Effects on clustering quality of direct and indirect communication among agents in ant-based clustering algorithms

Por

Ing. Marco Antonio Montes de Oca Roldán

TESIS

Presentada a la División de Electrónica, Computación, Información y Comunicaciones
como requisito parcial para obtener el grado académico de

Maestro en Ciencias en Sistemas Inteligentes

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Monterrey, N.L. Mayo de 2005

Instituto Tecnológico y de Estudios Superiores de Monterrey

División de Graduados en Electrónica, Computación, Información y Comunicaciones

Dirección de Programas de Posgrado en Electrónica, Computación, Información y Comunicaciones

Los miembros del comité de tesis recomendamos que la presente tesis de Marco Antonio Montes de Oca Roldán sea aceptada como requisito parcial para obtener el grado académico de Maestro en Ciencias en:

Sistemas Inteligentes

Comité de tesis:

Dr. Leonardo Garrido Luna
Asesor de la tesis

Dr. José Luis Aguirre Cervantes
Sinodal

Dr. Ramón Felipe Brena Pinero
Sinodal

Dr. David Alejandro Garza Salazar
Director del Programa de Graduados en Electrónica, Computación, Información y Comunicaciones

Mayo de 2005

A Rosa María Roldán Ramírez, mi queridísima mamá.

Acknowledgments

I'd like to thank all the people who have contributed directly or indirectly to the development of this thesis. I'll try not to omit any person involved nor their contributions, but if someone feels I left him/her out of my list, I apologize.

First, I'd like to thank Dr. Leonardo Garrido for the freedom he gave me to find and follow my own research interests. The discussions I held with him during my studies were very helpful in my academic, professional and personal life. Thank you very much Leonardo.

I'd also like to thank Dr. Ramón Brena for letting me work in the Distributed Knowledge and Intelligent Agents Research Group he leads. Although indirectly, he also gave me the opportunity to find a thesis and research topic I really enjoyed working on. He served as a member of my thesis committee, and I'd like to thank him all his suggestions for the improvement of this work.

Dr. José Luis Aguirre as a member of my thesis committee, equally helped me to improve this work. His enthusiasm was really encouraging during the writing of this thesis and some of the papers derived from it.

I'd also like to thank all the *losers* of the Intelligent Systems Laboratory. A special mention goes to (as they come to my mind): Richie, Carolina, Claudia, Alejandro, Marcelo, Mike, César V., Eduardo, Erick, Alberto, Iñaki, René, Escoffie, Emmanuel, Chuy, and of course, César Marín, my friend.

Even though she doesn't belong to the losers group, I want to thank Tere for her sincere friendship. Thank you.

Finally, I'd like to thank the Monterrey Institute of Technology for the scholarship that made this thesis possible and allowed me to meet all these nice people.

Marco Antonio Montes de Oca Roldán

Instituto Tecnológico y de Estudios Superiores de Monterrey
May 2005

Effects on clustering quality of direct and indirect communication among agents in ant-based clustering algorithms

Marco Antonio Montes de Oca Roldán, M.C.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2005

Thesis advisor: Dr. Leonardo Garrido Luna

Ant-based clustering algorithms are knowledge discovery tools inspired by the collective behavior of social insect colonies. In these algorithms, insects are modeled as software agents that communicate with each other indirectly through the environment. This particular kind of communication is known as stigmergic communication.

In the classic ant-based clustering algorithm, a group of agents that exhibit the same behavior move randomly over a toroidal square grid. In the environment there are data objects that were initially scattered in a random fashion. The objects can be picked up, moved or dropped in any free location on the grid. An object is picked up with high probability if it is not surrounded by similar objects and is dropped with high probability if an agent's neighborhood is densely populated by other similar objects and its location is free. Here, stigmergy occurs when an object is placed next to another. The resultant structure is much more attractive to agents to drop other similar objects nearby.

However, stigmergy is not the only way social insects interact with each other. In most species, trophallaxis or liquid food exchange among members of the same colony, plays a key role in their social organization. Consider the case of some termite species which require intestinal protozoa to derive benefits from cellulose. Their early instar nymphs are fed either by oral or anal trophallaxis. The latter infects them with symbiotic protozoa or bacteria contained in the proctodeal liquid. The subsocial association result of this codependence have evolved into a complex social and morphological structure.

Inspired by the trophallaxis phenomenon observed in some ant and termite species, two different communication strategies among agents in ant-based clustering algorithms are investigated: (i) direct and (ii) indirect communication. The impact on the final clustering quality is evaluated by comparing the development of the clustering process generated by each strategy. It is shown that benefits on the final clustering are directly related to the usefulness of the exchanged information, its use, and on the number of participating agents.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Collective complexity out of numerous interactions among simple individuals is the way we can explain how an eusocial insects colony works. To be considered eusocial, insect species must have overlapping generations of parents and offspring living together in an organizational unit or colony, division of labor occurs within the colony, and when they develop a physical structure or nest to live in. Ants, termites and some bee and wasp species are considered eusocial [8]. At a "macroscopic" level the colony seems to be a living organism by itself; nevertheless, it is made of thousands or millions of autonomous creatures interacting with each other with no central entity controlling their organization. In other words, they *self-organize* [16].

Bonabeau et al. [4] define self-organization as "a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components" (p. 9). They also identify four basic elements of self-organization, which are positive feedback, negative feedback, the amplification of fluctuations, and the existence of multiple interactions among the constituents of a system. Self-organization is, in other words, the emergence of spatial and temporal patterns from local interactions among relatively simple entities in an initially homogeneous environment. Termite mounds are good examples of what can be achieved via self-organization. An external view of a termite mound of genera *Macrotermes* is shown in figure 1.1.

Termite mounds are thousands of times bigger than the size of a single termite. Therefore, they must work together, even across various generations, to complete such an astonishing structure. But, how do they coordinate the construction of nests without blueprints or a hierarchical work organization? They use *stigmergy*, an indirect nonintentional communication mechanism through local modifications of the environment.

In eusocial insects different activities are performed simultaneously by specialized individuals. This division of labor can be seen in activities such as nest cleaning and brood care. An example of such behavior can be seen in some ant species such as *Pogonomyrmex badius* and *Lasius niger* whose workers pile corpses of dead nestmates and defeated enemies on a distant spot away from their nest entrance [25, 4]. Another example is brood care, which is a highly specialized activity that require, in some cases, the separation of larvae according to their development stage. In experiments, the ant *Leptothorax unifasciatus* gather its larvae

Figure 1.1: External view of a termite mound of genera *Macrotermes*

together according to their size. Small larvae are aggregated near the center of a plate and larger larvae near the periphery of it [4].

Collective behaviors of social insects have inspired the design of several computational techniques for solving different kinds of problems. The idea is to use the same principles of self-organization that exist in social insects to coordinate populations of artificial agents that collaborate to solve computational problems [7]. Different behaviors of social insect colonies have inspired the design of heuristic algorithms to solve certain problems. For example, brood sorting and cemetery organization in some ant colonies have inspired the design of data clustering algorithms. The relatively new discipline of Swarm Intelligence studies these algorithms and their applications. Swarm Intelligence can be situated somewhere between multiagent systems and evolutionary computing.

Swarm systems were first studied in robotics [3]. In multiagent systems, *swarms* are composed of a multitude of simple agents with no central control over the their behavior, instead, they self-organize. Swarm systems offer some advantages over their deliberative counterparts, among others: robustness, flexibility, scalability and adaptability [21]. One key factor that makes swarm systems appealing is that the system's entities are *simple*. In this context, simple means that an agent's behavior is not based on explicit symbolic representations of its environment or other agents.

In "ant algorithms", as they have been labeled, agents coordinate their activities without direct communication. Instead, they use stigmergy. In classic ant-based clustering algorithms there is no direct interactions among agents; however, if we look again to the inspiration source, we will see that direct interaction *does* happen among insects. For example, in most termite species, trophallaxis or liquid food exchange among members of the same colony, plays a key role in their social organization [25]. Consider the case of some termite species which require intestinal protozoa to derive benefits from cellulose. Their early instar nymphs are fed either by oral or anal trophallaxis. The latter infects them with symbiotic

protozoa or bacteria contained in the proctodeal liquid. The subsocial association result of this codependence have evolved into a complex social and morphological structure [8].

This thesis will focus on ant-based clustering algorithms, and inspired by trophallaxis, the main hypothesis is that if we allow agents to exchange local information about the state of the environment, the clustering performance can be improved both in speed and quality.

1.1 Problem definition

The problem to be solved falls within the study area of Swarm Intelligence. According to Bonabeau et al. [4], Swarm Intelligence is "any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies" (p. 7).

To understand how algorithms have been designed using as inspiration the collective behavior of social insects colonies, let us consider the foundation of a termite nest. A termite nest is founded when a winged female lands on soil and starts expelling pheromones to attract males. The new couple make a hole in a tree or a subterranean camera in which the first eggs are laid [24]. As the colony grows, termites start building pillars and tunnels through the accumulation of soil pellets.

The inspiring behavior for clustering through artificial termites is the pillar building behavior. The insects carry about and put down pellets in an apparently random fashion after an exploration phase in which they move without taking any action. At this stage, a pellet just put down by a termite worker is often picked up and placed somewhere else by another worker. When a pellet is placed on top of another, the resultant structure appears to be much more attractive and termites soon start piling more pellets nearby, making the dropping spot even more attractive [25].

In the clustering algorithm inspired by this behavior, there is an artificial environment, usually a toroidal square grid, with randomly located data elements[1] and a number of unloaded agents moving randomly over it. Each agent picks up or drops a datum with a probability that depends on the model being used. All models are based on the idea that a sufficiently compact data group is more attractive for agents to drop data elements on it than to pick elements from it. At the same time, isolated elements are easily picked up. Most algorithms assume that two or more agents can be in the same location simultaneously and that they do not interact while being there. The environment is the only mean of interaction among agents.

The assumptions just mentioned artificially impose restrictions on the agents' behavior that need not be there. One of them is that agents are "blind", i.e., they move randomly over the grid and may waste too much time searching in unpopulated areas affecting the convergence speed of the algorithm. Another restriction is that agents may miss the point

[1] I also use the terms "data objects", "data elements", or simply "objects" throughout this thesis, to refer to elements belonging to a patterns database.

of their task, i.e., they may drop objects in areas populated by half-way similar objects, affecting the final clustering quality.

The problem of convergence speed is mentioned and studied in references [12, 16, 4], but the issue of communication among agents in ant-based clustering algorithms has not been studied before. In this thesis we propose to explore the effects of letting agents communicate in ant-based clustering algorithms. The rationale is: if agents exchange information about data distribution on the grid, they no longer are blind[2] and therefore the convergence time is reduced and the final clustering quality improved. In experiments, the performance effects of this modification will be measured and compared with the same algorithm without communication.

1.2 Research Objectives

The main research objective can be accomplished by subdividing it into two particular objectives. These are specified in the following subsections.

1.2.1 Main objective

The main objective of this thesis is:

To determine the performance effects of direct and intentional indirect (in contrast to just the stigmergic one) communication among agents in ant-based clustering algorithms. The content of the communication is local knowledge about the spatial distribution of data elements and their data types in the environment. The ultimate goal for information exchange is to let agents choose the best location on the environment on which to drop a data object, and therefore create better clusters faster than with noncommunicating agents.

1.2.2 Particular objectives

To satisfy the objective stated above, the effect in convergence time and clustering quality of the following communication strategies will be determined:

- Direct communication. The effects of direct information exchange among agents will be determined. The communication will occur whenever two or more agents simultaneously meet at a point on the grid that serves as environment.
- Indirect communication. Agents will communicate indirectly through information packets dropped on the environment. This is intentional communication, not just stigmergic.

[2]However, they do not get full observability of the environment since agents exchange local information

1.3 Hypothesis

The hypothesis of this thesis is:

In ant-based clustering algorithms, it is possible to reduce convergence time and improve clustering quality by letting agents exchange information about the spatial distribution of data and their type.

1.3.1 Research questions

The questions this thesis seek to answer are:

1. What are the performance effects, if any, of direct and indirect communication among agents in ant-based clustering algorithms?
2. How much knowledge is it possible to exchange without negatively affect the performance of ant-based clustering algorithms?
3. What kind of knowledge is it possible to exchange?
4. What are the advantages of direct and indirect communication among agents in ant-based clustering algorithms?

1.4 Thesis organization

The rest of this thesis is organized as follows. Chapter 2 presents background knowledge about the application problem, that is, data clustering. Chapter 3 presents background knowledge about ant-based clustering algorithms. Chapter 4 presents our experimental design and results using direct communication among agents. In chapter 5, we present the experimental design and results using intentional indirect communication among agents. In chapter 6, obtained results are discussed and the final conclusions drawn from them are presented.

Chapter 2

Data Clustering

In this chapter, basic definitions of the problem of data clustering are given. The problem of data representation and the notion of similarity between multivariate data are discussed. Finally, traditional approaches to clustering are described.

2.1 Clustering Problem

There are two types of data analysis, *exploratory* and *confirmatory* [18]. In exploratory data analysis the researcher wants to discover the relationships among data elements in a data set. In confirmatory studies, the researcher wants to see how well a model fits the data. We can locate cluster analysis into the first category. Cluster analysis helps us discover natural groups within data.

The goal of cluster analysis is to partition a set of data or objects into *clusters* (groups, subsets, classes) so that elements belonging to the same cluster are as similar as possible and elements that belong to different clusters are as dissimilar as possible [17].

2.1.1 Definition

Hoppner et al. [17] define the analysis space (the space of possible solutions to a clustering problem) as

Let $D \neq \phi$ be a set and R a set of sets with $(|R| \geq 2) \vee (\exists r \in R : |r| \geq 2)$. We call D a data space and R a result space. Then, $A(D,R) := \{f | f : X \rightarrow K, X \subseteq D, X \neq \phi, K \in R\}$ is called an analysis space. A mapping $f : X \rightarrow K \in A(D,R)$ represents the result of a data analysis by the mapping of a special, given set of data $X \subseteq D$ to a possible result $K \in R$.

In order to establish a criterion to select among all the possible solutions, they also define an objective function as

Let $A(D,R)$ be an analysis space. Then, a mapping $J : A(D,R) \rightarrow \Re$ is called an objective function of the analysis space.

The value $J(f)$ can also be considered a quality measure that can be used to compare

different solutions for the same problem.

A cluster partition is defined as

Let $A(D, R)$ be an analysis space, $X \subseteq D, f : X \to K \in A(D, R), A_k := f^{-1}(k)$ for $k \in K$. Then, f is called a cluster partition if $\{A_k | k \in K\}$ is a (hard) partition of X, that is,

$\bigcup_{i \in K} A_k = X$

$\forall i, j \in K : i \neq j \Rightarrow A_i \cap A_j = \phi$

$\forall i \in K : \phi \neq A_i \neq X$

Although Hoppner et al. [17] do not define the clustering problem explicitly, we can use the concepts defined above to state it as

Let C be a set of partitions of $X \subseteq D$, with $C \in A(D, R)$. The clustering problem consists in finding a $c_{opt} \in C$ such that $J(c_{opt}) = \max\{J(c) | c \in C\}$

2.2 Components of a Clustering Task

According to Andritsos [2] and Jain et al. [18], the components of a clustering task are:

1. Data Collection. This subtask includes the selection of relevant data objects from among different sources. The data objects select must provide information relevant to the clustering task. It is goal-driven.

2. Initial screening. This subtask includes the transformation of data in order to be properly used by the representation scheme and by the clustering method selected.

3. Pattern Representation. This subtask is composed of feature extraction/selection and selects a suitable form for the data to be processed by the clustering algorithm.

4. Definition of a similarity/dissimilarity measure. Depending on the data representation, the similarity or dissimilarity measure is selected. In this stage, one can incorporate background knowledge.

5. Application of a clustering method. Includes the selection of proper parameters and strategy for the clustering algorithm.

6. Assessment of output. There are three types of validation studies. An *external* assessment compares the output of a clustering algorithm to an *a priori* partition. An *internal* validation tries to determine if the recovered structure is appropriate for the data. A *relative* test compares two structures and measures their relative quality.

7. Data abstraction. This stage includes the combination of clustering results with other studies to draw conclusions or decisions.

2.3 Data Representation

Data objects can be represented through their characteristics or attributes. Depending on the characteristic they are describing, attributes can be of different data types. In [2] it can be found a categorization of the different types of variables that model data objects attributes. The classification is based on two schemes: the *Domain Size* and the *Measurement Scale*.

2.3.1 Classification Based on Domain Size

This classification distinguishes data objects based on the size of their domain, i.e., the number of distinct values the data objects may assume. According to [2] the classes available are:

1. An attribute is *continuous* if its domain is *uncountably infinite*, i.e., if its elements cannot be associated one-to-one with the set of positive integers.

2. An attribute is *discrete* if its domain is a *finite set*, i.e., if its elements can be put into a one-to-one correspondence with a finite subset of the positive integers. *Binary* attributes are a particular case of discrete attributes that have only two discrete values in their domain.

2.3.2 Classification Based on the Measurement Scale

This classification distinguishes attributes according to their measurement scales. According to [2] the classes available are:

1. A *nominal scale* distinguishes among categories. They are a generalization of binary attributes with a domain of more than two discrete values.

2. An *ordinal scale* involves a nominal scale with the additional feature that their values can be totally ordered.

3. An *interval scale* measures values in a linear scale. This allows to not only totally ordered values, but it can also establish distances among these values.

4. A *ratio scale* is an interval scale with a meaningful zero point.

Nominal and ordinal scaled attributes are called *qualitative* or *categorical* attributes. Interval and ratio scaled are called *quantitative*.

2.4 Proximity Measures

A notion of central importance to clustering is that of *proximity*. There are two types of proximity measures, *similarity* and *dissimilarity* measures. Possible definitions for these measures depend on the attributes data types of the data objects to be clustered. The discussion of proximity measures is based on the work of Everitt et al. [9].

2.4.1 Standardization

Standardization is a transformation on data so that the effects on proximity measures of different measure scales on attributes is eliminated. Attributes of a data object may have different measurement units. For example, one can measure time in years, days, hours, seconds or milliseconds. The selection of units can have a tremendous influence on proximity measures by making attributes with large standard deviations weight more and thus, biasing the result. To compensate this effect one can *standardize* the data set.

Standardizing functions

If the collection of data objects is arranged in a $m \times n$ data matrix with data objects subscripted $i = 1, 2, 3, \ldots, m$ and attributes subscripted $j = 1, 2, 3, \ldots, n$, then the matrix can be standardized using

$$Z_{ij} = \frac{X_{ij} - \overline{X}_j}{S_j} \tag{2.1}$$

where

$$\overline{X}_j = \frac{1}{m} \sum_{i=1}^{m} X_{ij}$$

and

$$S_j = \left(\frac{1}{m-1} \sum_{i=1}^{m} (X_{ij} - \overline{X}_j)^2 \right)^{1/2}$$

It is important to notice that standardized values Z_{ij} are unitless and that $\overline{Z}_j = 0$ and the standard deviation $S'_j = 1$. Because of this, all values Z_{ij} are likely to be in the range of $-2.0 \leq Z_{ij} \leq 2.0$.

There are other standardizing functions that transform entries in the unstandardized data matrix into values in the range $0 \leq Z_{ij} \leq 1$. To use these functions, all the values X_{ij} must be nonnegative and no column can be all zeros.

