A Dynamic Clustering Algorithm for Mobile Objects

Dominique Fournier[1], Gaële Simon[2], and Bruno Mermet[2]

[1] LITIS EA 4051, 25 rue Philippe Lebon, BP 540 76058 Le Havre cedex
dominique.fournier@univ-lehavre.fr
[2] GREYC CNRS UMR 6072, 6 Boulevard du Maréchal Juin 14050 CAEN cedex
{gaele.simon,bruno.mermet}@univ-lehavre.fr

Abstract. In this paper, a multiagent algorithm for dynamic clustering is presented. This kind of clustering is intended to manage mobile data and so, to be able to continuously adapt the built clusters. First of all, potential applications of this algorithm are presented. Then the specific constraints for this kind of clustering are studied. A multiagent architecture satisfying these constraints is described. It combines an ants algorithm with a cluster agents layer which are executed simultaneously. Finally, the first experimental results of our work are presented.

1 Introduction

1.1 Agents Clustering

In this article, a dynamic technique for clustering mobile objects is proposed. Initially, this work aims at characterizing groups of agents during the execution of a multiagent system. A subset of agents is considered as a group if, for a given period, the internal properties of the agents evolve similarly. Each observed agent can be represented by a vector of properties evolving in time. For example, these properties can represent the number of communications of the agent, its reinforcement value (for agents with training capacities), or its position in a located environment (ants [1] for instance). Thus, groups of agents can be highlighted as evolutionary clusters. More generally, we have to solve a problem of dynamic clustering in which the cardinality of the set of data to cluster is not constant (some agents can appear or disappear during the clustering process). Moreover, already clustered data can be modified or reorganized according to the evolution of the corresponding agents. These characteristics also appear in other kinds of applications like meteorology (detection and follow-up of cyclones), road traffic analysis, animals migrations, ... Thus, a dynamic method of clustering is necessary in order to adapt the set of clusters continuously so that it reflects as accurately as possible the current state of data (of agents in our case).

1.2 Related Works

Recently, some works are focused on particular kinds of clustering where the set of data to cluster is not completely known by the algorithm at the beginning of the process. This is for instance the case of data stream clustering [2,3]. A data stream is a sequence of numerous data such as the log of actions of a website user. The large volume of data prevents the storage of the stream in main memory; this is why clustering algorithms have to manage the streams gradually. So, most of them work on subsets of streams constituted by several consecutive data. The first disadvantage of these approaches is that the number of clusters built remains generally constant. Moreover data already clustered cannot evolve with time, which thus does not correspond to our specific needs. There also exists evolutionary data streams clustering algorithms [4,5,6]. The main difference with the previous category is that, the underlying distribution of the data stream can evolve significantly with time. Thus, these algorithms have to be able to modify strongly the built clusters, and even to destroy some of them during the clustering process. This point joins one of our concerns. But, as in the first case, these algorithms do not take into account the fact that already clustered data can also evolve. Evolutionary clustering[7] also considers the problem of clustering data over time, even if in this case, at time t the clustering algorithm uses all objects seen so far. The main difference with our work is that, at time t, the set of clusters produced by our algorithm is expected to only characterize the current state of objects.

The closest work to our problem relates to an algorithm for clustering mobile data presented in [8]. This algorithm allows to cluster evolving data. It is based on moving micro-clusters which are adapted from the micro-cluster notion defined in [9]. The location of each object to cluster is described by its location at time t_0 together with a velocity vector. Using this two components, the current location of an object can be evaluated.

One major advantage of this algorithm is its efficiency when updating the set of micro-clusters thanks to their profile. Indeed, if micro-clusters are stable enough, the update can be performed using only profiles which summarize the state of each micro-cluster. Moreover, using time velocities as additional dimensions in the clustering process is very interesting.

The main disadvantage of this approach with respect to our constraints is the need to use a standard clustering algorithm jointly. The authors have chosen K-Means which needs to provide the number of clusters to build. Moreover, K-Means is first used to build initial micro-clusters. This implies that the set of all mobile objects to cluster must be known at the beginning of the clustering process which is not always possible in our context. The other important disadvantage is that the algorithm does not allow to take into account new mobile objects during the clustering process which is necessary in our agents clustering context.

2 Our Approach

In addition to the algorithms previously presented, there also exists in the MAS community, ants algorithms that are able to achieve clustering tasks [10]. In these algorithms, data are distributed in a grid on which ants agents can move.

They can also carry data and gather them in heaps. These algorithms have properties which seem to fit with our needs, in particular to take into account the evolution of data, which is necessary in our context. Unfortunately, they have a slow convergence, which is accentuated in an unstable context.

