composition and is thus able to make decisions that require an overview of the cluster as a whole. The cluster head is the only agent that knows the cluster composition, all other agents know only the address of the cluster head and the address of the five objectives that they are linked to in other agents.

The search for new matches is performed independently by the agents' objectives. This search is done by objectives first testing initial unmatched links to determine if a match should be formed. If an objective finds a match it informs its cluster head. Otherwise one objective in the link pair sends its neighbor's address to its own cluster head. The cluster head collects these rejected addresses, and after a time shuffles them, returning new addresses to the waiting unmatched links, who can then begin the process over again. During this search we maintain the invariant that any objective at any time has only one neighbor whom it can contact an who can contact it.

When two objectives agree on a match, they both inform their cluster head of the new matched link along with the Euclidean distance between their points. The cluster heads stores a list with details of all matched links in the cluster. It considers matches with a shorter distance to be better then longer ones.

The creation and breaking of connected links, which changes cluster composition, occurs “every now and then”, based on some internal timing in the cluster head. A cluster head first selects its best matched link and attempts to make it into a connected link by sending a request through the link to the neighboring cluster head. If the request is accepted, a new connection is agreed, one cluster head surrenders its data to the other, forming a single cluster, and the cycle ends. Clusters, however, have a maximum size that limits how many agents they can contain. If a connection request is refused because the resulting cluster would be too large both clusters consider operations to change their size. This involves checking if they want to break one of their connected or matched links. The cluster head chooses one of its longer links to break. Breaking a matched link does not effect cluster size, but leaves objectives free to find better matches. Breaking a connected link requires the cluster head to calculate a new cluster map. If this no longer forms a connected graph the cluster head appoints a new second head and sends it the information for the new cluster to be broken off.

The process above is described formally in our previous work [6]. In that paper we found that clusters will grow to their maximum size and over time consolidate to contain most of the points in an contiguous area. If the maximum size is set to be as large as a real data cluster and smaller than two neighboring data clusters, then the agent clusters will correspond to data clusters. Thus agents can self organize to find data clusters, with the limitation that the maximum size of the agent clusters must be set correctly beforehand. Most centralized clustering algorithms contain similar limitations, and thus we found that decentralized clustering compared favorably. The size of data clusters is, however, a parameter that can change for each data set, and even between clusters in a data set. Moreover, it is something that intelligent agents might be able to learn, giving an important advantage to agent clustering methods.

3 Growing Clusters

In order to be able to discover natural clusters in data, our agent clusters need a way to determine when they should increase or decrease their maximum size. To understand the operations open to an agent cluster we must first consider what data it has available. Imagine ourselves as a cluster, looking over the data landscape. Our entire view of the world is composed of our internal data and links to a few outside objectives. Agents in a cluster each know their own data point, and for each matched and connected link the length of that link. We should not assume that agents know the data point of the agent on the other end of a link since points may be complex or private collections of data.

From this picture a cluster as a whole knows:

- The density of the ‘found’ points for its region, i.e. the internal agents’ points.
- Possibly, the existence of some external points that suggest if its surroundings are of equal density, indicated by matched links of a short length.

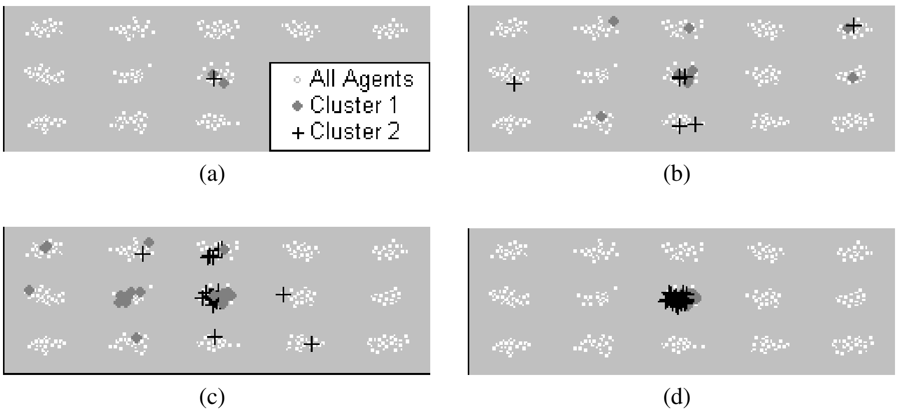


Fig. 2. The development of two clusters.