The first of these functions is

$$Z_{ij} = \frac{X_{ij}}{\mathrm{MAX}_j} \tag{2.2}$$

where MAX_j is the maximum value in the jth column of the data matrix. As a result, every column will have at least one element $Z_{ij} = 1$ which will occur when $X_{ij} = \mathrm{MAX}_j$.

The second standardizing function is

$$Z_{ij} = \frac{X_{ij} - \text{MIN}_j}{\text{MAX}_j - \text{MIN}_j} \tag{2.3}$$

where MAX_j is the maximum value in the jth column of the data matrix as before, and MIN_j is the minimum value in the jth column. In this case, every column will have at least one value $Z_ij = 0$ that will occur when $X_{ij} = \text{MIN}_j$ and at least one value $Z_{ij} = 1$ which will occur when $X_{ij} = \text{MAX}_j$.

The third function normalizes the standardized values in the jth column so that they sum 1. This function is

$$Z_{ij} = \frac{X_{ij}}{\sum_{i=1}^{m} X_{ij}} \tag{2.4}$$

2.4.2 Proximity Measures for Quantitative Data

Proximity measures for quantitative data can be differentiated by their sensitiveness to certain data transformations. Such transformations are:

Additive translation. For a given data profile $\mathbf{p}$ in a data set, another data profile $\mathbf{p}'$ is defined by $\mathbf{p}' = \mathbf{p} + \mathbf{c}$ where $\mathbf{c}$ is a constant vector.

Proportional translation. For a given data profile $\mathbf{p}$ in a data set, another data profile $\mathbf{p}'$ is defined by $\mathbf{p}' = C\mathbf{p}$ where C is a constant.

Some proximity measures are insensitive to additive translations, others to proportional translations and some others to both. The following subsections describe in detail some proximity measures and catalog them according to their sensitiveness to translations.

Euclidean Distance

The Euclidean distance is by far the most commonly used proximity measure. The Euclidean distance is a *dissimilarity* measure, i.e., the greater the distance, the more different the objects are. It is defined by

$$e_{ik} = \left(\sum_{j=1}^{n} (X_{ij} - X_{kj})^2 \right)^{1/2} \tag{2.5}$$

where X_{ij} is the jth attribute of the ith object in the data matrix. The Euclidean distance can also be represented in vectorial form

$$e_{ik} = \| \mathbf{X}_i - \mathbf{X}_k \| \tag{2.6}$$

the range of e_{ik} is $0 \leq e_{ik} \leq \infty$.

Average Euclidean Distance

The average Euclidean distance is defined by

$$d_{ik} = \left(\frac{\sum_{j=1}^{n} (X_{ij} - X_{kj})^2}{n} \right)^{1/2} \tag{2.7}$$

It is defined in the range $0 \leq d_{ik} \leq \infty$. From equations 2.5 and 2.7 it can be seen that

$$d_{ik} = \frac{e_{ik}}{\sqrt{n}} \tag{2.8}$$

The main difference between the classical Euclidean distance and the average Euclidean distance is not the resultant value after their computation, but the possibility of using the average Euclidean distance when there are missing values in the data matrix.

Shape Difference Coefficient

This dissimilarity coefficient is defined in the range $0 \leq z_{ik} \leq \infty$, and it is a function of the average Euclidean distance d_{ik}. It can be obtained using

$$d_{ik} = \left(\frac{n}{n-1} (d_{ik}^2 - q_{ik}^2) \right)^{1/2} \tag{2.9}$$

where

$$q_{ik} = \frac{1}{n^2} \left(\sum_{j=1}^{n} (X_{ij} - X_{kj}) \right)^2$$

The shape difference coefficient is insensitive to additive translations.

Cosine Coefficient

This is a similarity coefficient defined in the range $-1 \leq c_{ik} \leq 1$. The value 1 indicates maximum similarity, but not necessarily identity. It is defined by

$$c_{ik} = \frac{\sum_{j=1}^{n} X_{ij} X_{kj}}{\left(\sum_{j=1}^{n} X_{ij}^2 \right)^{1/2} \left(\sum_{j=1}^{n} X_{kj}^2 \right)^{1/2}} \tag{2.10}$$

or in vectorial form

$$c_{ik} = \frac{\mathbf{X}_i \cdot \mathbf{X}_k}{\| \mathbf{X}_i \| \| \mathbf{X}_k \|} \tag{2.11}$$

The cosine coefficient is insensitive to proportional translations.

Correlation Coefficient

The correlation coefficient r_{ik} is defined by

$$r_{ik} = \frac{\sum_{j=1}^{n} X_{ij}X_{kj} - (\frac{1}{n})(\sum_{j=1}^{n} X_{ij})(\sum_{j=1}^{n} X_{kj})}{\left\{\left[\sum_{j=1}^{n} X_{ij}^2 - \frac{1}{n}(\sum_{j=1}^{n} X_{ij})^2\right]\left[\sum_{j=1}^{n} X_{kj}^2 - \frac{1}{n}(\sum_{j=1}^{n} X_{kj})^2\right]\right\}^{1/2}} \tag{2.12}$$

It is defined in the range $-1 \leq r_{ik} \leq 1$. The value $r_{ik} = 1$ indicates maximum similarity but not necessarily identity, and $r_{ik} = -1$ indicates minimum similarity. It is insensitive to both, additive and proportional translations.

Canberra Metric Coefficient

This dissimilarity coefficient is defined in the range $0 \leq a_{ik} \leq 1$, where a value of 0 indicates maximum similarity, which occurs when the ith object is identical to the kth object. It is obtained using

$$a_{ik} = \frac{1}{n}\sum_{j=1}^{n} \frac{|X_{ij} - X_{kj}|}{X_{ij} + X_{kj}} \tag{2.13}$$

Each term is scaled between 0 and 1, equalizing the contribution of each attribute to overall similarity.

Bray-Curtis Coefficient

This dissimilarity coefficient is defined in the range $0 \leq b_{ik} \leq 1$, where a value of 0 indicates maximum similarity, which occurs when objects i and k are identical. It is obtained using

$$b_{ik} = \frac{\sum_{j=1}^{n} |X_{ij} - X_{kj}|}{\sum_{j=1}^{n} (X_{ij} + X_{kj})} \tag{2.14}$$

For both these coefficients, a_{ik} and b_{ik}, the values in the data matrix must be greater than or equal to zero. Therefore, these coefficients cannot be used when the data matrix is standardized.

Table 2.1 summaries the sensitiveness of the previously described proximity coefficients to additive and proportional translations.

2.4.3 Proximity Measures for Qualitative Data

Similarity measures are common when dealing with data objects with categorical or qualitative attributes. Two individuals i and j have a similarity coefficient s_{ij} of unity

Table 2.1: Sensitiveness of proximity measures to additive and proportional translations

Coefficient	Range	Ignores	
		Additive translations	Proportional translations
Dissimilarity			
e_{ik}	$0 \leq e_{ik} \leq \infty$	No	No
d_{ik}	$0 \leq d_{ik} \leq \infty$	No	No
a_{ik}	$0 \leq a_{ik} \leq 1$	No	No
b_{ik}	$0 \leq b_{ik} \leq 1$	No	No
z_{ik}	$0 \leq z_{ik} \leq \infty$	Yes	No
Similarity			
c_{ik}	$-1 \leq c_{ik} \leq 1$	No	Yes
r_{ik}	$-1 \leq r_{ik} \leq 1$	Yes	Yes

if both have identical values for all variables. A similarity of zero indicates that the two objects differ maximally for all variables. Similarity is often limited to the range $[0-1]$ with 1 indicating maximum similarity.

Similarity Measures for Binary Data

As mentioned before binary attributes are a special case of categorical data. There are many similarity measures for binary data, but all of them are defined in terms of the entries of a cross-classification of the counts of matches and mismatches in the p variables of two data objects. Table 2.2 shows the general version of this cross-classification.

Table 2.2: Counts of binary outcomes for two individuals

		Individual i		
	Outcome	1	0	Total
Individual j	1	a	b	$a+b$
	0	c	d	$c+d$
	Total	$a+c$	$b+d$	$p=a+b+c+d$

Table 2.3: Similarity measures for binary data

Measure	Formula
Matching coefficient	$s_{ij} = \frac{a+d}{a+b+c+d}$
Jaccard coefficient	$s_{ij} = \frac{a}{a+b+c}$
Rogers and Tanimoto	$s_{ij} = \frac{a+d}{a+2(b+c)+d}$
Sokal and Sneath	$s_{ij} = \frac{a}{a+2(b+c)}$
Gower and Legendre	$s_{ij} = \frac{a+d}{a+\frac{1}{2}(b+c)+d}$
Gower and Legendre	$s_{ij} = \frac{a}{a+\frac{1}{2}(b+c)}$

A list of some of the similarity measures often used for binary data is shown in Table 2.3. The sharp-eyed reader should notice that these coefficients differ in how they handle zero-zero matches and on different weightings for mismatches. The reason for these differences lies on the uncertainty associated with term d. For example, consider the case of an attribute encoding gender. There is no preference as to which of the two categories should be coded zero or one. In this case both occurrences should be considered. This is not always the case, however. Imagine we are comparing two attributes encoding some feature of an animal. The sole absence of a characteristic does not always indicates that two animals are as similar as if they both would have it.

Similarity Measures for Multilevel Qualitative Data

When qualitative data have more than two levels, representing them as a collection of binary-valued attributes will produce a large number of "negative" matches. As was mentioned above, this causes trouble when these matches are not meaningful.

An alternative method of measuring proximity between this kind of data is to allocate a score s_{ijk} of zero or one to each variable k, depending on whether two individuals i and j are the same on that variable. The proximity coefficient is then obtained using

$$s_{ij} = \frac{1}{p} \sum_{k=1}^{p} s_{ijk} \tag{2.15}$$

2.4.4 Proximity Measures for Ordinal-Scaled Data

Kendall's tau Coefficient

This measure is a measure of correlations between two ordinal-level variables. For a sample of n observations, there are $n(n-1)/2$ possible comparison points [23]. So Kendall's

tau, τ_{ik} is defined as

$$\tau_{ik} = \frac{C - D}{\binom{n}{2}} \tag{2.16}$$

where C is the number of pairs that are concordant and D is the number of pairs that are discordant. Kendall's tau is defined in the range $-1 \leq \tau_{ik} \leq 1$.

2.4.5 Proximity Measures for Mixtures of Qualitative and Quantitative Data

There are several strategies to deal with mixtures of qualitative and quantitative data. Among others, one can dichotomize all variables and use a similarity measure for binary data; another possibility would be to construct a dissimilarity measure for each type of variable and combine these, etc. Gower, in 1971, proposed a general similarity measure given by

$$s_{ij} = \frac{\sum_{k=1}^{p} w_{ijk}\, s_{ijk}}{\sum_{k=1}^{p} w_{ijk}} \tag{2.17}$$

where s_{ij} is the similarity between the ith and jth element as measured by the kth variable, and w_{ijk} is typically one or zero depending on whether the comparison is considered valid.

2.5 Clustering Techniques

2.5.1 Hierarchical Methods

Hierarchical clustering methods do not partition a data set into clusters in a single step, rather, they iteratively divide or join existing clusters to form a hierarchy of partial clusterings. Because it is possible to divide or join groups, hierarchical methods can be classified into agglomerative or divisive methods if they join or divide partial clusterings respectively.

Due to the fact that hierarchical methods create partial clusterings all the way from clusters containing only one element to a single big cluster containing all elements, one has to decide when to stop the algorithm either by guessing the optimal number of clusters or by establishing a distance threshold. In general, determining the optimal number of clusters is a difficult problem on its own.

Agglomerative methods

Agglomerative methods start with n single-member clusters and fuse them iteratively until a single cluster with n elements is created. The basic criterion used to fuse clusters

is their proximity in attribute space; therefore, depending on how we measure proximity between two elements, one element and a cluster, and between two clusters, is the final clustering hierarchy. In the following subsections, we present the most commonly used agglomerative techniques used. Figure 2.1 shows an example of a hierarchical agglomerative clustering process.

Single linkage clustering method

The defining feature of the single linkage method is that the distance between groups is defined as that of the closest pair of individuals whose members are in different groups. With the single linkage method, we need to recompute the proximity matrix after each clustering step.

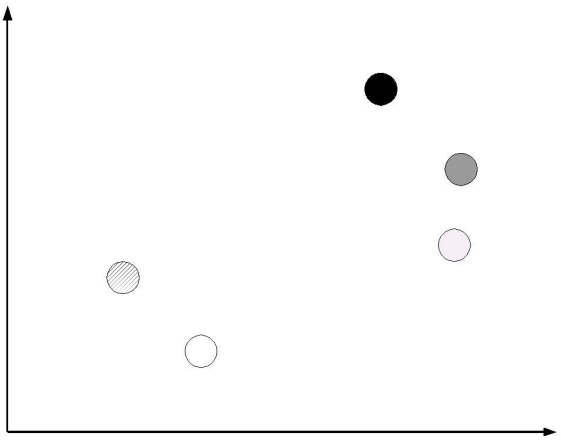
Figure 2.2 shows the merge criterion used by the single linkage method. The distance measure used in this example is the Euclidean distance.

Complete linkage clustering method

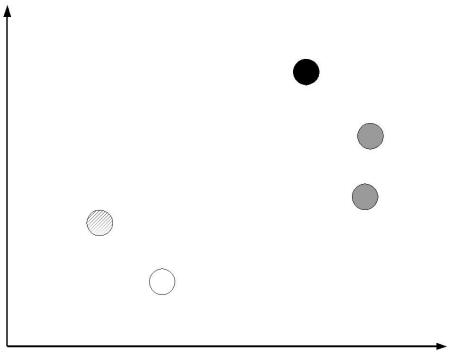
The complete linkage clustering method is very much like the single linkage method but instead of using as merging criterion the closest pair of individuals from different groups, this method takes the furthest. Figure 2.3 shows the merge criterion used by the complete linkage method. The distance measure used in this example is the Euclidean distance.

Average linkage clustering method

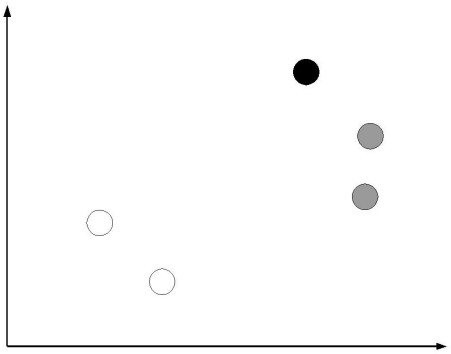
The average linkage clustering method is in between the single linkage method and complete linkage methods. In this method, the distance between two clusters is the average of the distance between all pairs of individuals that are made up of one individual from each group. Figure 2.4 shows the merge criterion used by the average linkage method. The distance measure used in this example is the Euclidean distance.



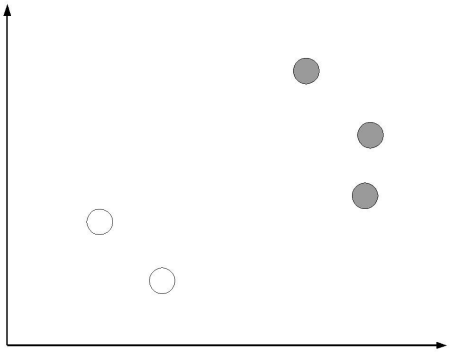
(a) Initial clustering



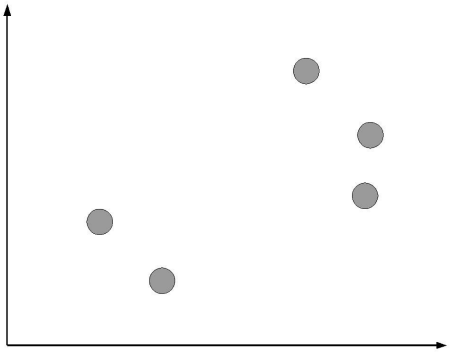
(b) First iteration



(c) Second iteration



(d) Third iteration



(e) Last iteration

Figure 2.1: Example of an agglomerative clustering on a data collection.

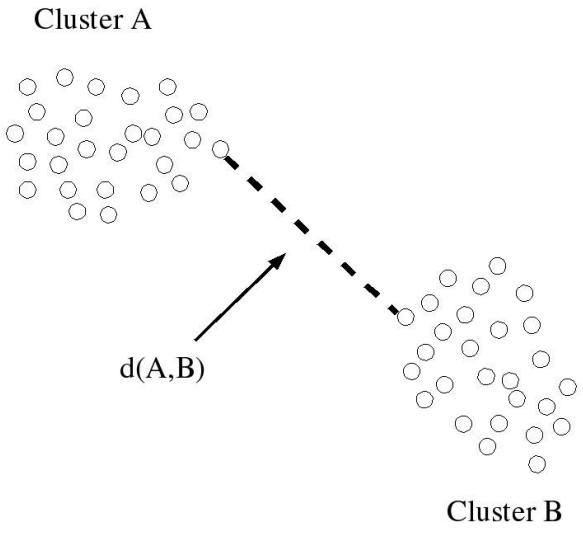


Figure 2.2: Inter-cluster distance measure for the single linkage method.

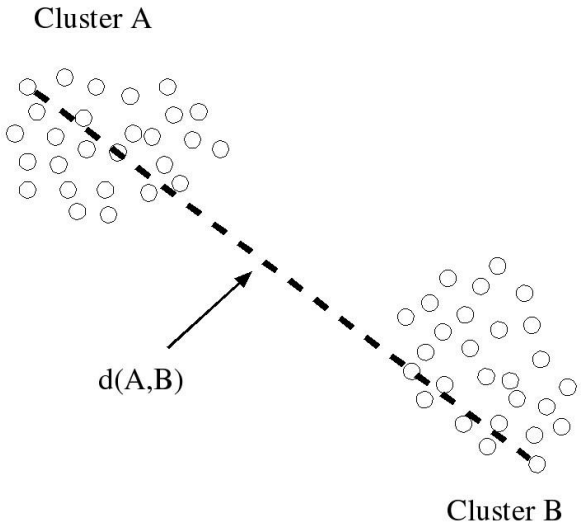


Figure 2.3: Inter-cluster distance measure for the complete linkage method.

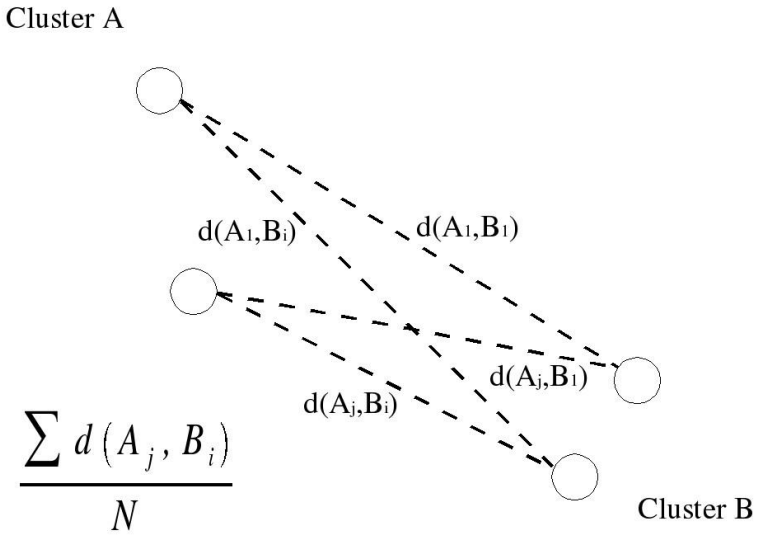


Figure 2.4: Inter-cluster distance measure for the average linkage method.