To improve the convergence of these approaches within a static framework, N. Monmarché proposed the AntClass algorithm [11] which successively associates in four steps an ants algorithm and the K-Means algorithm. This approach is not compatible with the dynamic aspect of our problem as shown in [12]. Nevertheless, we decided to associate a layer of cluster agents with the ants described in AntClass to make a two layers multiagent architecture. In this architecture, each heap created by the ants corresponds to a cluster and is encapsulated in a cluster agent. Thereafter, we describe the grid which constitutes the environment of our agents and the two kinds of agents : ants and clusters.

2.1 The Grid

The grid is divided into cells and allows to store non-clustered objects[1] and heaps. There are two kinds of non-clustered objects: objects which have never been put into a heap; objects which have been rejected from their initial heap by Cluster agents or by Ants (this can occur when objects evolve or when ants build inaccurate clusters).

2.2 Ants

Their behaviour is identical to the one specified in AntClass during the first stage of the algorithm. We summarize this behaviour quickly, more details being available in [11,13]. The ants move on the grid, can carry an object by picking up an isolated object found in a cell, or by taking the most dissimilar object of a heap. They can also drop an object o on a free cell, on an isolated object if they are similar enough (creating a new heap), or on an existing heap if o and the heap centre are similar enough. Similarity is evaluated with a distance measure. These actions are based on probabilities in order to produce a partial random behaviour which is fundamental for the effectiveness of ants algorithms.

Our architecture is based on the same ants algorithm as AntClass, yet we have defined a new measure to evaluate objects with respect to a cluster. This measure takes into account the dispersion of the distances of clustered objects from the gravity centre of their heap. It also allows to reduce the number of parameters used in AntClass while increasing the stability of the clustering. Thus, we suppose that the distribution of data in a cluster follows a Normal law represented by a variable R_C. So, considering the average μ and the standard deviation σ of R_C, by definition, one knows that 99.7% of data are in interval $I_3 = [0, \mu + 3\sigma]$. This led us to redefine the condition for an ant to remove the most dissimilar object o from a heap H: if the distance between o and the centre of H does not belong to the interval $[0, \mu + 3\sigma]$, then the ant removes o from H. Moreover, we have

[1] From here, when we talk about a piece of data, we will also use the word *object*.

also modified the mechanism used by ants to add data to heaps. Experiments shows that the aggregation criterion used by ants of AntClass leads sometimes to build inaccurate clusters which are difficult to remove. Thus a more restrictive criterion was defined: our ants drop an object in a heap if its distance from the centre of the heap is in the interval $I_2 = [0, \mu + 2\sigma]$.

2.3 Cluster Agents

In AntClass, after the work of ants, K-Means is used to reduce the number of remaining isolated data. In our approach, we dedicate the management of this problem to cluster agents.

A cluster agent encapsulates a heap C (i.e a cluster) created by ants. This one lives while its heap contains at least two objects. Moreover, just before its death, a cluster agent rejects its remaining objects on the grid. C is defined by a 4_tuple (G_C, R_C, V_C, S_C) where G_C is its centre, R_C its radius, V_C its volume (an hypersphere specified by given by G_C and R_C) and S_C the set of its data. In the following, C refers to a cluster agent and also to its encapsulated heap.

Main Behaviour. As soon as an ant drops an object in a heap or removes an object from a heap, the associated cluster agent must update its 4_tuple. Moreover, cluster agents can also grow by attacking other cluster agents perceived as obstructing their own development. A cluster agent C_{obs} is considered to *obstruct* an other cluster agent C_{att} when $V_{C_{att}}$ intersects $V_{C_{obs}}$. Moreover, we suppose that C_{att} attacks C_{obs} only if its size is at least equal to 20% of the size of C_{obs} (to avoid too many attacks). If such an intersection of volumes occurs, it is probably due to a misconfiguration of the clusters, maybe they could be unified, maybe one of them could be divided into two parts. Thus, the clusters must be updated. That is the reason why, when a conflict between two Cluster agents occurs, one of them attacks the other one which flees. As shown in next section, the flee mechanism allows to update the clusters.

Cluster agents must also detect the evolution of objects contained in their heap. If an object becomes too distant from the centre of its heap, the cluster agent rejects it into the grid and updates its 4_tuple. To determine if an object is too far away from the centre of its heap, a cluster agent uses the same measure as the one used by the ants (§ 2.2).

The Attack/Flee Behaviour. If a cluster agent called C_{att} has an obstructing cluster agent C_{obs}, C_{att} attacks C_{obs} and the former must flee. To describe the attack/flee interaction we use the following notations:

– $\delta(x, y)$ is the Euclidean distance between 2 objects x et y;
– $diss(C)$ is the most dissimilar object of C with respect to δ.