A cluster however does not know:

- The actual density of the area it covers, as there could be points in that area that have not yet been found.
- The actual density of its surroundings, since if a cluster only has distant matches, it does not mean that nearby ones do not exist.

From this information a cluster can compare its internal density to the density of its neighbors in the peer-to-peer system. However, unless this area of the system contains most of the relevant points form the data set, this tells us nothing about the data clusters. Luckily we can also assume that the majority of nearby points in the data set will group together. Our base procedure changes clusters by creating and breaking connections, favoring shorter connections, and thus naturally forms compact clusters over time. Figure 2 is a series of 4 time shots showing a system with data clusters containing 50 points each, being clustered by agents with a maximum cluster size of 25. We see how two agent clusters that end up splitting a data cluster grow over time. The last frame of figure 2 shows that a too small maximum cluster size results in a large data cluster being split into smaller regions. From this we may assume that an agent cluster contains most of the points in the area it covers after some time has passed and its composition has become stable. On the other hand, we can never be absolutely certain that this is the case since (i) we cannot be certain that cluster composition is stable at any point in time and (ii) in very large systems finding all the points in an area can take a long time. We will, however, use this assumption to grow the maximum cluster size, and make up for mistakes by shrinking clusters again should they become too large.

Figure 2 is a birds-eye view, not the actual picture seen by the cluster. To make cluster-wide decisions, including growing, we collect, in the cluster head, data from the individual agents. For the basic clustering algorithm the cluster head needs to store a map of the cluster which includes the addresses of agents in the cluster, the length of the matched links currently held those agents, and the length of the connected links between those agents. Figure 4(b) shows the ordered series of connected link lengths for the cluster whose data points are pictured in figure 4(a). This, along with a similar list with the length of the matched links, is the view of the world available to the cluster head.

Let us hypothesize that we have a function, shouldSplit(), that returns either that a cluster is a good size, or that certain links should be broken to form better, smaller clusters. We could then regulate cluster growth as follows: when considering a size change clusters first call this function to determine if they are currently too big. If so they spilt, resetting the maximum sizes of the resulting clusters. If not they should possibly be larger. To check this they call the shouldSplit() function again, this time also considering their best match as part of the cluster. If this still results in a good cluster we assume that there are other points that could belong to this cluster but do not because it is too small, and increase the maximum size.

The shouldSplit() function essentially calculates if an agent cluster's data belongs to one or more data clusters. Thus, given the data points in the cluster and the external data point for potential connections, in theory we should be able to use any existing clustering algorithm to implement it. Because the function is only used internally in relatively small clusters it is acceptable to use any of the centralized algorithms.

To test this method of cluster growth we ran some experiments with the resulting agent algorithm, using a "perfect" shouldSplit() function. This "perfect" function was given the minimum gap between clusters for a data set, and returned that any connected links longer then this gap should be broken. We generated 20 data sets using the the generation algorithm in [6], each containing 25 clusters with a fixed radius of 1, a gaussian internal distribution of points, and 20 to 200 points. We clustered these data sets, giving agents an initial maximum cluster size of 15. Measuring cluster stability to determine when to check for growth is difficult, and thus we simply guessed that after a long enough period the cluster will be about stable. Thus, we added to the algorithm in section 2 that every 10 times a cluster gets to the point where it checks if it should be breaking a link, it instead checks to see if it should change its maximum size, using the shouldSplit() function as described above. Figure 3 shows the resulting agent clusters for 15 of these data clusters. The majority of data clusters were found correctly. At point A an agent cluster has been incorrectly split into two. This occurs because the connections within a cluster are only an approximation of the minimal spanning tree for the cluster. These cluster will most likely rejoin after a time. At point B we see an agent cluster that has grown too large and now consists of two data clusters. Again, over time this mistake should also be corrected. Of the 500 clusters we generated we observed 23 incorrectly split clusters, and 15 too large clusters after running the algorithm for a fixed amount of time. We choose this time to be about twice as long as it takes clusters to initially grow to the correct size, thus giving them some time to improve. From this data we conclude that the above method is valid way of learning cluster size, provided that we can come up with a adequate real implementation of the shouldSplit() function.