Ward's minimum variance clustering method

In this method, the criterion to decide which clusters to merge is that of minimum intra-cluster variance increase. The method tries to minimize

$$E = \sum_{m=1}^{g} \sum_{j=1}^{p} \parallel \mathbf{e}_j - \mu_m \parallel^2 \tag{2.18}$$

where μ_m is the centroid of cluster $m = 1, 2, \ldots, g$, $\mathbf{e}_j$ is the element $j = 1, 2, \ldots, p$ of the cluster m and g is the total number of clusters at the moment of joining.

The Ward's method does not guarantee an optimal partitioning of objects into clusters. This is because of the greedy strategy used by the method and because the algorithm does not explore all possible partitions.

Weighted pair-group method using arithmetic averages

This method requires taking a weighted average of the resemblance coefficients when recomputing the proximity matrix. Take for example the case when the proximity coefficient, say R_{ij}, is used to measure the proximity between point i and point j. In that case, and using the average linkage method, the proximity value for clusters (124) and (35) would be

$$R_{(124)(35)} = \left(\frac{1}{6}\right) R_{13} + \left(\frac{1}{6}\right) R_{15} + \left(\frac{1}{6}\right) R_{23} + \left(\frac{1}{6}\right) R_{25} + \left(\frac{1}{6}\right) R_{43} + \left(\frac{1}{6}\right) R_{45}$$

In computing the average, each term is given the same weight. Using the weighted pair-group method, the proximity value would be

$$R_{(124)(35)} = w_{13}R_{13} + w_{15}R_{15} + w_{23}R_{23} + w_{25}R_{25} + w_{43}R_{43} + w_{45}R_{45}$$

that is, every proximity coefficient is weighted differently.

Centroid method

In this method the similarity between two clusters is equal to the similarity between their centroids. This method has the problem of *reversals*, that is, the distance between two clusters after a merge decreases, instead of increasing. This gives some problems at the interpretation stage.

Flexible method

The Lance and Williams recurrence formula gives the distance between a group k and a group ij formed by the fusion of two groups (i and j) as

$$d_{k(ij)} = \alpha_i d_{ki} + \alpha_j d_{kj} + \beta d_{ij} + \gamma |d_{ki} - d_{kj}| \tag{2.19}$$

where d_{ij} is the distance between groups i and j.

By choosing certain values for the parameters of equation2.19, a completely different clustering method can be applied. For example, if one choose $\alpha_i = \alpha_j = 1/2$ and $\beta = 0$ and $\gamma = -1/2$, the distance between groups is

$$d_{k(ij)} = \frac{1}{2}d_{ki} + \frac{1}{2}d_{kj} - \frac{1}{2}|d_{ki} - d_{kj}|$$

if $d_{ki} > d_{kj}$ then $|d_{ki} - dkj| = d_{ki} - dkj$ and $d_{k(ij)} = d_{kj}$. Similarly, if $d_{ki} < d_{kj}$ then $|d_{ki} - dkj| = d_{kj} - dki$ and $d_{k(ij)} = d_{ki}$. Therefore, the recurrence formula gives the required $d_{k(ij)} = min\{d_{ki}, d_{kj}\}$ for the single linkage method.

2.5.2 Divisive Methods

There are basically two types of divisive methods: *monothetic* and *polythetic* methods. Monothetic methods use a single variable on which to base the split at a given stage, whereas polythetic methods consider all variables at each stage.

Monothetic divisive methods

If the data to be clustered consists of p binary variables, the division of elements can be done using an optimization criterion reflecting either cluster homogeneity or association

with other variables. A possible homogeneity criterion is the information content C, defined by p variables and n objects:

$$C = p\,nlogn - \sum_{k=1}^{p}[f_k log f_k - (n - f_k)log(n - f_k)] \tag{2.20}$$

where f_k is the number of individuals having the kth attribute. If a group X is to be split into two groups A and B, the reduction in C is $C_X - C_A - C_B$. Therefore, clusters are split at each stage according to possession of the attribute which leads to the greatest reduction in C.

Polythetic divisive methods

The MacNaughton-Smith et al. method starts by finding the object that is farthest away from the others in a group, and using it as a seed for a splinter group. Each object is considered for entry to the splinter group, the closest object is moved into it. This process is repeated with the largest cluster in diameter.

2.5.3 Optimization Methods

Optimization clustering techniques produce a partition of data according to some optimization criterion. Differences between optimization methods arise because of the different optimization criteria and algorithms that might be used.

A basic notion on which the following discussion is based is that every partition of a collection of n individuals into g groups has an associated index $c(n, g)$ that measures the quality of that partition.

Clustering criteria derived from the dissimilarity matrix

These criteria are based on two concepts: *homogeneity* and *separation*. Using these criteria, we look for compact and well-separated clusters.

Table 2.4 lists some of the most widely used criteria in optimization clustering algorithms.

In general, an optimization-based clustering algorithm aims at maximizing the homogeneity (or minimizing heterogeneity) and maximizing separation among groups. To accomplish this task and after having chosen an index measuring the heterogeneity or separation of a group, clustering criteria must be defined by a suitable aggregation over groups. Such aggregations could be

$$c_1(n, g) = \sum_{m=1}^{g} h(m) \tag{2.21}$$

or

$$c_2(n, g) = \max_{m=1,\ldots,g} [h(m)] \tag{2.22}$$

Table 2.4: Quality measures of the mth group containing n_m data. These measures are based on a dissimilarity matrix Δ with elements $\delta_{ql,kv}$ measuring the dissimilarity between the lth object in the qth group and the vth element of the kth group

Measure	Index ($r \in \{1, 2\}$)
Heterogeneity	$h_1(m) = \sum_{l=1}^{n_m} \sum_{v=1, v\neq l}^{n_m} (\delta_{ml,mv})^r$
Heterogeneity	$h_2(m) = \max_{\substack{l=1,\ldots,n_m \\ v=1,\ldots,n_m \\ v\neq l}} [(\delta_{ml,mv})^r]$
Heterogeneity	$h_3(m) = \min_{v=1,\ldots,n_m} \left[\sum_{l=1}^{n_m} (\delta_{ml,mv})^r\right]$
Separation	$i_1(m) = \sum_{l=1}^{n_m} \sum_{k\neq m} \sum_{v=1}^{n_k} (\delta_{ml,kv})^r$
Separation	$i_2(m) = \min_{\substack{l=1,\ldots,n_m \\ k\neq m \\ v=1,\ldots,n_k}} [(\delta_{ml,kv})^r]$

or

$$c_2(n, g) = \min_{m=1,\ldots,g} [h(m)] \tag{2.23}$$

Cluster criteria can also be defined in terms of homogeneity and separation.

Clustering criteria derived from continuous data

A commonly used clustering criterion derived from continuous data is the minimization of the sum of squared Euclidean distances between individuals and their group mean, that is,

$$\sum_{m=1}^{g} \sum_{l=1}^{n_m} d(ml, m)^2 \tag{2.24}$$

where $d(ml, m)$ is the Euclidean distance between the lth individual in the mth group and the mean of the mth group.

This criterion has some serious problems: the implicit spherical shape that assumes on the part of the clusters and its scale dependency.

There are other clustering criteria that overcome the shape problem, one of them is:

$$\prod_{m=1}^{g} [\det(\mathbf{W}_m)]^{n_m} \tag{2.25}$$

where $\mathbf{W}_m$ is the dispersion matrix within the mth group

$$\mathbf{W}_m = \sum_{l=1}^{n_m} (\mathbf{x}_{ml} - \bar{\mathbf{x}})(\mathbf{x}_{ml} - \bar{\mathbf{x}})' \tag{2.26}$$

and n_m is the number of individuals in the mth group.

The selection of a given optimization criteria depends on some background knowledge of the data to be clustered.

Optimization algorithms

The most basic straightforward optimization algorithm for cluster analysis is the one that calculates the value of the criterion for all possible partitions and choose the one that gives the optimal value for that criterion. In practice, however, this is not practical since the number of different partitions of n objects into g groups is given by

$$N(n, g) = \frac{1}{g!} \sum_{m=1}^{g} (-1)^{g-m} \binom{g}{m} m^n \tag{2.27}$$

For this reason, the *hill-climbing* algorithms family has been used to solve this problem. Essentially, these algorithms take the following steps when trying to optimize the clustering criterion:

1. Find some initial partition of the n objects into g groups.
2. Calculate the change in the clustering criterion value of changing each object from its group into another group.
3. Make the change that maximizes the improvement in the value of the criterion value.
4. Repeat from step 2 until no further improvement is achieved.

All optimization algorithms that are applied in practice vary in the way the find the initial partition. Of course, the final clustering depends very much on this initial partition. Therefore, optimization-based clustering algorithms are run several times with varying initial partitions in order to avoid local optima.

A good example of such algorithm is the well-known *k-means*, which iteratively updates a partition by simultaneously relocating each object to the group to whose mean it is closest and then recalculating the group means.

Chapter 3

Ant-Based Clustering Algorithms

In this chapter, ant-based clustering algorithms are described. The chapter begins with a review of the inspiring behaviors exhibited by ants and termites. It continues with the description of the first works on ant-based sorting and clustering. The chapter ends with the presentation of some recent improvements of ant-based clustering algorithms.

3.1 Cemetery Organization, Brood Sorting and Nest Building

An interesting collective behavior of several ant species is the creation of "cemeteries" or piles of corpses. When dead bodies of nestmates and defeated enemies are scattered around the nest entrance of some ant species such as *Pogonomyrmex badius* and *Lasius niger*, worker ants create piles of their corpses on a distant spot away from it [25, 4].

Although this behavior is not fully understood, computer scientists, with help from entomologists have designed simple models in which insects are simulated by computational agents that move randomly in their environment and pick up and drop objects that have proved successful for spatial sorting and data clustering. The basic idea in these models is that large clusters are attractive to ants to make them larger, while small clusters are attractive to make them smaller. The underlying mechanism by which this is possible is *stigmergy*, the indirect influence on the behavior of others through local environment modifications.

While studying termites of *Macrotermes bellicosus*, Pierre-Paul Grassé [11] observed that when workers of this species were placed in a container with some soil pellets, the insects carried about and put down pellets in an apparently random fashion after an exploration phase where they moved through the container without taking any action. At this stage, a pellet just put down by a termite worker is often picked up and placed somewhere else by another worker. When a pellet is placed on top of another, the resultant structure appears to be much more attractive and termites soon start piling more pellets nearby, making the dropping spot even more attractive [25]. To explain this behavior, he proposed that stigmergy, from the Greek phrase "incite to work", was a key factor in that process. In this case, stigmergy operates in the following way: If a loaded termite finds a heap of pellets, it

drops its load on top of it making the heap larger. The heap triggered the dropping behavior on the termite. The larger the heap, the higher the probability the termite drops a pellet on top of it. It is clearly a positive feedback process. On the other hand, if an unloaded termite finds an isolated pellet, it will pick it up almost deterministically; however, if the pellet is surrounded by other pellets, the probability of picking it up lowers. Therefore, a small heap or group of pellets is likely to disappear.

Another interesting behavior that goes beyond simple object piling is brood sorting. In the ant *Leptothorax unifasciatus*, workers gather larvae according to their development stage. In experiments, worker ants of this species put early larvae at the center of a plate and older larvae in the periphery [4]. This suggests that in brood sorting, objects are not just put together, as it happens with cemetery organization or nest building, but spatially sorted according to some of their distinctive characteristics. This fact inspired the basic model for spatial sorting using artificial ants. It is discussed in the next section.

3.2 Spatial Aggregation and Sorting

3.2.1 Spatial Aggregation

Deneubourg et al. [6] proposed a model for spatial aggregation based on the behaviors described above. A group of agents exhibiting the same behavior move randomly over a toroidal square grid. In the environment there are objects that were initially scattered in a random fashion. The objects can be picked up, moved or dropped in any free location on the grid. An object is picked up with high probability if it is not surrounded by other objects and is dropped by a loaded agent if its neighborhood is populated by other objects and the location of the agent has no object on it. The probability p_p with which an object is picked up is given by

$$p_p = \left(\frac{k_1}{k_1 + f}\right)^2 \tag{3.1}$$

where k_1 is a constant and f is considered to be a measure of object density in the neighborhood of the agent.

Figure 3.1 shows the behavior of the probability as a function of k_1 and f.

If the object is surrounded by other objects, then f should be large in comparison with k_1 in order to make p_p small. On the other hand, if the object is not surrounded by any other object, then f should be small making p_p high. The probability p_d for an agent to deposit an object is given by

$$p_d = \left(\frac{f}{f + k_2}\right)^2 \tag{3.2}$$

where k_2 is another constant. If the object is surrounded by other objects, then f should be large in comparison with k_2 in order to make p_d approach 1. On the other hand, if the

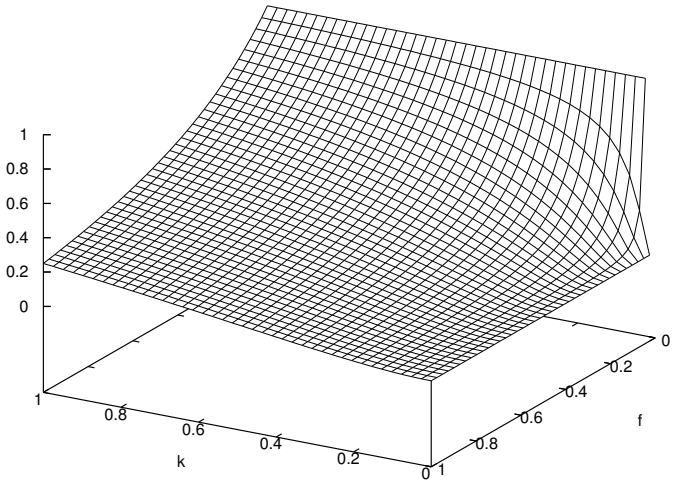


Figure 3.1: Probability of picking an object as a function of k_1 and f in Deneubourg's model.

object is not surrounded by any other object, then f should be small making p_d small. As expected, p_d has an opposite behavior than p_p. This can be seen in figure 3.2.

Having a robotic implementation in mind, Deneubourg et al. [6] proposed to compute f through a short-term memory each agent maintains. Keeping track of the last T time units, an agent counts the number of objects it finds; f is the number of objects encountered by the agent in the last T time steps divided by the maximum number of objects an agent can find in T time units.

3.2.2 Spatial Sorting

Let us now consider the case where there are N different types of objects. The behavior necessary to spatially sort all objects in the environment is very much the same as the described above but with N different f's instead of just one. This small change has profound effects on the collective behavior of agents because they are now trying to put together all objects of the same type not just aggregating objects. This model assumes that agents know how to perfectly differentiate one object from another; a knowledge that is normally not available for most practical purposes.

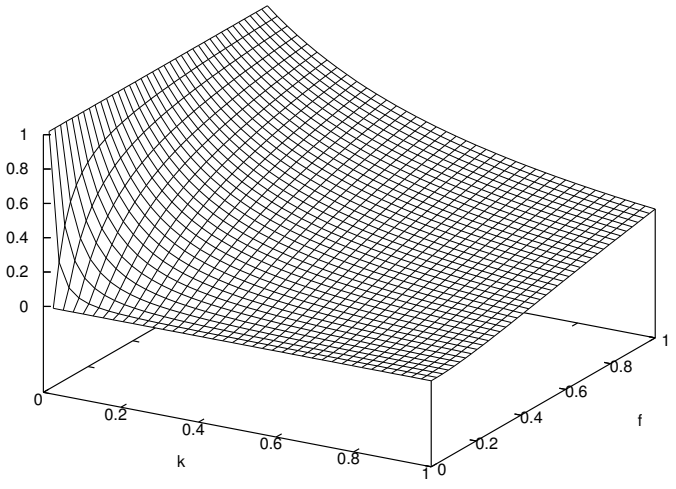


Figure 3.2: Probability of dropping an object as a function of k_2 and f in Deneubourg's model.

3.3 Exploratory Data Analysis

Lumer and Faieta [22] generalized the spatial sorting algorithm to apply it to exploratory data analysis. Taking as base model the one proposed by Deneubourg et al. [6], they extended it by using agents that could compute a similarity or dissimilarity measure between objects in the environment. The probabilities are computed as follows. The probability of picking a data element i is defined as

$$p_p(i) = \left(\frac{k_p}{k_p + f(i)} \right)^2 \tag{3.3}$$

where k_p is a constant and $f(i)$ is a similarity density measure with respect to element i. Likewise, the probability of dropping a data element is given by

$$p_d(i) = \begin{cases} 2f(i) & \text{if } f(i) < k_d \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

where k_d is a constant. The similarity density $f(i)$ for an element i, at a particular grid location τ, is defined as

$$f(i) = \begin{cases} \frac{1}{s^2} \sum_{j \in Neigh(\tau)} \left(1 - \frac{d(i,j)}{\alpha} \right) & \text{if } f > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where s^2 is the area of an agent's perception zone, an arrangement of $s \times s$ sites, centered at the location of the agent and α is a scaling factor of the dissimilarity measure $d(i, j)$ between elements i and j. If it is assumed that the elements can be represented as points in an n-dimensional space, then the Euclidean distance could be used as dissimilarity measure.

Algorithm 1 presents a high level description of the basic algorithm of Lumer and Faieta.

Algorithm 1: Lumer and Faieta's basic clustering algorithm

```
/* O is the set of data objects to be clustered */
/* A is the set of agents */

/* Initialization */
for all o_i ∈ O do
  Place o_i randomly on any free location on the grid
end for
for all agents ∈ A do
  Place agent randomly on any free location on the grid
end for

for t = 1 to t_max do
  for all agents ∈ A do
    if agent is unloaded and object o_i is on agent's location then
      Compute p_p(o_i) /* Using expression 3.3 */
      Draw random number r between 0 and 1
      if r ≤ p_p(o_i) then
        Pick up object o_i
      end if
    else
      if agent is loaded with o_i and location is free then
        Compute p_d(o_i) /* Using expression 3.4 */
        Draw random number r between 0 and 1
        if r ≤ p_d(o_i) then
          Drop object o_i
        end if
      end if
    end if
    Move randomly to any neighboring free location
  end for
end for
```

In experiments, Lumer and Faieta showed that the algorithm was capable of correctly classify elements of a synthetic data set; however, the number of clusters found were, in general, more than those expected. To fix this problem, they proposed three modifications to the basic algorithm.

The first modification was the introduction of variability into the agent population. The element of change was the velocity of displacement. Velocities then range from 1 to V_{max} and represent the number of grid positions the agent can move in a single time unit along a given direction vector. This variability was coupled to the sensitiveness of each agent to dissimilarity in the following way

$$f(i) = \begin{cases} \frac{1}{s^2} \sum_{j \in Neigh(\tau)} \left(1 - \frac{d(i,j)}{\alpha\left(1 + \frac{v-1}{V_{max}}\right)}\right) & \text{if } f > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The result is that an agent moving slowly will be more selective in their similarity criterion than an agent moving rapidly. The effect of this variability is that the number of final clusters is closer to the expected than the model with homogeneous agents. Fast moving agents appear to create clusters in a coarser scale than slowly moving agents which place data elements more carefully.