To manage the flight of C_{obs}, two kinds of situations are considered depending on whether the intersection between $V_{C_{att}}$ and $V_{C_{obs}}$ may contain objects or may not. In the first case, all objects located in the intersection are added to the heap of C_{att}. In the second case, C_{obs} drops one by one its $diss(C_{obs})$ until the intersection of volumes disappears. If $\delta(diss(C_{obs}), G_{C_{att}}) < R_{C_{att}} + 2\sigma$ then $diss(C_{obs})$ is put in C_{att} and on the grid if not. This aggregation mechanism of the objects into the attacking cluster during a flight is identical to the aggregation mechanism of data into a heap of the ants behaviour (§ 2.2). Here our aim is to reinforce homogeneity of C_{att} and to avoid the construction of too large heaps which are difficult to dissociate.

3 Experiments

3.1 Scenarios

In order to evaluate our dynamic clustering algorithm, a simulation platform is used to generate sets of mobile objects specified by "scenarios". A scenario allows to describe "populations" and "trajectories". A population is considered to be a set of mobile objects which are supposed to evolve similarly (i.e. to have a similar motion) and distributed inside an hyper-sphere. The description of the motion of this hyper-sphere is called the population's trajectory. As a consequence, at the clustering level, each detected cluster is expected to correspond to an existing population.

The main goal of the first experiments is to show the ability of our algorithm to detect the population trajectories as they are evolving. In this section, we have chosen to present three particular scenarios. The main differences between these scenarios are the number of populations, the kind of population trajectories used and the presence of noise. In the following, each scenario is summarized by a graphical representation (Fig. 1) inside which each population is represented by a circle. Bold and dotted circles represent respectively initial states and intermediate or final states of the populations. The trajectory of each population is illustrated by arrows. In each scenario, a population contains 100 mobile objects distributed according to a normal law. It is important to notice that, for the first experiments, only two attributes have been used for mobile objects in order to simplify the visualisation and the results analysis. The *Noisy* scenario contains a single population (Fig. 1(a)) and 33 additional mobile objects which are randomly distributed to simulate a kind of noise. They also move randomly during the scenario. The *Disjoined* and the *Crossroad* scenarios contains two different populations. These scenarios are rather different because in the *Disjoined* one, trajectories are such that populations converge without any overlapping in a first time and then move in two different directions. In the *Crossroad* scenario, populations not only converge but also overlap, then they also continue on their own ways.

3.2 Results

Evaluating the results produced by a dynamic clustering is a difficult problem because usual criteria defined only on the final results of the clustering process can not be used alone. More precisely, this implies to be able to compare, almost

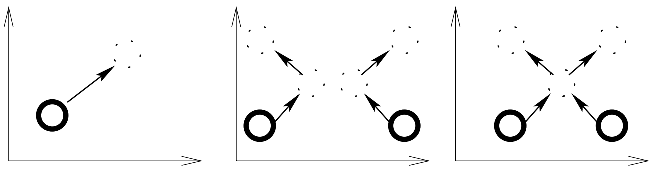
(a) Noisy (b) Disjoined (c) Crossroad

Fig. 1. Scenarios schemas

continuously, the state of the different populations of the scenario with the set of detected clusters. Moreover, the temporal gap between what is detected by the clustering algorithm and the different states of the populations in the simulation must be evaluated.

For the moment, three types of temporal graphs have been used which allow to analyse the following measures over time: the number of mobile objects put in clusters, the number of clusters and the mean clusters purity. The purity is a measure used in [5] and defined as follows:

$$purity = \frac{\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|}}{K} \tag{1}$$

where K is the number of detected clusters at time t, $|C_i|$ is the cardinality of cluster i and $|C_i^d|$ is the number of mobile objects of cluster C_i associated to the population d which is the most represented population in cluster C_i. It allows to evaluate, at time t, the mean quality of clusters with respect to populations in the simulation. Due to a lack of space, these graphs are not detailed in the paper. That's why, in the next section, results evaluation is presented using mean measures of the clustering process. Nevertheless, they take into account a potential evolution of the number of populations during the execution of the scenario.

For each experiment, the following measures have been used:

- #data: number of all mobiles objects in the scenario
- NAvgC: mean ratio between the number of detected clusters and the number of populations in the scenario at the same moment (the closer to 1 this measure is, the better is the clustering result)
- NAvgDC: mean number of aggregated mobile objects (i.e. the number of mobile objects which are put in a cluster).
- TimeNbP: amount of simulation time corresponding to states in which the number of detected clusters is equal to the number of populations in the scenario.
- PM: mean of the mean clusters purity.

It is important to notice that for these measures, a heap containing two mobile objects is considered as a cluster.