4 Determining When a Cluster Is Too Large

As we discussed in the previous section, if we know some information about the data set, like cluster density or size, the shouldSplit() function is relatively simple to implement,

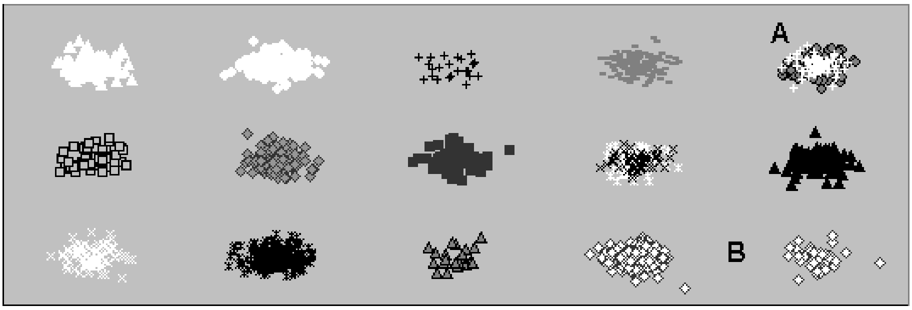
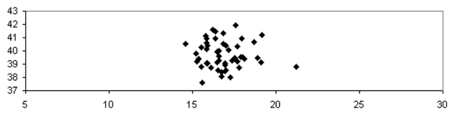


Fig. 3. Example clustering, given a perfect shouldSplit() function

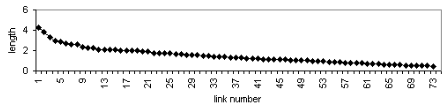
we only need to check if the current cluster conforms to the known criteria. Without this information a cluster essentially needs to run a clustering algorithm internally to decide if it should be one or more clusters. However, this isn't as easy as it sounds. Agents' attributes can be large or complex data points, such as entire text documents or user profiles, which would be expensive to store in the cluster head where shouldSplit() is calculated. Furthermore, when checking for cluster growth we also considered the nearest matched data point. This is data from an agent in another cluster, and might be private. For these reasons, in this section we experiment with a shouldSplit() function that only uses the data already available to the cluster heads.

In order to run the basic clustering algorithm cluster heads need to store some summary information about their matched and connected links: the lengths of those links. For estimating cluster density this information might be sufficient. Instead of clustering the data points themselves to implement shouldSplit(), as we suggested above, we could cluster the length of the connected links. Assuming that connected links are between nearby agents (in the data space), this would determine if the connected links form one set, indicating a continuous cluster density, or two (or more) sets, indicating that some links are over a larger gap between two data clusters. If two sets are found breaking all of the longer connections should split a large cluster into two (or more) good clusters. Clustering link lengths in this manner has the further advantage that it considers lower dimensional data, and is thus less computationally intensive.

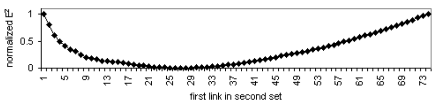
Clustering link lengths, however, requires that we can tell the difference between longer links between clusters and random fluctuations in link length, due to the random placement of points and the fact that connections are only an approximation of the minimum spanning tree between points. The examples in figures 4 and 5 show two clusters, one of the correct size, and one that should be split. These data clusters were created to have a random gaussian distribution of points, according to the algorithm described in [6]. Frame (a) of the figures shows the data points in the clusters, frame (b) shows the the lengths of each of the connected links in the clusters, sorted from largest to smallest. A comparison of the two figures shows that a large change between two consecutive links indicates that a cluster that should be split. However, graph 4(b) shows that the good cluster also contains relatively large changes in link length. We have found, though experimentation, that it is not enough to simply consider the ratio between the



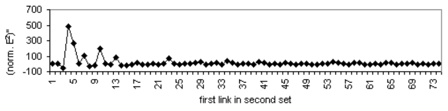
(a) Cluster points



(b) Ordered connected-link lengths

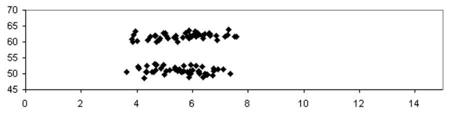


(c) Normalized E^2 series

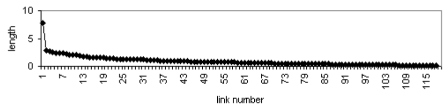


(d) Second derivative of 4(c)

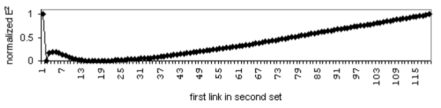
Fig. 4. A good agent cluster.