The second modification was the introduction of a short-term memory into agents so that they could keep track of the dropping location of their last m carried elements. This modification has the effect of lowering the probability that a just picked up element can create an independent cluster by itself, thus reducing the final number of clusters. This happens because every time an agent picks an element, it looks in its memory for the location of the closest element in attribute space. The agent, instead of moving randomly, moves directly toward that particular location.

The third modification was that of behavioral switches. If an agent has not manipulated an item for a predetermined number of time units, it starts destroying the so far created clusters. This does not end up in a total cluster destruction because other agents that have not reached that state will reposition the disseminated objects.

3.4 Engineered Ant-Based Clustering Algorithms

As stated before, ant-based clustering algorithms are *inspired* by the collective behavior of social insects' colonies. However, this does not mean that we should "stick" with the natural inspiration. In engineering, we are interested in ant-based clustering algorithms because they offer advantages over other more traditional approaches, not because they help us model ants' behavior. Therefore, we look for improvements on their speed, efficiency, and the quality of their final solutions.

Handl et al. [13, 15, 14] proposed a series of modifications to the Lumer and Faieta's clustering algorithm to improve its clustering, sorting, and robustness performance. They started by tackling the problem of automatic clusters extraction from the grid. Because there is no information about the correct number of clusters, they applied a single-linkage hierarchical clustering algorithm over the grid. However, this algorithms presents the problem of "bridging", i.e., it tends to merge two clusters into one just because of some isolated

data elements between them. To overcome this problem, they added a weighting term to encourage the merging of the element around the borders of clusters. In addition, they also modified the similarity density function $f(i)$ in such a way as to penalize high dissimilarities.

Another modifications introduced by Handl et al. for improving the spatial separation of clusters on the grid were: the use of the Lumer and Faieta's short-term memory model to explore the neighborhoods of all remembered dropping locations in order to bias the agents' random walks to the most probable dropping location; an increasing radius of perception to favor a rapid dissolution of early clusters and the improvement of final clusters; a time-dependent modulation of the neighborhood function to improve spatial separation of clusters on the grid; and a modified threshold functions to take advantage of all the previous modifications.

With respect to robustness, Handl et al., proposed the activity-based adaptation of the parameter α. Because the parameter α affects the agents' sensitiveness to dissimilarities among data objects, the number of cycles an agent has not manipulated an object is a signal of an erroneous α value. Therefore, the α parameter is proposed to adapt all the time depending on the activity of a given agent. For more information about the modifications introduced by Handl et al. and their results, see [14].

Algorithm 2 presents a high level description of the basic algorithm proposed Handl et al.

3.4.1 Ant-Based Clustering Quality Assessment

In an effort to formally evaluate the clustering quality of ant-based clustering algorithms, Handl et al. [12] applied four validity measures to the resulting clustering from a modified version of the Lumer and Faieta [22] algorithm.

The four validity measures used by Handl et al. in [12] were:

The F-Measure. The harmonic mean of recall and precision, also known as the *F-Measure*. Commonly associated to the information retrieval field, recall and precision are measures that give us some idea of how well a clustering algorithm is identifying the classes present in a database. In the context of classification, recall is defined as $r(i,j) = \frac{n_{ij}}{n_i}$ where n_{ij} is the number of elements of class i in cluster j and n_i is the number of elements of class i. Precision is defined as $p(i,j) = \frac{n_{ij}}{n_j}$ where n_j is the number of elements in cluster j. For a class i and a cluster j the *F-Measure* is defined by

$$F(i,j) = \frac{2\,p(i,j)\,r(i,j)}{p(i,j) + r(i,j)}$$

The overall *F-Measure* for the classification generated by the clustering algorithm is given by

$$F = \sum_i \frac{n_i}{n} \max_j \{F(i,j)\} \tag{3.7}$$

Algorithm 2: Handl et al.'s basic clustering algorithm

```
/* O is the set of data objects to be clustered */
/* A is the set of agents */

/* Initialization */
for all o_i ∈ O do
   Place o_i randomly on any free location on the grid
end for
for all a_j ∈ A do
   Select randomly a data object o_i
   Load agent a_j with object o_i and place it randomly on any free location on the grid
end for

for t = 1 to t_max do
   Select randomly an agent a_j
   Displace agent a_j stepsize units along its direction vector
   if a_j's location is free then
      Compute p_d(o_i)
      Draw random number r between 0 and 1
      if r ≤ p_d(o_i) then
         Drop object o_i
         while agent a_j is unloaded do
            Select randomly a data object o_i
            Compute p_p(o_i) /* In object's neighborhood */
            Draw random number r between 0 and 1
            if r ≤ p_p(o_i) then
               Pick up object o_i
            end if
         end while
      end if
   end if
end for
```

where n is the size of the data set. F is limited to the interval $[0, 1]$ with a value of 1 with a perfect clustering.

The Rand Statistic. Determines a similarity measure between the known perfect classification C and the partition generated by the clustering algorithm P. It is defined as

$$R = \frac{a + d}{a + b + c + d} \tag{3.8}$$

where

- a is the number of pairs where both elements belong to the same class in C and to the same group of the partition P.
- b is the number of pairs where both elements belong to the same class in C and to different groups in P.
- c is the number of pairs where both elements belong to different classes in C and to the same group in P.
- d is the number of pairs where both elements belong to different classes in C and to different groups in P.

Note that $a + b + c + d = N(N - 1)/2$ where N is the total size of the data set. The Rand statistic is limited to the interval $[0, 1]$ with a value of 1 with a perfect clustering.

The intra-cluster variance. This measure tries to capture the idea that members that belong to the same cluster should be as close to each other as possible. It is given by

$$I = \sum_{i=1}^{n} \sum_{p \in C_i} \|p - \mu_i\|^2 \tag{3.9}$$

where n is the total number of groups in the partition generated by the clustering algorithm, and μ_i is the centroid of group i. This measure is to be minimized.

The Dunn index. This measure will give a high value whenever the partition gives compact and well separated clusters. It is given by

$$D = \min_{c,d \in C} \left\{ \frac{\|\mu_c - \mu_d\|}{\max_{e \in C}\{diam(e)\}} \right\} \tag{3.10}$$

where μ_i is the centroid of cluster i and $diam(i)$ is the diameter of cluster i which could be considered a dispersion measure. It is defined as

$$diam(c) = \max_{x,y \in c}\{\|x - y\|\}$$

For comparison purposes, we chose to use the same clustering quality measures

Chapter 4

Direct communication among agents in ant-based clustering algorithms

This chapter presents the effects on the quality of the clustering obtained by using direct communication among agents in ant-based clustering algorithms. The information exchange occurs whenever a group of agents coincide on the grid. An agent receives information from its peers about the environment state to update its own environment state representation. The local environment state is later used by an agent to bias its random walk in search for dropping locations.

The isolated effects of the environment state on agents' actions are also explored by eliminating the update of each agent's local environment state representation, but allowing the bias in their random walks.

4.1 Direct communication strategies

In all previous attempts to perform clustering tasks using a simulated insect colony there is no direct interaction among agents. This is perhaps due to the fact that early works on ant-based clustering and sorting focused on robotic implementations [6, 16, 1, 20] where direct real-time communication is much more complicated than in software simulations.

If we let agents exchange information about the spatial distribution of data objects on the environment, it should be possible for them to enrich or update their own environment state representations. The ultimate goal for information exchange is to let agents choose the best location on the environment on which to drop a data object, and therefore create better clusters. This is what has been done in the series of experiments presented in this chapter.

From what was said in the previous paragraphs it is clear that we need agents represent their environment. To explore the effects of using different environment representations, I first used a natural extension of the short-term memory model proposed by Lumer and Faieta [22]. Whenever two or more agents coincide on the grid, they exchange their short-term memories and each augment its own short-term memory. When two agents have recently manipulated the same data object, the agent with the oldest information updates its memory while the other leaves it unchanged.

The second strategy explored in this thesis is the creation of environment maps. In an environment map, an agent represents the spatial distribution of data elements. A succinct representation of a spatial distribution is accomplished through the use of self-organizing maps or SOM's [19]. However, because of the limitations of conventional SOM's; namely, the *a priori* fixed number of neurons and the problem of "dead" neurons or neurons that do not update their weight vectors due to a misplacement in the input space, each agent must create a dynamic self-organizing map. The particular implementation of the dynamic self-organized map should not change the final effects as long as it accurately represents the spatial distribution of data elements. In order to accurately represent the environment, an agent should update its map as it explores the environment, regardless whether it is in search for a dropping location or just wandering through the environment. With the environment map representation, each time two agents exchange information they complement their maps as is described in subsection 4.2.2.

As with the model of Lumer and Faieta [22], whenever an agent picks up a data object, it looks for a favorable dropping location. However, instead of looking into its memory for the closest previously manipulated object, it classifies it using its own self-organizing map to bias its random walk to the location of the winner neuron. The winner neuron should be located near the most similar object to the one just picked up because the network reflects the spatial distribution of objects in the environment.

In the second series of experiments in which no representation update is done, loaded short-term memory agents explore their peers' memories to look for the closest object to their load in attribute space. If an agent finds a closer object in one of its peers' memories, the agent redirects itself toward the location of that object.

In the environment map representation, whenever a loaded agent coincides on the grid with one or more agents, it classifies its load using its peers' neural networks to determine the closest winner neuron. If it is closer in the attribute space than its own winner neuron, then the agent redirects itself toward the location of that neuron on the grid.

4.2 Experimental Setup

Two well-known test databases were used for the experiments. Each one of them with different difficulty levels. In order to deal with them, the way an agent evaluates its neighborhood to compute a similarity density was modified. In the following, a more detailed explanation of the experimental setup is presented.

4.2.1 Test Data

In order to evaluate the impact of the introduction of information exchange between agents I used two real data collections from the UCI Machine Learning Repository [5]. These were:

- Iris Plant Database. This database is composed of 150 instances with 4 numeric attributes each. There are 3 classes composed of 50 instances each. Of these, one is linearly separable from the other two; the latter are not linearly separable from each other.

- Wine Recognition Database. This database is composed of 178 instances with 17 numeric attributes each. There are 3 classes in the database where class 1 has 59 instances, class 2 has 71 instances, and class 3 has 48 instances.

To eliminate the bias on similarity measures provoked by different scales within data attributes, all data sets were standardized. The similarity measure used in all the experiments was the cosine metric[1].

4.2.2 Agent Models

The agents picking and dropping probabilities were computed using expressions 3.3 and 3.4 respectively. However, because the cosine metric is a similarity measure, expression 3.5 is not directly applicable. Therefore, for the local similarity density $f(i)$, the following expression was used

$$f(i) = \frac{1}{s^2} \sum_{j \in Neigh(\tau)} \left(\frac{1}{1 + e^{-S\frac{d(i,j)}{\alpha} + D}} \right) \tag{4.1}$$

where S is the steepness of the response curve and D serves as a displacement factor. In our experiments, S was fixed to 5 because it provides a similarity value close to 0 when the cosine measure is minimum, that is, when the cosine measure gives a value of -1, and D to 1 because this allows us to better distinguish vectors with separation angles between 0 and $\pi/2$. The resultant similarity response surface is shown in figure 4.1.

This local similarity density function permits the integration of simple background knowledge. In particular, if it is known that data classes are difficult to separate, one can move the displacement factor near the maximum value of the similarity measure to better discriminate among very similar data elements. This expression also has the advantage of limiting the range of values α can take to $(0, 1]$, given that $-1 \leq d(i, j) \leq 1$, as is the case for the cosine metric.

Table 4.1 summarizes all other agent settings used in the experiments[2].

All previously described settings were used for the Lumer and Faieta [22] basic algorithm and for the simple memory exchange algorithm. For the environment map building algorithm, the technique of choice was a growing neural gas network to represent data spatial distribution. Introduced by Fritzke [10] as an approach to overcome some of the limitations of conventional self-organizing maps, a growing neural gas network consists of:

[1] In preliminary experiments, it proved to give better results than Euclidean distance.

[2] The settings in tables 4.1 and 4.2 were derived experimentally, and proved to give good results for the given test databases.

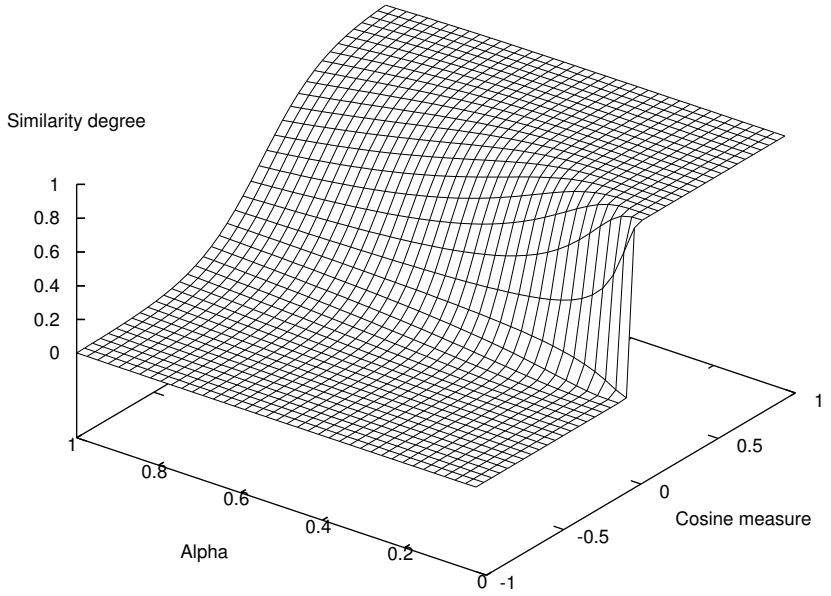


Figure 4.1: Response surface. Similarity response defined as a function of the cosine metric between two data elements and the scaling factor α. Taken from the similarity component of equation 4.1 where $S = 5$ and $D = 1$.

- a set A of units (or nodes). Each unit $c \in A$ has an associated *reference vector* $w_c \in \mathbf{R}^n$. The reference vectors can be regarded as positions in input space of the corresponding units.
- a set N of connections (or edges) among pairs of units. These connections are not weighted. Their sole purpose is the definition of topological structure.

The idea behind the training algorithm is to successively add new units to an initially small network by evaluating local statistical measures gathered during previous adaptation steps. The network topology is generated incrementally using a competitive Hebbian learning rule. The dimensionality depends on the input data and can vary locally.

The training algorithm basically begins with two randomly located units and an input signal that is to be learned. The learning rule used to adapt the reference vectors of the unit that is closest in the input space to the current input signal and its topological neighbors is

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \epsilon(\xi - \mathbf{w}^{(t)}) \tag{4.2}$$

where $\mathbf{w}$ is the reference vector of the adapting unit, ϵ is a constant called *learning rate*, ξ is the input signal being learned. The ϵ constant has different values for the adapting unit and its topological neighbors and are labeled ϵ_b and ϵ_n respectively.

The squared error of the nearest unit in turn is accumulated, so that after λ input signals have been learned, a new unit is created half between the two neighboring units

Table 4.1: Agent settings for direct communication experiments

Parameter	Value
k_p	0.1
k_d	0.15
α	0.7
Neighborhood size	5×5
Short-term memory size	8

with the highest accumulated errors. The deletion of units happens whenever a unit is not topologically connected with any other unit. This occurs after that a given number of input signals have not fired a unit. This process continues until a stop criterion is met. The complete training algorithm can be found in [10].

The embedded growing neural gas networks have a parameter set on their own, the reader is referred to [10] for a detailed explanation of the meaning and effects of all these parameters. Table 4.2 summarizes the parameter set for the embedded growing neural gas networks used in our experiments.

Table 4.2: Embedded growing neural gas networks settings

Parameter	Value
Winner neuron learning rate ϵ_b	0.8
Neighboring neurons learning rate ϵ_n	0.005
Maximum edge age	50
Growing threshold λ	100
Local error decreasing rate α	0.5
Global error decreasing rate d	0.995

A learning rate of 0.8 is normally considered too high, however, due to the fact that the ant environment is highly dynamic, a high learning rate is needed in order to allow the network learn the real data distribution on the grid. A crucial condition for the proper operation of this model.

In the experiments, the growing neural gas networks were represented by two sets of nodes. One set of nodes in the attribute space of the database, and the other in the plane, i.e., in the grid. Because the algorithm uses a competitive Hebbian learning rule, adaptation occurs whenever an input vector falls within the Voronoi region of a node. The Voronoi

region for a node p_i is defined as

$$V(p_i) = \{x : |p_i - x| \leq |p_j - x|, \forall j \neq i\} \tag{4.3}$$

A Voronoi diagram is a combination of Voronoi regions as can be seen in figure 4.2.

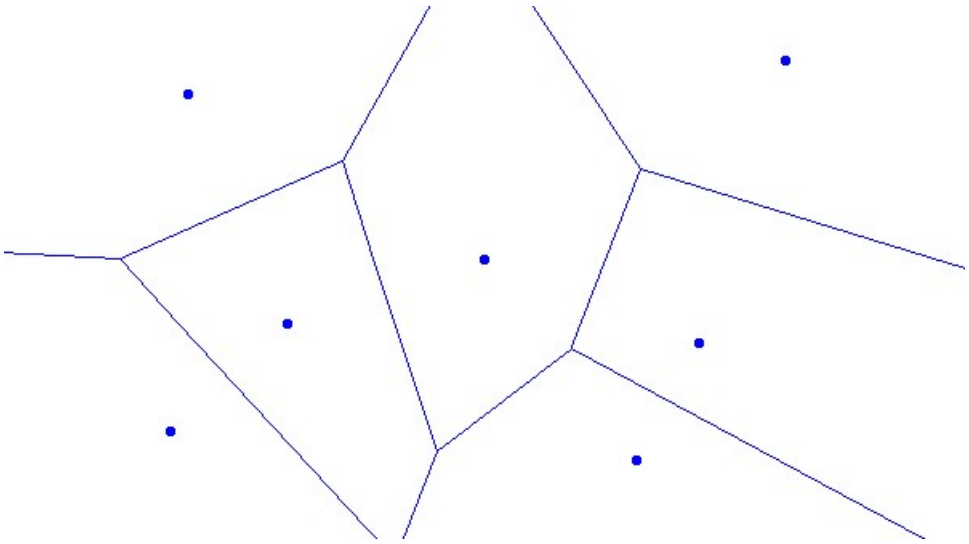


Figure 4.2: Voronoi diagram of a set of 6 nodes in the plane.

To combine two growing neural networks, we relied on their Voronoi diagrams. To exemplify the process, suppose we want to combine the diagram shown in figure 4.2 with the one shown in figure 4.3.

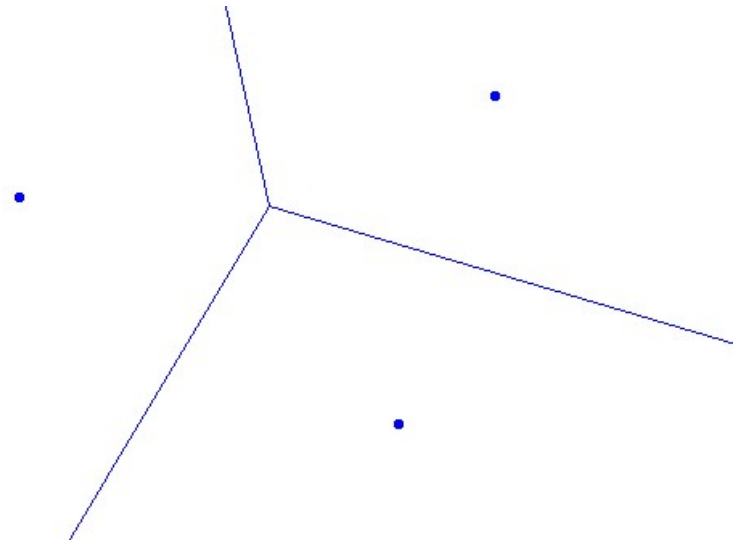


Figure 4.3: Voronoi diagram of a set of 3 nodes in the plane.