Table 1. Experiments results

Test	**#data**	**NAvgC**	**NAvgDC**	**TimeNbP**	**PM**
Noisy	133	1.5	93.0	77.0	0.97
Disjoined	200	1.15	180.0	84.9	0.99
Crossroad	200	2.5	132.6	35	0.7

Purity measures show that, generally, clusters are good ones i.e. they contain mobiles objects associated to the same population. NAvgC values are often too high but they show, however, that the number of detected clusters is close to the number of populations in the scenario. Indeed, this is also entailed by the analysis of graphs giving the number of detected clusters with time. This analysis shows also that, at the beginning of the simulation, the clustering algorithm needs a short adaptation period during which the number of clusters is big. The mean number of aggregated objects shows that a little part of the mobiles objects is not clustered at the end of the simulation. The results show that the kind of populations trajectories does not modify the results quality. Moreover, the good results obtained on *Noisy* scenario show that the clustering algorithm has not been disturbed by noisy objects.

However, the algorithm does not provide good results on *Crossroad* scenario. In this scenario, the 2 populations are temporarily superposed. At the clustering level, the consequence is that the two corresponding clusters are merged into a single big cluster SC which is a normal phenomenon. But, when in the simulation the two populations begin to move away, the clustering algorithm keeps the cluster SC instead of splitting it. Indeed, as there exists only one big cluster, it can not be attacked by the new little clusters built by ants. On the contrary, these last ones are attacked by SC and then are merged with SC. This phenomenon is a consequence of the attack constraint described previously which is however useful in most cases.

4 Conclusion

In this paper, a dynamic clustering algorithm for mobile objects (or agents) based on a multiagent architecture has been presented. This last one is made of an ants layer coupled with a cluster agents layer, the two layers being executed simultaneously. The first experiments on scenarios corresponding to various kinds of populations evolutions give good results. Nevertheless, new solutions must be found, particularly to solve the problem of unsplitable "big clusters". A first solution to explore is to use the velocity of mobile objects in the clustering process [8] which could avoid that two clusters corresponding to two populations with different dynamics are merged. Further experiments have also to be performed on new scenarios in which the number of populations and the size of these populations evolve significantly during the simulation. To better evaluate the accuracy of our results we plan to use the MONIC framework[14] which proposes an algorithm for cluster transition detection. Indeed, this framework can help us to compare more precisely scenarios and clustering results.

References

1. Drogoul, A., Corbara, B., Lalande, S.: artificial societies: The Computer Simulation of Social Life. In: Conte, Gilbert (eds.) MANTA: new experimental results on the emergence of (artificial) ant societies (1995)
2. Barbara, D.: Requirements for clustering data streams. SIGKDD Explorations 3(2), 23–27 (2002)
3. Guha, S., Mishra, N., Mortwani, R., O'Callaghan, L.: Clustering data streams. In: IEEE Annual Symposium on Foundations of Computer Science, pp. 359–366. IEEE Computer Society Press, Los Alamitos (2000)
4. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) Databases, Information Systems, and Peer-to-Peer Computing. LNCS, vol. 2944, Springer, Heidelberg (2004)
5. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: SIAM Conference on Data Mining, p. 11 (2006)
6. Nasraoui, O., Uribe, C.C., Coronel, C.R., Gonzalez, F.: TECNO-STREAMS: Tracking Evolving Clusters in Noisy Data Streams with a Scalable Immune System Learning Model. In: IEEE International Conference on Data Mining, pp. 235–242. Melbourne, Florida (2003)
7. Chakrabarti, D., Kumar, R., Tamkins, A.: Evolutionary clustering. In: KDD'06, USA (2006)
8. Li, Y., Han, J., Yang, J.: Clustering moving objects. In: KDD (Knowledge Discovery in Databases), pp. 617–622 (2004)
9. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. In: SIGMOD (International Conference on Management of Data) (1996)
10. Deneubourg, J.L., Goss, S., Franks, N., Sendova-Franks, A., Detrain, C., Chretien, L.: The dynamics of collective sorting: robot-like ant and ant-like robots. In: Meyer, J.-J., Wilson, S. (eds.) Proceedings of the First International Conference on Simulation of Adaptative Behavior (1990)
11. Monmarché, N.: Algorithmes de fourmis artificielles: applications á la classification et á l' optimisation PhD thesis, Université de Tours, France (2000)
12. Simon, G., Fournier, D.: Agents clustering with ants. In: Proceedings of 5th International Workshop on Agent-Based Simulation (ABS'04, pp. 147–152. SCS Publishing House (2004)
13. Coma, R., Simon, G., Coletta, M.: A multi-agent architecture for agents clustering. In: Proceedings of 4th International Workshop on Agent-Based Simulation (ABS'03), SCS Publishing House (2003)
14. Spiliopoulou, M., Ntoutsi, I., Theodoridis, Y., Schult, R.: The MONIC Framework for Cluster Transition Detection. In: Fifth Hellenic Data Management Symposium, Greece, p. 10 (2006)