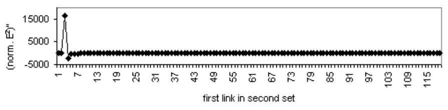
(a) Cluster points



(b) Ordered connected-link lengths



(c) Normalized E^2 series



(d) Second derivative of 5(c)

Fig. 5. An agent cluster that contains two data clusters.

largest drop and the average drop. Any value for this ratio high enough to avoid splitting the majority of good clusters, also failed to recognize some large clusters, eventually allowing the system to conglomerate into a single cluster.

In place of considering only the distance between consecutive lengths, we need a measure that takes into account all lengths when contemplating a split. To do this we assume that the ordered link lengths form two sets, those that are too long, and those that are good. Our task now is to determine where to divide the link length series into these two sets. We create a series of all the possible divisions, first with all lengths in the same set, then with the highest value in the first set and all others in the second, next with the two highest values in the first set, and so forth. Considering each of these groupings as a clustering of the lengths, we calculate a measure of the goodness of these clusterings, the total error squared used in the k-means algorithm . Total error squared is defined as:

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

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 sums the square of the distance between the vector of points in each cluster, and the average vector for that cluster, thus indicating how far a clustering is from a perfect clustering. Figures 4(c) and 5(c) show the resulting E^2 series from the example clusters.

A correct clustering of a data set has a lower E^2 value than an incorrect clustering. However, a clustering into two clusters will also have a lower E^2 then a clustering with one cluster. For this reason we cannot simply choose the partition of lengths with the lowest E^2. We also need to determine if an E^2 graph indicates that our lengths form one group or two. Consider the length series $\{1000, 1000, 1000, 1, 1, 1\}$. The correct partition $\{1000, 1000, 1000\}$, $\{1,1,1\}$ will have an E^2 of 0, while the incorrect partitions to either side of it in the E^2 series, $\{1000, 1000, 1000, 1\}$, $\{1,1\}$ and $\{1000, 1000\}$, $\{1000, 1, 1\}$ will have high E^2 values. This results in a discontinuity in the E^2 series. On the other hand, while the E^2 series for the length set $\{1000, 999, 998, 997, 996, 995\}$ also falls and rises, it does so gradually. Thus we calculate the second derivative of the E^2 series, E^2", depicted in 4(d) and 5(d). Large changes in length result in peaks in the second derivative. We use the approximation for the second derivative for an equidistant series of points: $f"(x) = (y_2 - 2y_1 + y_0)/h^2$. For the distance between points, h, we use $1/N$ where N is the number of connected links in the cluster. This scales the calculation by the size of the cluster.

A large agent cluster made up of two perfect data clusters each containing an infinite number of identical points and placed an infinite distance apart will exhibit a peak of infinity in the E^2" curve described above, while two correct agent clusters for this data will have a constant E^2" curve with value 0. We, however, do not have perfect clusters, and thus the large peaks in bad agent clusters will be well below infinity, while good agent clusters will still display small peaks. There are several reasons for this: data points within a cluster are spread out, clusters are close together, and our clusters have small numbers of data points making our estimate for the second derivative imperfect. On the other hand, all that is important is that we can find a cutoff value that distinguishes between too-high low values, and too-low high values. For the clusters in figures 4 and 5 any value between 500 and 15,000 would do. By normalizing the E^2 curve between 0 and 1 we make this cutoff value independent of the radius of the data clusters. However, the number of data points in a cluster does effect it. Small clusters with randomly placed points will show more variation in their strength values, raising the height of peaks that should be 0. Less data points will also result in a cruder estimate of discontinuities in the E^2", lowering the height of peaks. From this we observe that small clusters are much harder to distinguish then larger ones, and that a cutoff that works for small clusters should also be sufficient for larger ones.