Since the goal of combining two environmental maps represented by growing neural gas networks is to better approximate the actual distribution of data on the grid, it is obvious

that large maps will have priority over small ones. In our example, it means that we will alter the network with six nodes with the information provided by the map with three nodes.

The first step in the combination process is to find on which Voronoi region fall the nodes of the modifying network. Figure 4.4 shows graphically this process.

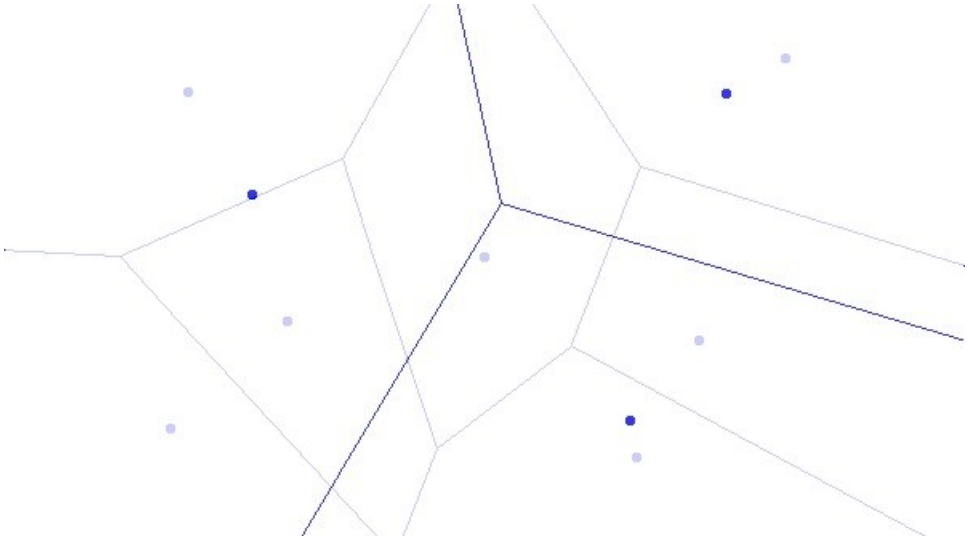


Figure 4.4: Superimposition of two Voronoi diagrams.

The second step consists in updating the nodes that are closest to the nodes just introduced, that is, the nodes that generate the Voronoi region on which the modifying nodes fell. The new position of a node, in both the attribute space and in the grid, is the centroid of the nodes involved. This is shown in figure 4.5.

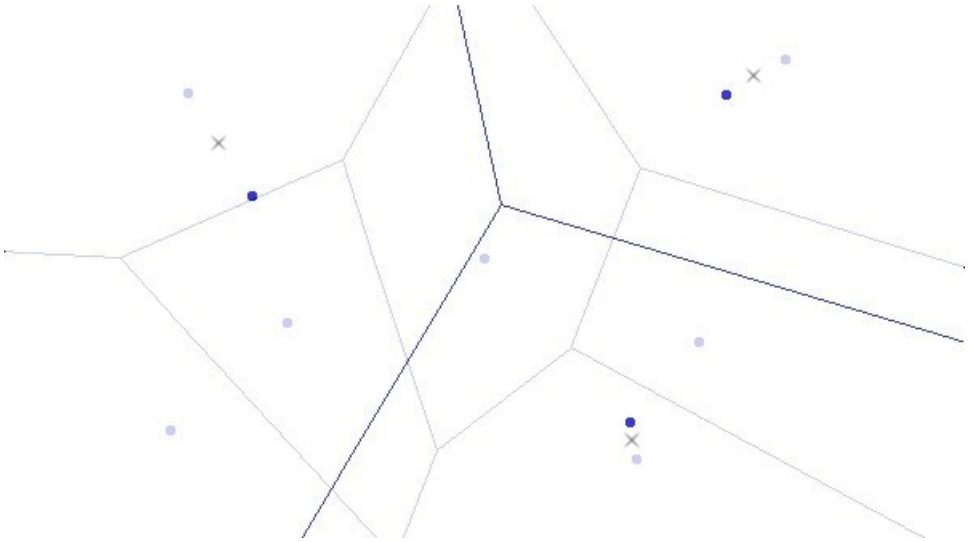


Figure 4.5: Network update as a result of two superimposed Voronoi diagrams. The stars indicate the new position of the modified node.

As a result of the network update, the two agents involved in the exchange end up with the same network. Figure 4.6 shows the final configuration of the Voronoi diagram after the modification.

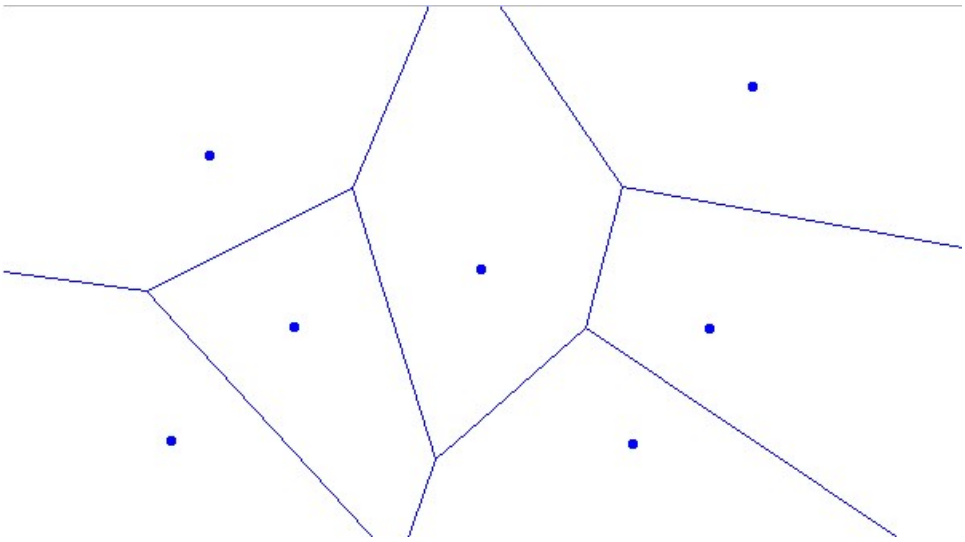


Figure 4.6: Voronoi diagram of the resultant network.

4.2.3 Experiments Design

To observe the effects of information exchange among agents during the clustering process, the four measures used by Handl et al. in reference [12] were applied every 10,000 simulation cycles to the partial clustering hitherto obtained. Each simulation cycle was composed of N individual actions, where N was the number of agents in the simulation. All algorithms were tested 30 times with every database for 1,000,000 simulation cycles.

In the experiments, information exchange occurs only whenever two or more agents coincide on the grid. We can expect therefore, that the probability of an encounter raises as the number of agents is increased, or more precisely, as the agent density of the environment increases. By having more agent encounters over time, the effects of information exchange should have more impact. Therefore, we tried with populations of 10 and 30 agents within an environment of 100×100 locations in all the experiments.

Since the final goal of information exchange is to approximate the environment state, we also include the results obtained with a version of the algorithm in which agents are omniscient, i.e., agents that know exactly the spatial distribution of data objects on the grid all the time. In this version, when agents pick up a data object, they direct themselves toward the closest data object available on the grid. Hence, there is no need for any environment representation in these agents.

In order to collect information from the spatial partition formed by the agents on the grid, an agglomerative hierarchical algorithm was applied on the data using Euclidean

distance and the single linkage strategy. The maximum distance to merge clusters was set equal to the size of the agents neighborhood. The distance used to compute the variance and the clusters diameters in the attribute space was also the Euclidean distance.

4.3 Results of Experiments on Information Exchange for Environment Representation Updates

In this section and in the following, five different algorithms were tested:

- the basic Lumer and Faieta's short-term memory agents algorithm without communication among agents;
- the Lumer and Faieta's algorithm with map creating agents without communication among them;
- an omniscient agents algorithm;
- an algorithm in which agents update their short-term memories;
- and an algorithm in which agents update their distribution maps.

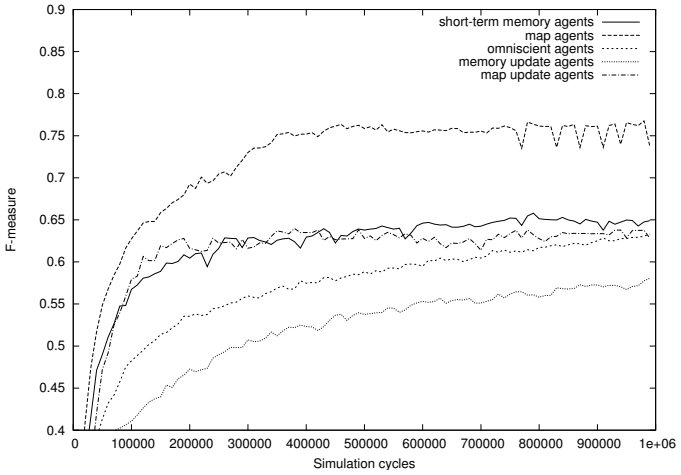
In the last two algorithms, the update is done whenever agents exchange information.

The results are grouped by validity measure. Firstly, the resemblance measures between the obtained clustering and the known correct classification, i.e., the F-measure and Rand statistic. Secondly, the quality measures based on general statistical properties a good clustering should have, i.e., the intra-cluster variance and Dunn index. Subsection 4.3.5 presents the convergence of the algorithms to the correct number of clusters. Results with 20 agents are skipped because, in all cases, they showed a behavior in between the results with 10 and 30 agents.

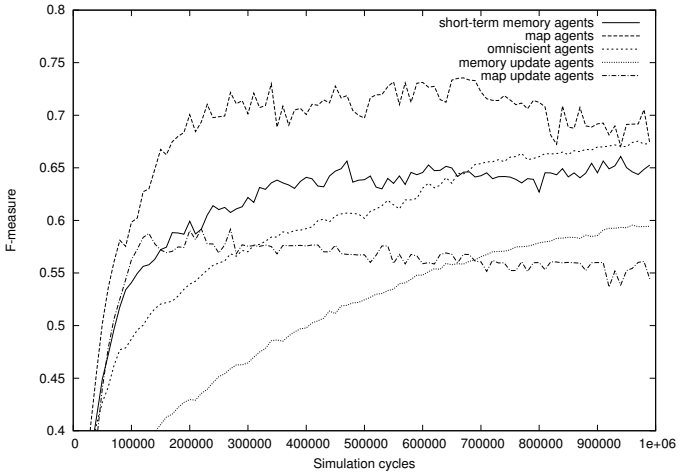
4.3.1 F-Measure

Figure 4.7 shows the F-Measure scores over time obtained by the tested algorithms on the Iris Plant and Wine databases using 10 agents. On both databases, the algorithm with noncommunicating map creating agents performs better than the other four algorithms.

Interesting enough, the algorithms with noncommunicating agents perform better than the algorithm with omniscient agents during the first 680,000 cycles with the Wine database and all the time with the Iris database. The algorithm with omniscient agents has a different response function than the algorithm with noncommunicating agents. It is slower during the first cycles, but it improves almost monotonically. The same is true for the algorithm with memory updating agents. The only difference is a negative offset with respect to the omniscient agents algorithm.



(a) Iris database

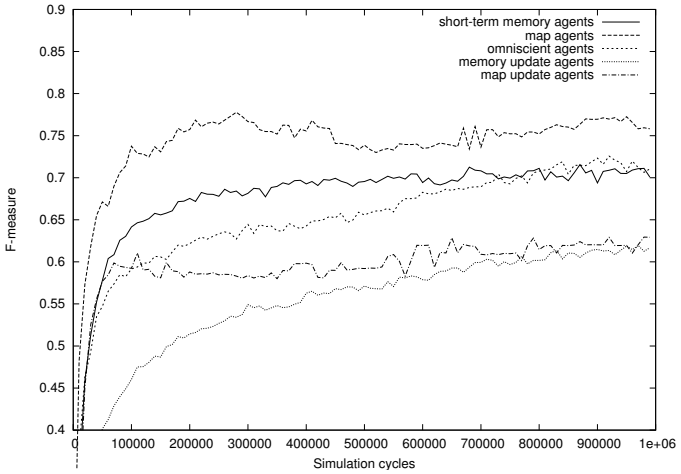


(b) Wine database

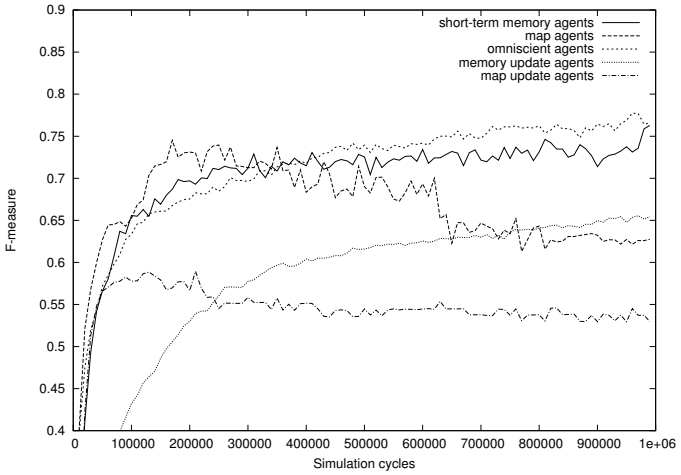
Figure 4.7: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents. A score value of 1 means perfect classification.

Figure 4.8 shows the same scores than figure 4.7 but with 30 agents. In the Iris database, the algorithm with noncommunicating map creating agents performs better than the others all the time. This is not the case in the Wine database in which the best performing algorithm is the one with omniscient agents. The algorithm with map updating agents performs terribly bad following the same trend of above. The characteristic response function of the algorithm with memory updating agents is the same as before, including the negative offset respect to

the omniscient agents algorithm.



(a) Iris database

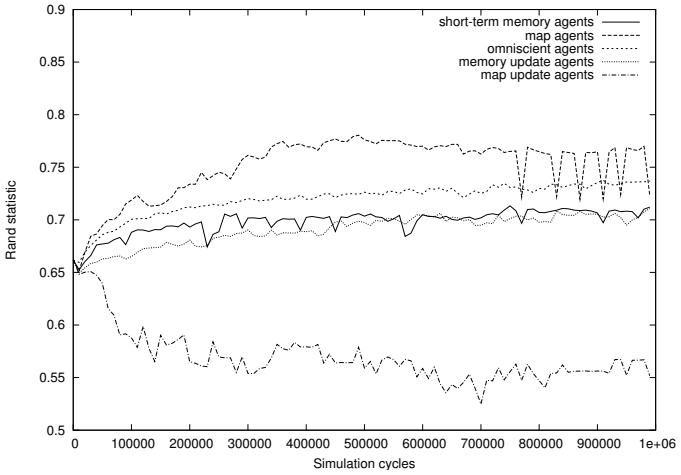


(b) Wine database

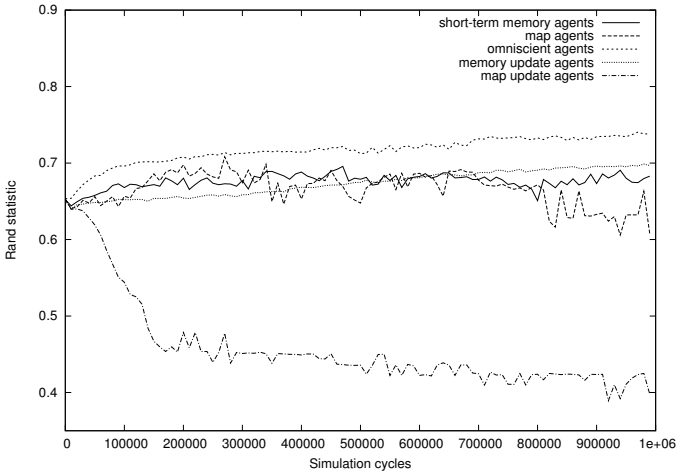
Figure 4.8: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents. A score value of 1 means perfect classification.

4.3.2 Rand Statistic

The Rand statistic scores over time for all five algorithms on the Iris Plant and Wine databases using 10 agents are shown in figure 4.9.



(a) Iris database



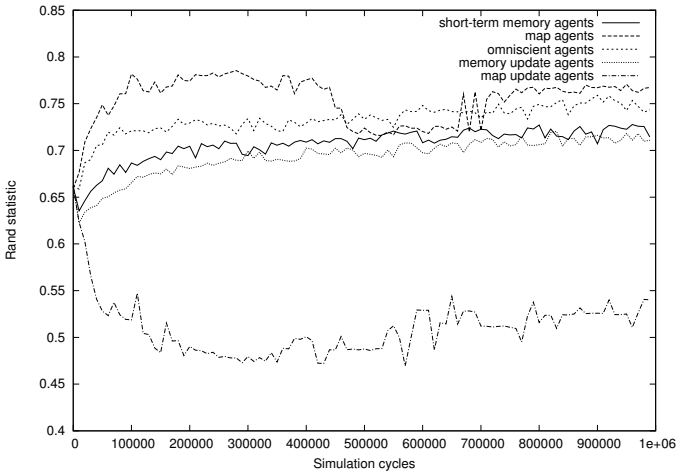
(b) Wine database

Figure 4.9: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents. A score value of 1 means perfect classification.

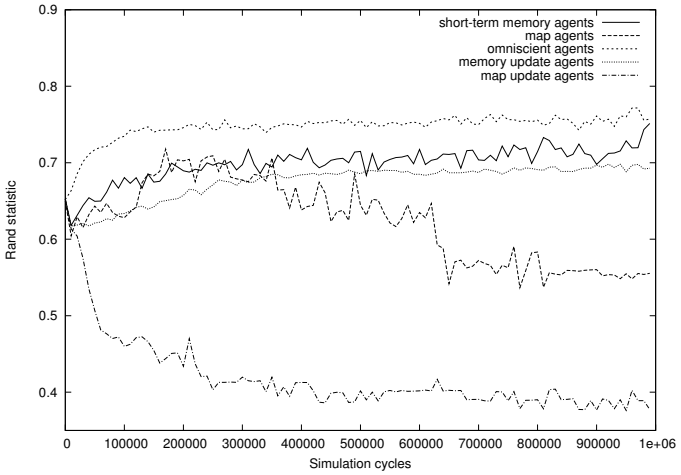
In both cases the worst performing algorithm is the one with map updating agents. The algorithm with noncommunicating short-term memory agents and the memory updating agents algorithm perform equally well. Notice that the memory updating agents algorithm has the same negative offset respect to the omniscient agents algorithm seen before. With the Iris database the map creating agents algorithm is the best.

In figure 4.10 almost the same situation than in figure 4.9 can be seen. In the Iris

database the best algorithm is the one with map creating agents. The best performing algorithm in the Wine database is the one with omniscient agents, the worst is the map updating algorithm, and the other two algorithms behave almost the same.



(a) Iris database

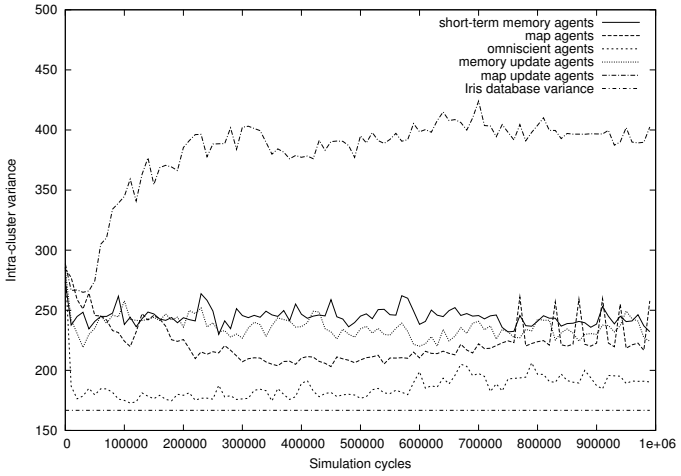


(b) Wine database

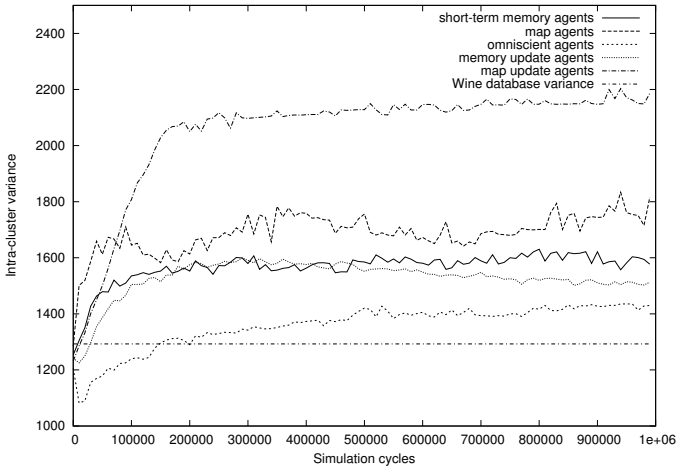
Figure 4.10: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents. A score value of 1 means perfect classification.

4.3.3 Intra-cluster Variance

The plots seen in figure 4.11 correspond to the intra-cluster variance scores for all tested algorithms with 10 agents on the two test databases.



(a) Iris database

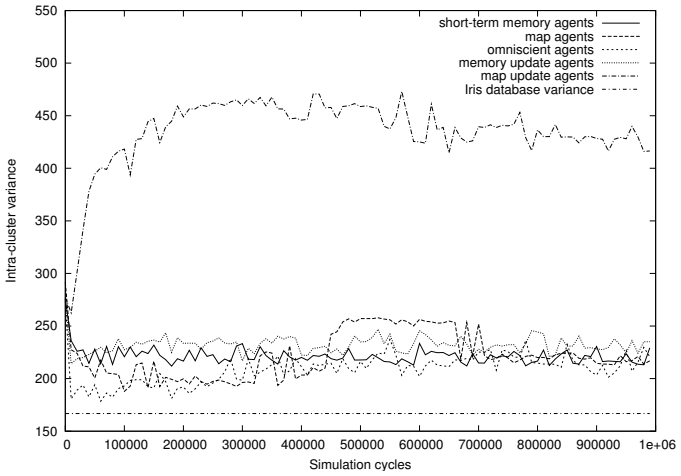


(b) Wine database

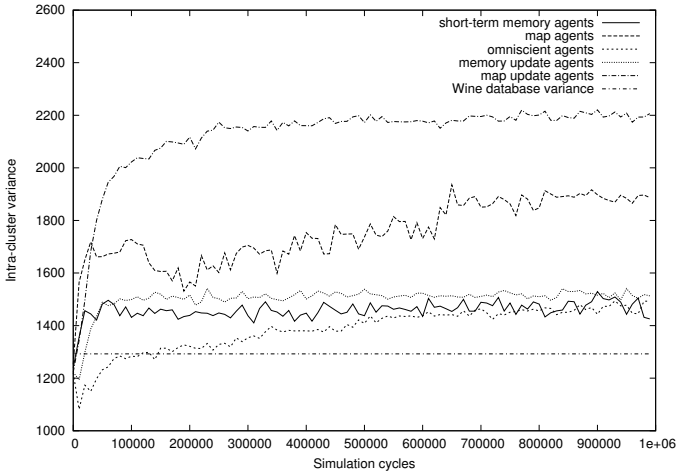
Figure 4.11: Total intra-cluster variance over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents. As a reference, the total intra-cluster variance of the correct clustering is shown.

The algorithm with omniscient agents performs better than the others. The worst

performing algorithm is the one with map updating agents, and Lumer and Faieta's short-term memory agents algorithm and the memory updating agents algorithm perform almost equally well. As a reference, the total intra-cluster variance of the correct clustering is shown. Remember that this measure is to be minimized.



(a) Iris database

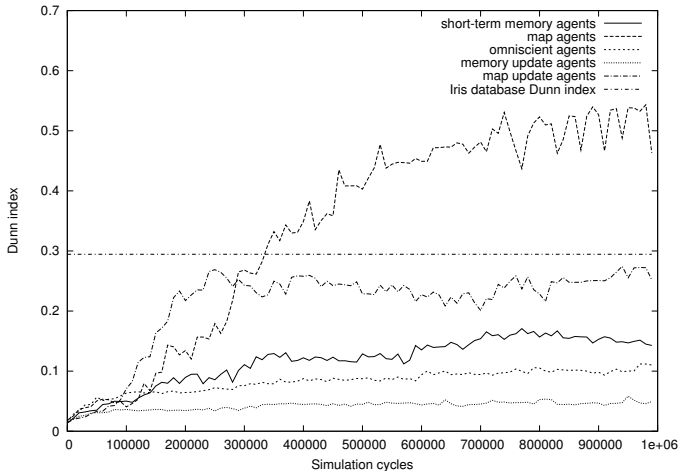


(b) Wine database

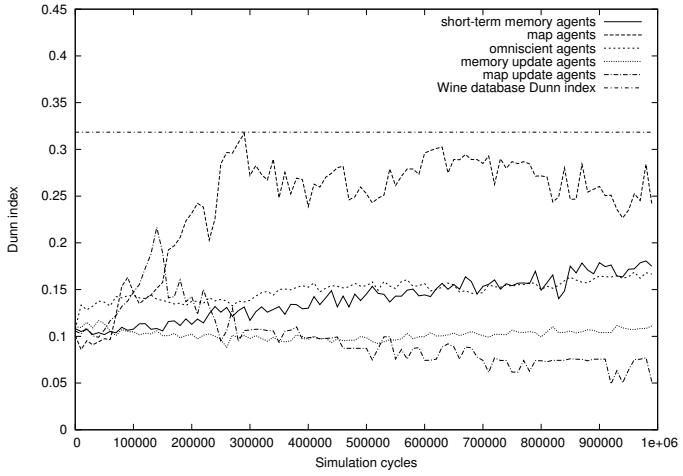
Figure 4.12: Total intra-cluster variance over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents. As a reference, the total intra-cluster variance of the correct clustering is shown.

Figure 4.12 shows how the total intra-cluster variance gap between the omniscient

agents algorithm and the memory-based algorithms gets smaller as the number of agents increases. Wit the Iris database, the final performance is almost the same for all algorithms except for the map updating agents algorithm. The map creating agents algorithm exhibits the same behavior no matter how many agents there are.



(a) Iris database

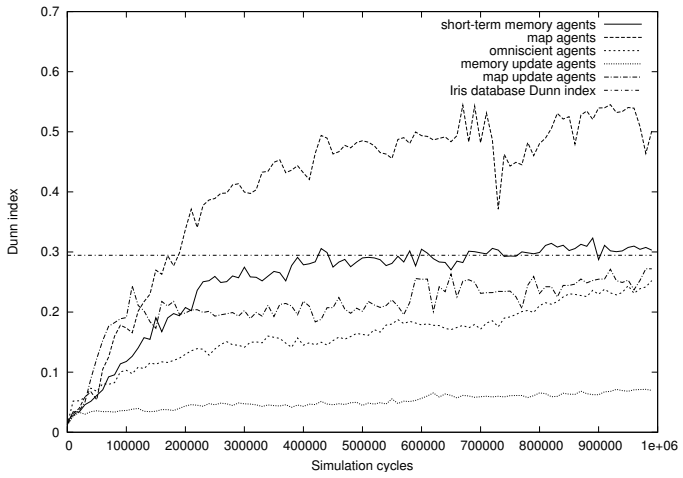


(b) Wine database

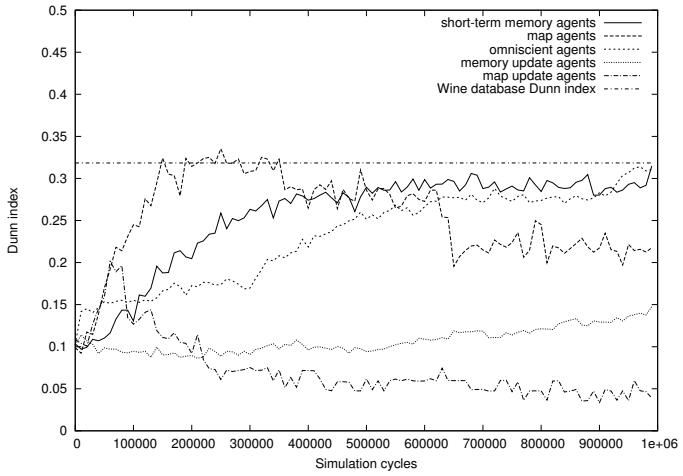
Figure 4.13: Dunn index score over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents. As a reference, the Dunn index score of the correct clustering is shown.

4.3.4 Dunn index

Figure 4.13 shows the Dunn index scores for all tested algorithms with 10 agents on both test databases.



(a) Iris database



(b) Wine database

Figure 4.14: Dunn index score over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents. As a reference, the Dunn index score of the correct clustering is shown.

In the figure 4.13, for both databases, the map creating agents algorithm achieves the best scores. In the Iris database, the best performing algorithm is the one with map

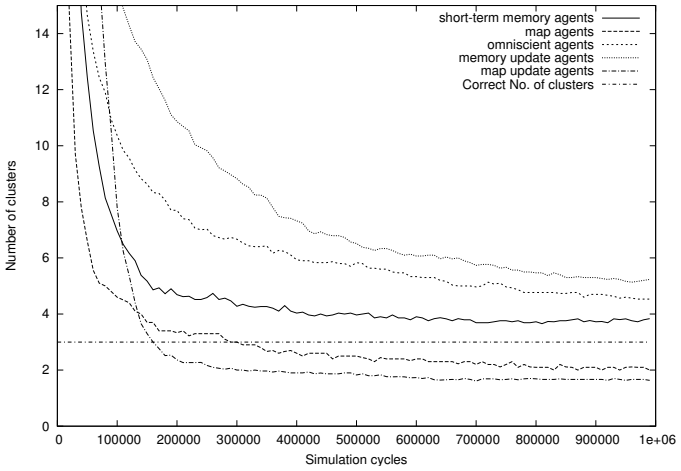
updating agents. This does not necessarily means it is finding good clusters as is shown in figures 4.7,4.9,and 4.11. The performance of the other two algorithms is in between the performance of the two just mentioned. In the wine database, the best performing algorithm during the first cycles was the map updating algorithm. However, after 120,000 cycles approximately, it starts dropping to become the worst performing.

Figure 4.14 shows the Dunn index scores for all four algorithms with 30 agents on the two test databases. In this case, the map creating agents algorithm is still the best, however, after a while and just with the Wine database, the basic short-term memory agents algorithm becomes the best. The memory updating agents algorithm behaves exactly as before, i.e., it makes no substantial improvement over time.

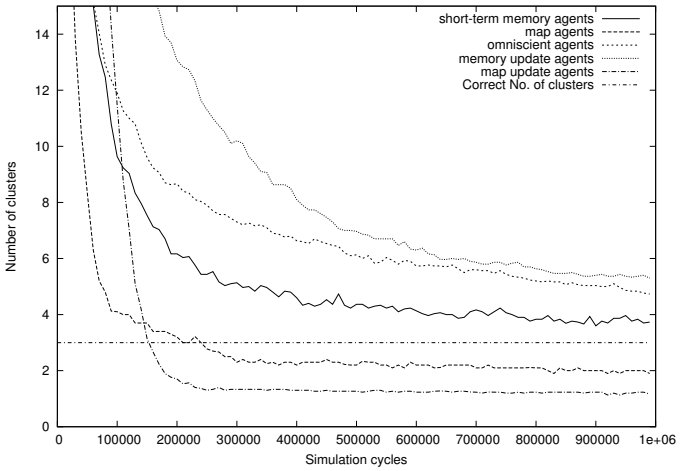
4.3.5 Number of clusters

Figures 4.15 and 4.16 show the number of clusters discovered over time with 10 and 30 agents respectively.

Two important things can be noted by inspecting figures 4.15 and 4.16. First, the relative convergence speed toward a particular number of clusters is the same across both databases and in spite of using a different number of agents. The slowest is the memory updating agents algorithm, it is followed by the omniscient agents algorithm which is followed by the simple short-term memory agents algorithm, followed by the map creating agents algorithm. The fastest algorithm is the one with map updating agents. Perhaps too fast, because it misses by far the correct number of clusters. Second, they show how the number of found clusters decreases as the number of agents is increased.

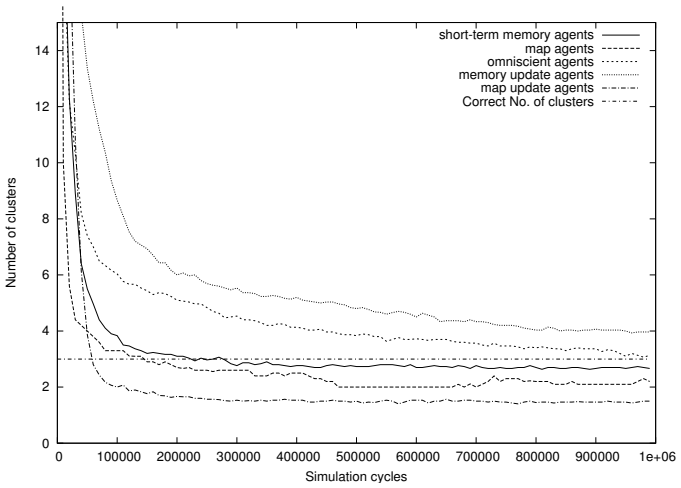


(a) Iris database

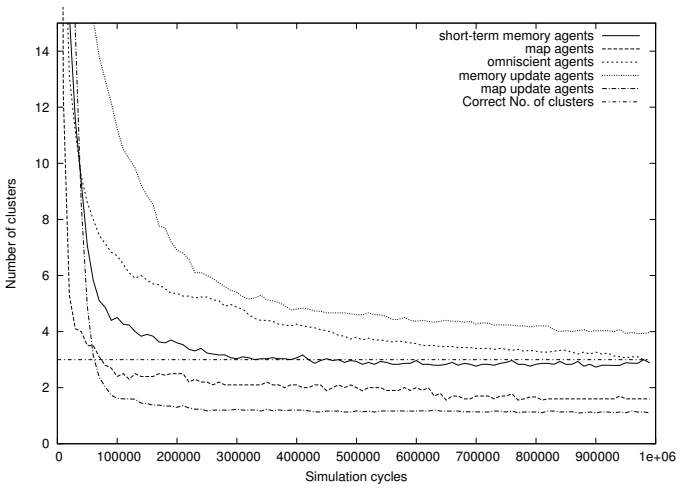


(b) Wine database

Figure 4.15: Number of found clusters over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents. As a reference, the correct number of clusters is shown.



(a) Iris database



(b) Wine database

Figure 4.16: Number of found clusters over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents. As a reference, the correct number of clusters is shown.

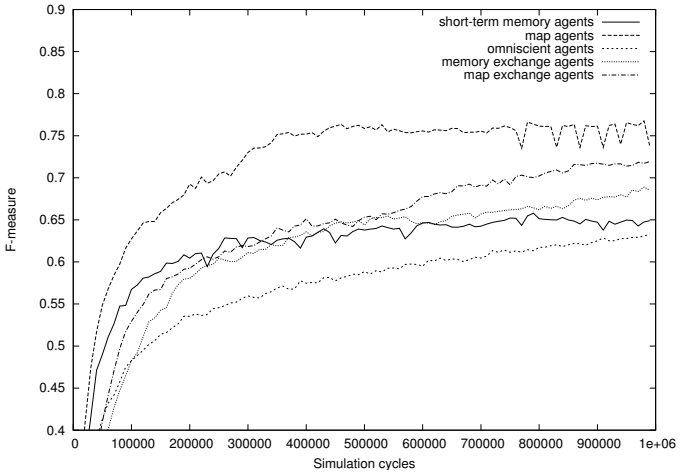
4.4 Results of Experiments on Information Exchange for Behavior Changes

The same algorithms that were tested in section 4.3 were used in this section. The difference rely on the agents' actions after information exchange. In this section, agents

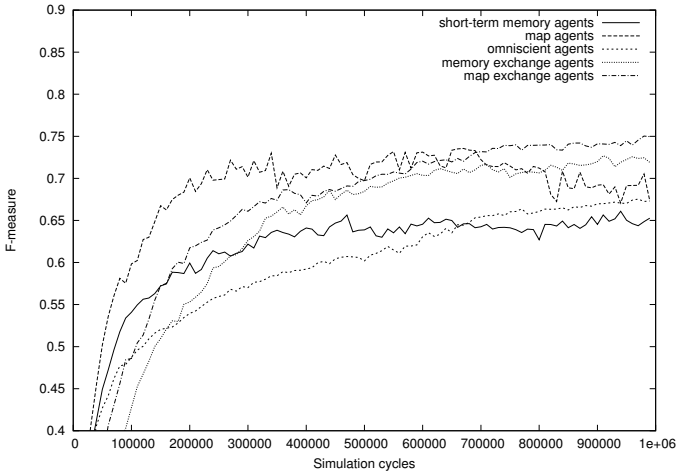
change their dropping spot search trajectory based on the contents of other agents' messages. The results are also grouped by validity measure.

4.4.1 F-Measure

Figure 4.17 shows the F-Measure scores over time on the Iris Plant and Wine databases using 10 agents. With the Iris database, the best performing algorithm is the one with map creating agents, followed by the map exchange agents version. With the Wine database, the best performing algorithm is the one with map creating agents but only during the first 600,000 cycles. After that, the best is the map exchange agents version. In both cases, the short-term memory agents algorithm performs very poorly. The omniscient agents algorithm is the second worst.