In experiments with a range of data set we find that a cutoff of 175 consistently produces good clusters. This value is fairly low, and sometimes results in links within good clusters being broken. However, as these links are usually the long ones that reach across the cluster breaking them still leaves a connected graph of shorter links intact. Agent clusters that have grown to include more than two data clusters can contain more then the assumed two sets of link strengths, resulting in several peaks in the E^2" series. A cutoff value of 175 however is low enough that it usually detects the last peak, splitting the cluster correctly. Due to the scaling of E^2, an extremely large link strength can

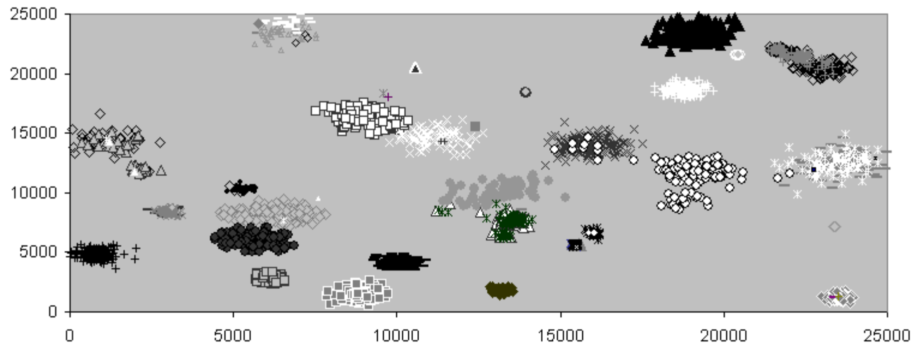


Fig. 6. Example clustering, using the shouldSplit() function form Section 4.

overshadow other large link strengths, however, since the clusters repeatedly check if they are too large, a cluster that is not completely split at one time will be split the next time round.

Figure 6 shows the resulting algorithm run on 36 randomly placed clusters, with radii between 1 and 1000, and 20 to 250 points. We see some of the same mistaken joins and splits seen in the more even clusters in figure 3. Overall, however, we find that our clustering algorithm does a good job of identifying the clusters, even when they are of very different size and density, and even in some cases where two such clusters are not well separated. This is a large improvement over our original agent algorithm which would have been unable to cluster this data due to the fact that the clusters have largely varying numbers of points.

5 Discussion and Conclusions

Overall, we have replaced a very data dependant clustering parameter, the expected maximum cluster size, with a cutoff parameter measuring sudden changes in link density, that allows agents to detect a much larger range of clusters with different areas and densities. We have shown some initial experiments indicating that this method is effective. However, we still need to fully explore the range of clusters that it can handle.

Ideally we would like agents to find clusters, within the same data set, independent of the number of points they contain, their density and the distance between them. The method we presented in this paper is dependant on the number of points, because of of our estimate of E^{2}". However, our clusters are even more limited in size by their internal communication, meaning that we only need to be able to deal with clusters with up to a few thousand points.

The method in this paper is also dependant on the ratio of the distance between points within a cluster and the distance between clusters themselves. Two clusters that are overlapping will look like the same cluster to most algorithms. Two clusters that are close together can also look the same to our algorithm since it doesn't consider all of the spatial data available. Thus we find a chaining effect where a string of points between two clusters will lead to them being considered as one. This is a common problem in density based clustering. Our algorithm will also be unable to distinguish small gaps between clusters whose density gradually decreases, as small increases in the length of links between clusters can look like part of the natural increase in link length within the cluster.

Finally, our method of cluster boundary detection is also dependant on the distribution of points within a cluster. The experiments here presented clusters with a random gaussian distribution, showing that we can deal with smoothly changing distances between cluster points. On the other hand, sudden changes will be detected as gaps between clusters. Such sudden changes can occur in very regular clusters, for instance in clusters with points that are placed evenly on a grid. All of the above issues require further experimentation to determine the exact range of clusters that our agents can now handle.

References

1. 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
2. Jain, A.K., Murty M.N., Flynn, P.J.: Data Clustering: A Review. ACM Computing Surveys, Vol 31, No. 3, September (1999). 264-322
3. 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
4. Kaufman, L. and Rousseeuw, P. Finding Groups in Data: an Introduction to Cluster Analysis, John Wiley and Sons (1990)
5. Klusch, M. and Gerber, A.: Dynamic Coalition Formation Among Rational Agents. IEEE Intelligent Systems. Vol 17, No. 3 (2002) 42-47
6. Ogston, E., Overeinder, B., Van Steen, M., and Brazier, F.: A Method for Decentralized Clustering in Large Multi-Agent Systems. Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (2003) To be published
7. Olson, C.: Parallel Algorithms for Hierarchical Clustering. Parallel Computing 21, (1995) 1313-1325
8. 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
9. Xu, X., Ester, M., Kriegel, H., and Sander, J.:A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases. Proceedings of the 14th International Conferece on Data Engineering (1998) 324-332