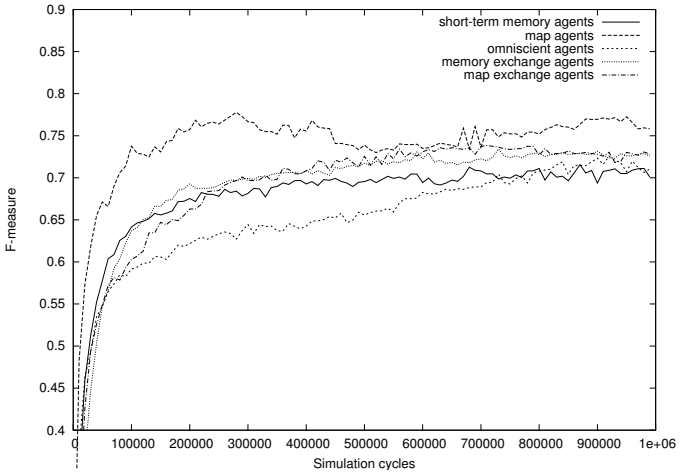
(a) Iris database



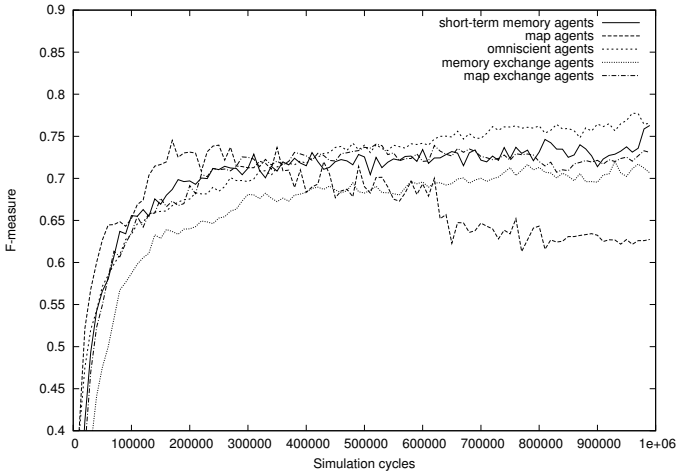
(b) Wine database

Figure 4.17: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents. A score value of 1 means perfect classification.

Figure 4.18 shows how the relative performance of the algorithms with agents exchanging information about the distribution of data objects on the environment falls when using 30 agents. The best performing algorithm with the Iris database is the map creating agents version. With the Wine database, the best algorithm is the omniscient agents algorithm. The information exchange algorithms become the worst with the Wine database.



(a) Iris database



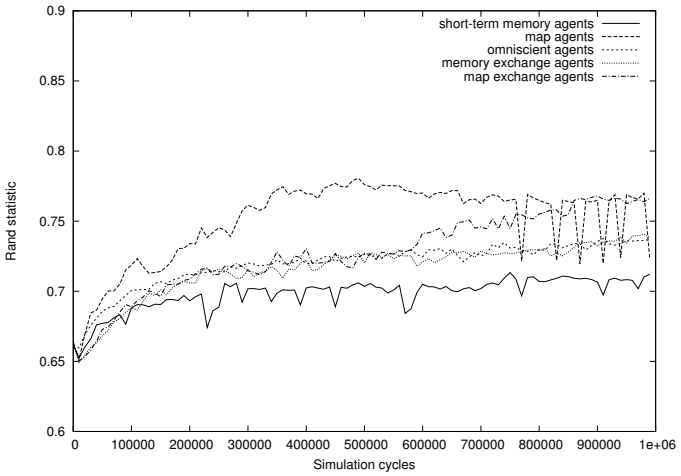
(b) Wine database

Figure 4.18: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents. A score value of 1 means perfect classification.

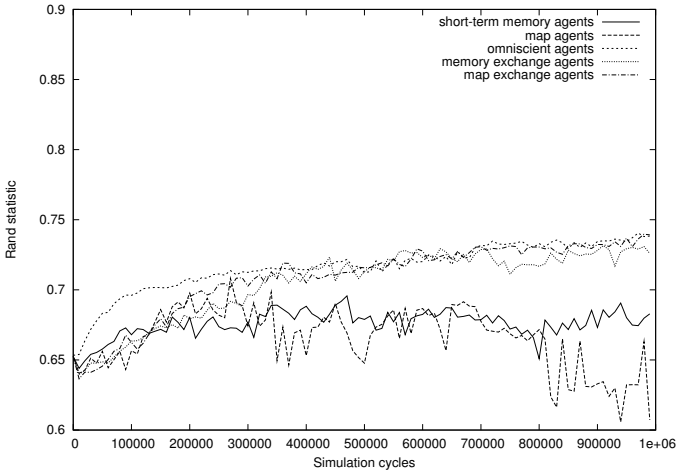
4.4.2 Rand Statistic

The Rand statistic scores over time for both algorithms on the Iris Plant and Wine databases using 10 agents are shown in figure 4.19. With the Iris database, the best score is obtained by the map creating agents algorithm. With the Wine database, the best perform-

ing algorithm is the omniscient agents algorithm, although its performance is equaled by the information exchanging agents algorithms. With the Iris database, the worst is the noncommunicating short-term memory agents algorithm. With the Wine database, the worst is the map creating agents algorithm.



(a) Iris database

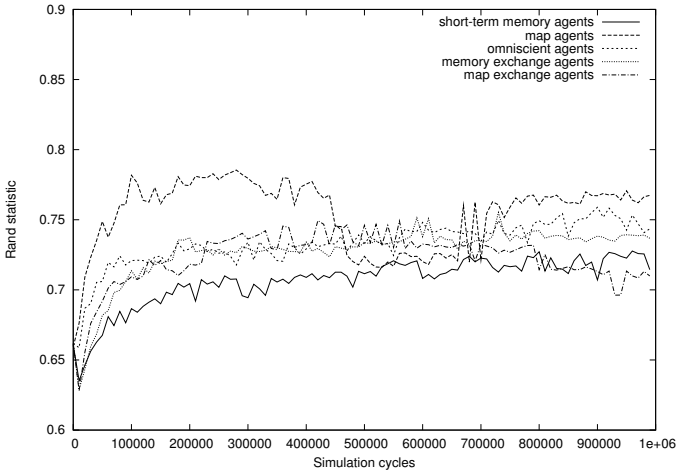


(b) Wine database

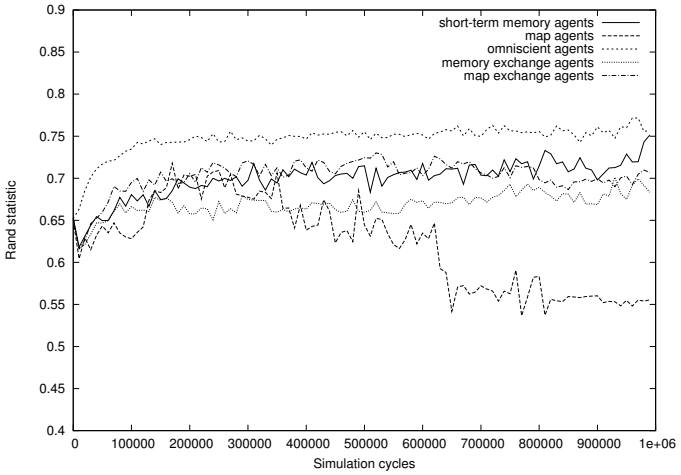
Figure 4.19: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents. A score value of 1 means perfect classification.

In figure 4.20 it can be seen how the performance of the communicating agents algo-

rithms degrades. The map creating agents algorithm is the worst with the Wine database.



(a) Iris database



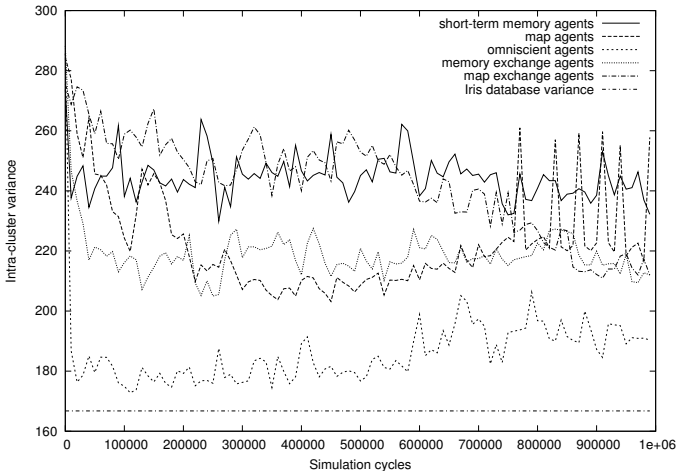
(b) Wine database

Figure 4.20: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents. A score value of 1 means perfect classification.

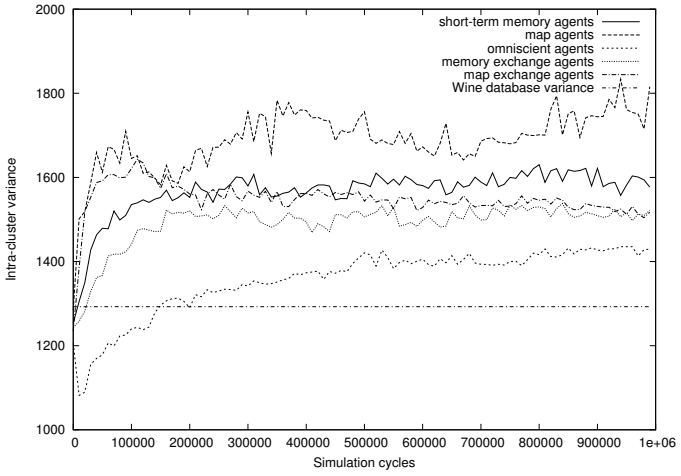
4.4.3 Intra-cluster Variance

The plots seen in figure 4.21 correspond to the intra-cluster variance scores for the tested algorithms with 10 agents on the two test databases. As was the case with the

F-Measure and the Rand statistic, the algorithms where agents can exchange information perform better than the short-term memory algorithm without direct communication among agents. However, they do not perform as well as the omniscient agents algorithm which in both cases is the best performing algorithm. Remember that this measure is to be minimized.



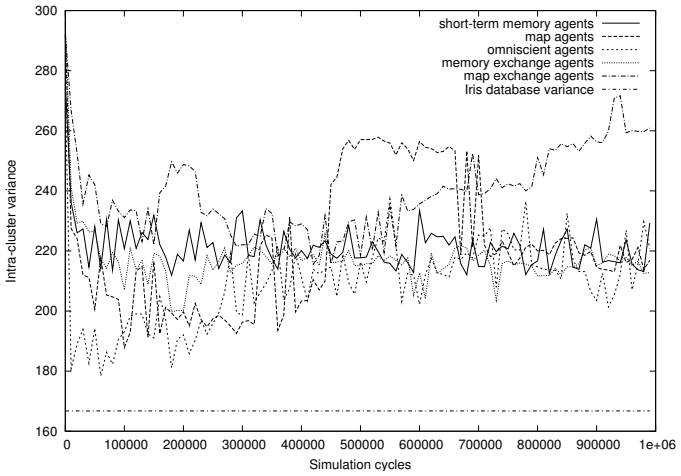
(a) Iris database



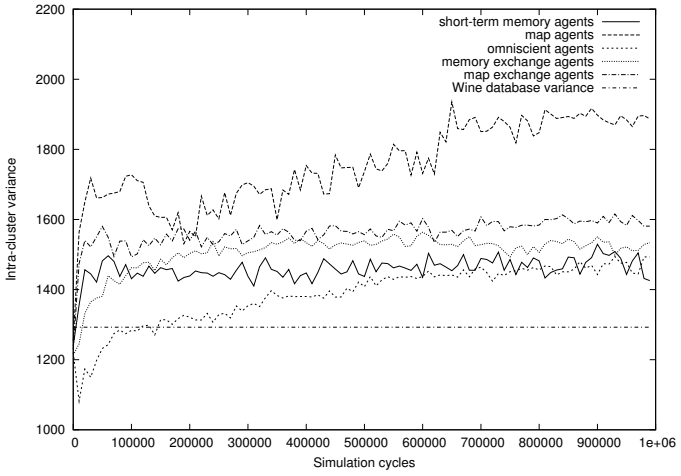
(b) Wine database

Figure 4.21: Total intra-cluster variance over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents. As a reference, the total intra-cluster variance of the correct clustering is shown.

Figure 4.22 shows how the total intra-cluster variance gap between the noncommunicating agents algorithm and the algorithms with information exchange capabilities gets smaller as the number of agents is increased. With both databases, the curves even swap making the information exchange versions the worst.



(a) Iris database



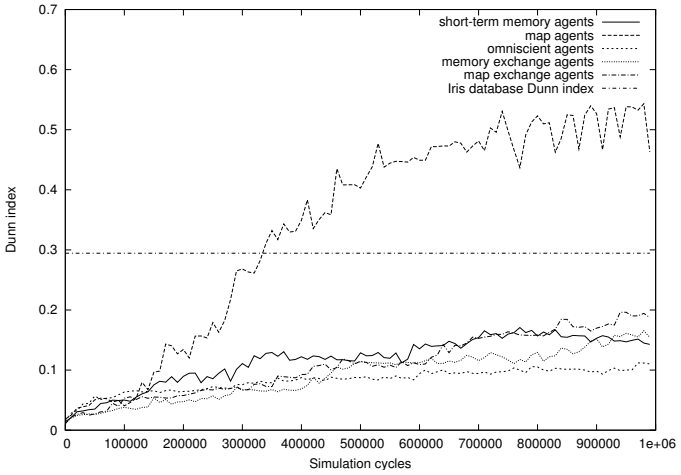
(b) Wine database

Figure 4.22: Total intra-cluster variance over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents. As a reference, the total intra-cluster variance of the correct clustering is shown.

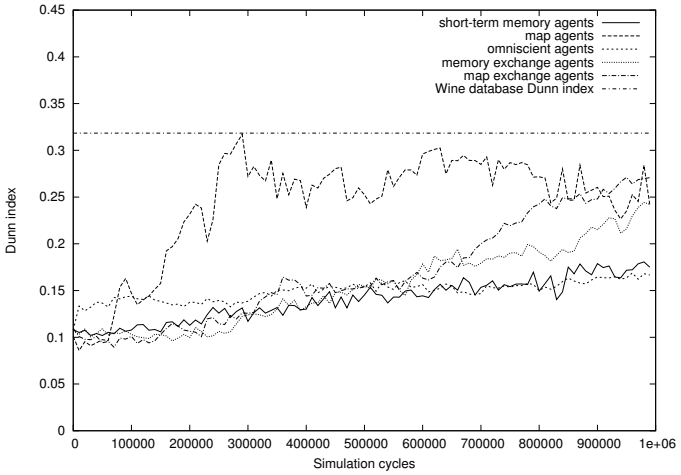
4.4.4 Dunn index

Figure 4.23 shows the Dunn index scores for all tested algorithms with 10 agents on both test databases. On both databases, the best performing algorithm is the map creating agents algorithm. It is followed by the map exchanging agents version, the memory exchange version, the noncommunicating short-term memory agents algorithm, and the omniscient agents algorithm. In that order.

In this case, the performance of the short-term memory agents algorithm without information exchange capabilities is better in the Iris database test and worse in the Wine database. The omniscient agents algorithm is the worst with both databases.



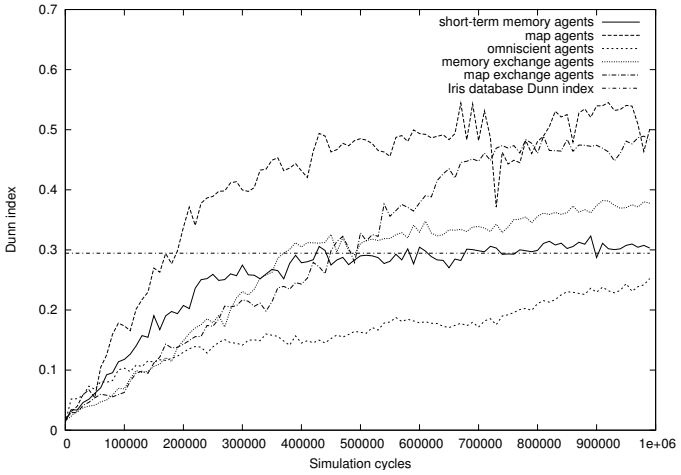
(a) Iris database



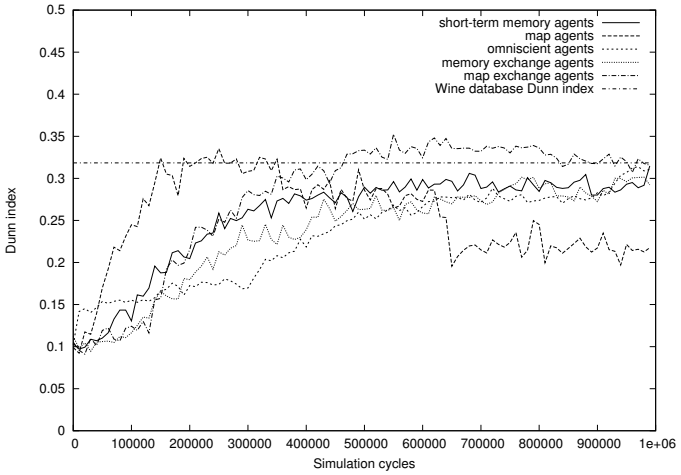
(b) Wine database

Figure 4.23: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents. As a reference, the Dunn index of the correct clustering is shown.

Figure 4.24 shows the Dunn index scores for the tested algorithms using 30 agents. With the Iris database, the relative performance seen before is maintained. With the Wine database, all algorithms score almost the same except the map creating agents algorithm which is the worst.



(a) Iris database

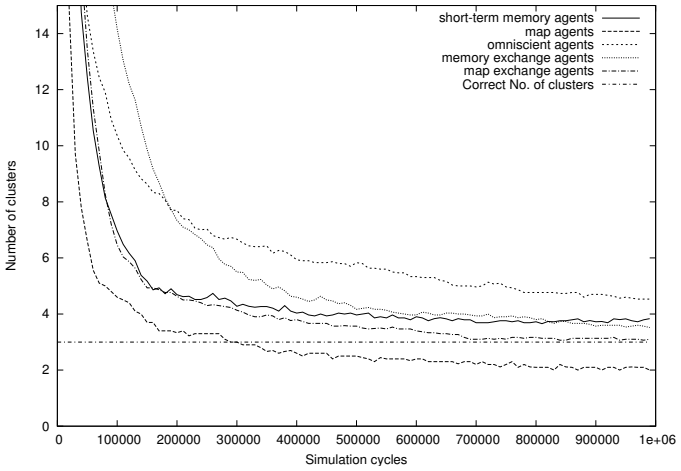


(b) Wine database

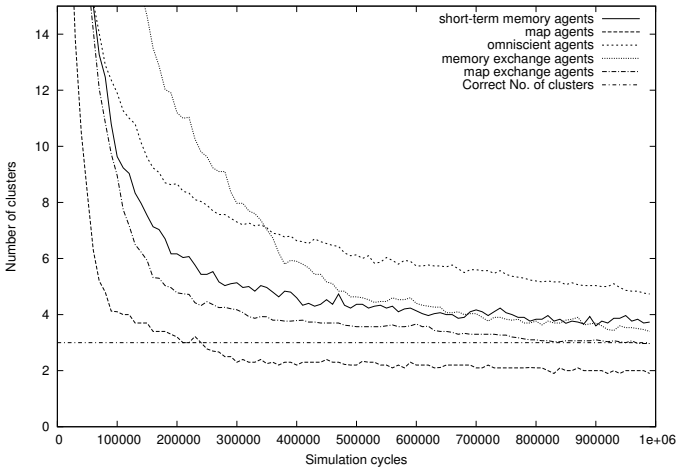
Figure 4.24: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents. As a reference, the Dunn index of the correct clustering is shown.

4.4.5 Number of clusters

Figures 4.25 and 4.26 show the number of clusters discovered by the tested algorithms over time with 10 and 30 agents respectively.

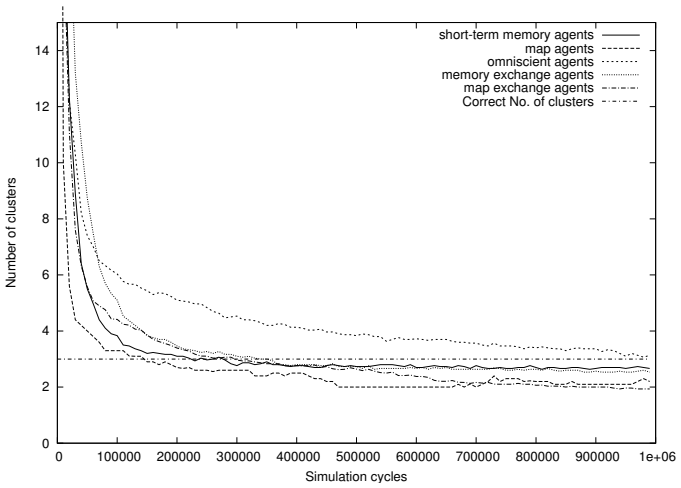


(a) Iris database

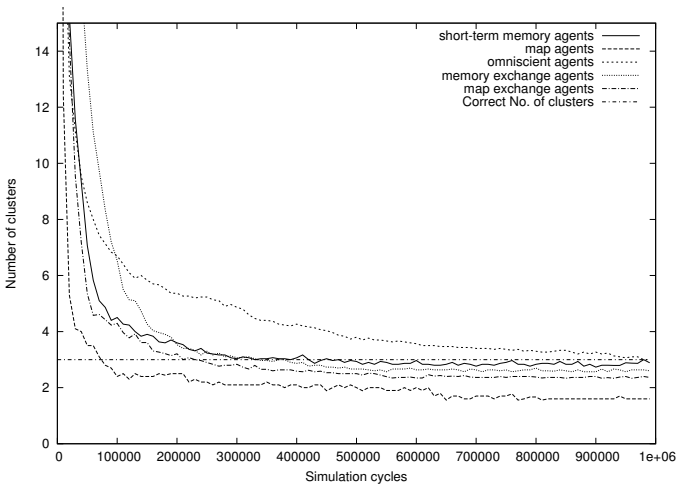


(b) Wine database

Figure 4.25: Number of found clusters over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents. As a reference, the correct number of clusters is shown.



(a) Iris database



(b) Wine database

Figure 4.26: Number of found clusters over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents. As a reference, the correct number of clusters is shown.

Figure 4.26 shows how the number of clusters decreases as the number of agents increases. This confirms the behavior observed in the previous experiment series. In both figures, the algorithm with map-creating agents converges faster than the others.

4.5 Discussion of Results

Two series of experiments were run. In both, agents exchanged directly information about the spatial distribution of data objects on the environment. They only differed in the way agents used that information. In the first series of experiments, agents used the information provided by other agents to update their own environment representations. In these experiments, agents wanted to have an up-to-date environment representation to bias their dropping spot search to the most promising region in the environment. In the second one, loaded agents that already were in search for a dropping spot, used other agents' information to change their search trajectories if the information they received was useful for their immediate purposes.

Let us analyze our results starting with speed issues. It is clear from figures 4.15, 4.16, 4.25, and 4.26 that the omniscient agents algorithm has a slower convergence speed toward a given number of clusters than the simple short-term memory agents algorithm. In other words, having complete knowledge about the spatial distribution of objects does not help in finding the correct number of clusters in a given database faster than with partial knowledge. It is not true, however, that the less information available to the agents, the faster the convergence to the correct number of clusters. This is confirmed by Lumer and Faieta in reference [22] and this very fact was indeed the motivation to modify the basic algorithm in which agents did not have any knowledge about the distribution of objects in the environment.

Among the factors that affected the convergence speed in our experiments is the number of agents. Comparing the obtained results with 10 and 30 agents, it can be seen that the more agents in the algorithm, the faster the convergence speed. Another important aspect to note is that the number of found clusters is also altered, with fewer clusters when using more agents.

In the first series of experiments, the slowest algorithm was the one with memory updating agents, and the fastest was the map updating agents algorithm. The explanation of this phenomenon is related with how accurate agents could approximate the environment state. More on this later. In the second series of experiments, since agents did not try to approximate the environment state, the behavior of the communicating agents algorithms was completely different. The algorithm with map creating agents was again the fastest, and even though during the first cycles the convergence speed of the memory exchanging agents algorithm is slower than that of the other algorithms, it is the second fastest in the end.

In terms of clustering quality, the omniscient agents algorithm is better than the short-term memory agents algorithm, except in figures 4.13, 4.14, 4.23, and 4.24 that show the Dunn index scores. However, as was pointed out earlier, having a high Dunn index score does not necessarily implies a good clustering. In fact, if we recall that in the Iris database two clusters are not linearly separable, a high Dunn index may be an indication of a confusion, i.e., the algorithm considering two or more clusters as just one. This clearly increases the

measure that tries to identify compact and well separated clusters.

The memory updating agents algorithm approximates the omniscient agents algorithm and that is reflected by the shapes of their response functions. The main difference between them is an (often negative) offset. However, it is an indication of the approximation task it was supposed to perform. We cannot say the same for the map updating agents algorithm, because in all the experiments it is the worst performing algorithm. Future work will have to focus on this issue, since only one parameter set was used, and it surely affected the obtained performance.

Something important to note is that when agents use information for updating their own environment representations, the overall performance is quite bad, and does not offer any advantage over the omniscient agents algorithm nor the simple short-term memory agents algorithm. However, if they only used the information to select the best possible dropping location, as was done in the second series of experiments, the overall performance is improved. This performance improvement takes place only when using 10 agents, because with 30 agents, the performance drops again.

4.6 Conclusions

The goal of letting agents in ant-based clustering algorithms communicate, is to allow them share knowledge about the spatial distribution of data objects on the environment to create better clusters than when they cannot.

In the problem of communication, besides the way agents can exchange information, there are other participating variables such as *what* to exchange, *when* to do it, or *what to do* with new information. The first issue, that is, the way in which agents can exchange information was partially studied in this chapter and will be extended in the next one, this chapter dealt with direct information exchange among agents, the next one will deal with indirect information exchange. The second issue can be studied if we consider that agents individually maintain a model of the environment and that the exchanged information will serve to enrich or update that particular representation. This was done first, by using an extension of the basic short-term memory agents algorithm proposed by Lumer and Faieta, and then using agents that create maps of their environment as they explore it. The third issue, that is, the problem of establishing the best possible moment to exchange information, was solved by going back to the natural inspiration of this technique. In addition to the indirect communication through pheromones, real social insects also interact directly. So, to choose the moment when they meet on the grid as the right moment to exchange information, seemed the most natural and intuitively appealing solution to this problem. The fourth issue, the what to do with the information from other agents, was explored through two different strategies. In the first one, agents enriched and updated their environment representations with the hope that they approximate the real state of the environment. Once an agent had picked up an object from the environment, using its environment representation, it could

decide to go to the best possible region on which to drop that object. In the second strategy, agents that had already picked up an object, could "change their minds" about the best dropping spot based on other agents' information.

The results obtained in the first series of experiments, those with agents updating their environment representations, are somewhat discouraging. The worst performing algorithm is the one with map updating agents, and the second worst is the memory updating agents algorithm. In the first case, further experimentation must be done to know what went wrong in the experiments. It is very likely that the parameter set used in the experiments was not the correct one. We leave this issue as future work. In the case of the memory updating agents algorithm, the relatively poor performance is not as discouraging as is the obtained with map creating agents. The negative offset observed in the response plots with respect to the omniscient agents algorithm response plots, is an indication of the correct, although not perfect, approximation to the real state of the environment.

The results obtained with dropping spot search trajectory changing agents are quite the opposite from the previous ones. In this case, the clustering quality is improved by the communicating agents algorithms. They produce even better clusters than the version with omniscient agents. This is true only when using 10 agents. With 30 agents, the relative performance of these algorithms is deteriorated. This suggests the existence of some critical number of agents that should behave this way with a given database. Determining the optimum number of agents for a given clustering task remains a research issue.

Chapter 5

Intentional indirect communication among agents in ant-based clustering algorithms

This chapter presents the effects on the quality of the clustering obtained by using intentional indirect communication among agents in ant-based clustering algorithms. Agents lay packets which contain information about data distribution on the environment for others to pick and use. Although agents modify the environment and respond accordingly, this kind of communication differs from a purely stigmergic one in that this is *intentional*. An agent lays information packets for others to exploit, it is not just a result of its own actions.

As with the experiments with direct communication, agents use the information packets to update their environment representations and to bias their search for dropping locations. In the first two series of experiments, agents lay information packets with a certain frequency. A second laying strategy is also explored in which agents lay information only when they have modified the environment.

5.1 Indirect communication strategies

In the previous chapter information exchange among agents occurs only when two or more agents coincide on the grid. Two different agent population sizes were tried to explore the effects of increasing the number of exchanges during a simulation run since the number of encounters rises as the agent density in the grid increases. This strategy has two disadvantages: (i) even when the number of exchanges increases, we cannot expect many of them to happen since the number of agents must be kept small (for performance reasons), and (ii) many exchanges do not have any effect since agents walk in a randomly fashion, i.e., two agents coincide many times, over and over again, before they follow different trajectories.

If we let agents lay information on their environment, it could be possible to increase dramatically the number of exchanges without even increasing the number of agents. In this chapter this idea is explored. The information packets laid on the environment in the form of memory vectors or growing neural gas networks, are *references* to the original structures inside the agents that laid them. In other words, information packets always have updated information and therefore, they always give trustworthy information to their consumers. The

only parameter we have available to experiment with is the laying frequency.

As with the experiments of the previous chapter, the information is used in two different ways; to update the agents' representation of the environment and to bias their random walk in search for good dropping locations. The output quality is compared with the one obtained by the omniscient agents algorithm and the noncommunicating agents algorithms.

5.2 Experimental Setup

The test databases that were used for the experiments with direct communication, were also used for the experiments with indirect communication. The agent models, environment state representations and updating operations were also the same. See chapter 4, section 4.2 for details.

5.2.1 Experiments Design

The same experimental strategy used in the direct communication experiments was used for the experiments with indirect communication, except for the frequency laying parameter. The experiments explore the effects of letting agents lay information packets every 10, 100, and 1000 cycles. As before, the experiments were run with 10 and 30 agents. These experiments correspond to a periodic information laying policy. We conducted another series of experiments with an adaptive information laying policy in which agents lay information packets only after they have just modified the environment and a given delay has elapsed. In this case, three laying delays were tested: 1, 10 and 100 cycles.

5.3 Results of Experiments on Information Exchange for Environment Representation Updates using a Periodic Laying Policy

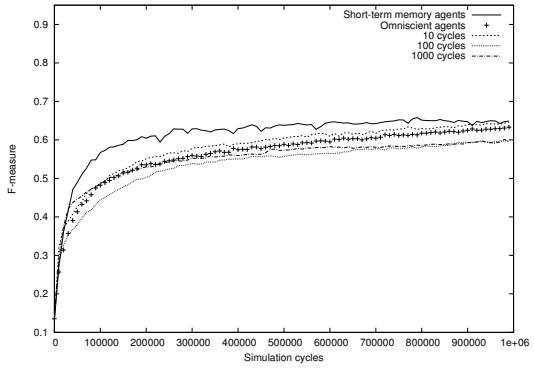
Five different algorithms were tested[1], see chapter 4, section 4.2 for details. In the experiments run in this section, the update or change in a search trajectory, is done whenever an agent comes across an information packet on the environment. The results are presented by validity measure.

5.3.1 F-Measure

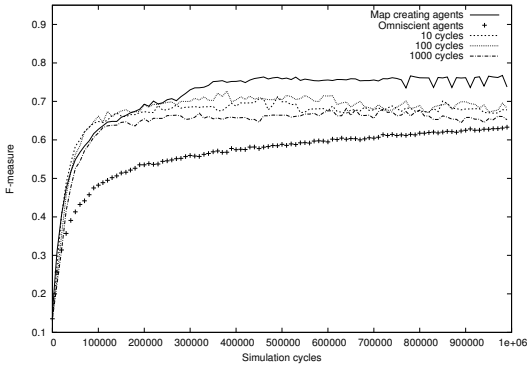
Figure 5.1 shows the F-Measure scores over time obtained by the tested algorithms on the Iris Plant and Wine databases using 10 agents. With the short-term memory model,

[1]Although strictly speaking they are only three different algorithms, using a different frequency alters importantly the behavior of the algorithm, so we will treat them as if they were in fact distinct algorithms.

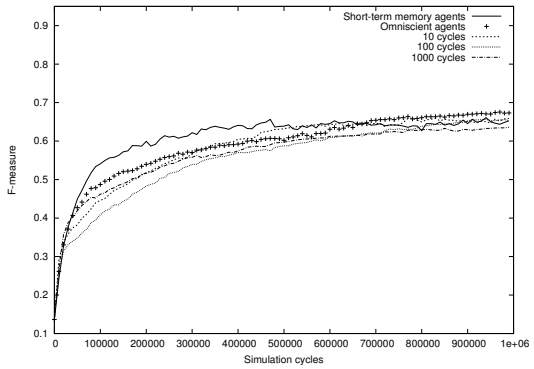
the algorithm in which agents have a short-term memory and do not communicate with each other obtains the highest score with both test databases. However, near the end of the simulation all algorithms perform almost equally well. With the growing neural gas model, i.e., with the self-organizing map as a representation tool, the best performing algorithm is the simple algorithm with noncommunicating agents. With the Wine database it is clear the effect of using different laying frequencies, being the algorithm with a frequency of 10 cycles the best and the one with a 1000 cycles the worst.



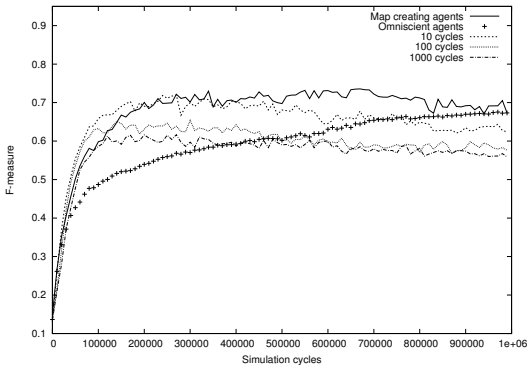
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

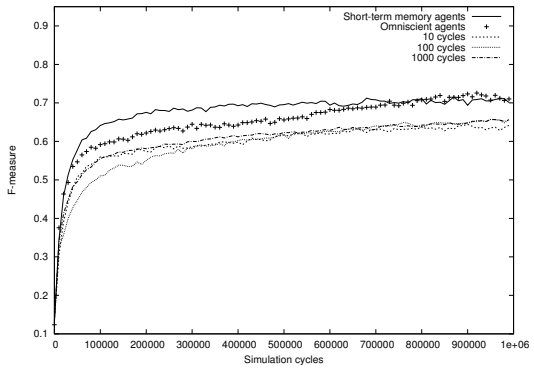


(d) Growing Neural Gas model on Wine database

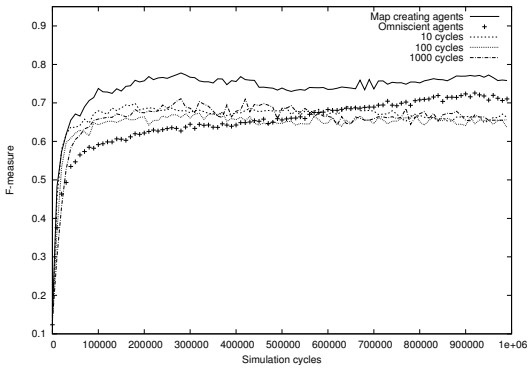
Figure 5.1: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using a periodic laying policy. A score value of 1 means perfect classification.

Figure 5.2 shows the F-Measure scores over time obtained by the tested algorithms on the Iris Plant and Wine databases using 30 agents. In all four graphs, it can be seen how the

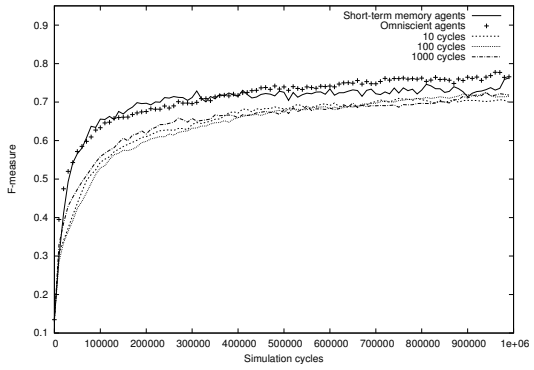
algorithm with noncommunicating agents performs better than the others at the beginning of the simulation. In the end, the best algorithm is the omniscient agents algorithm. The performance of the representation updating algorithms degrades with 30 agents.



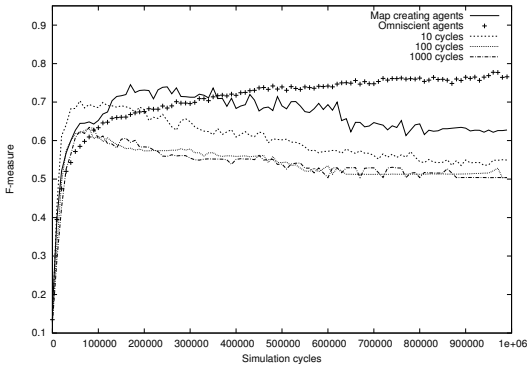
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

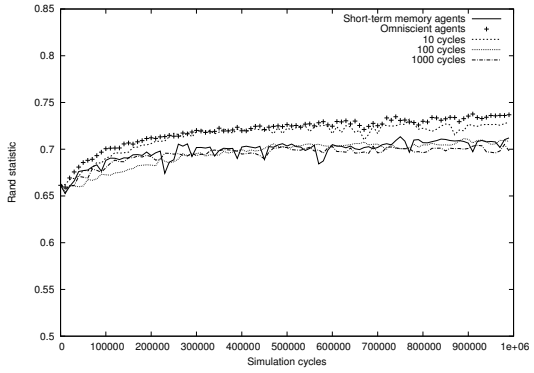


(d) Growing Neural Gas model on Wine database

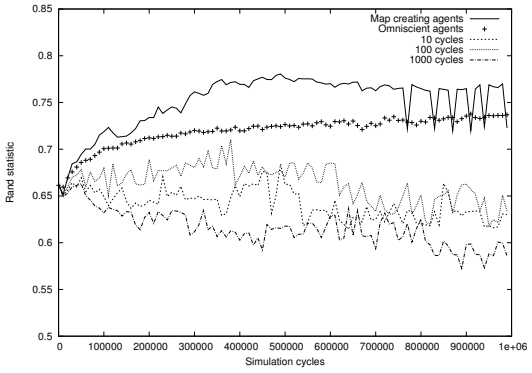
Figure 5.2: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using a periodic laying policy. A score value of 1 means perfect classification.

5.3.2 Rand Statistic

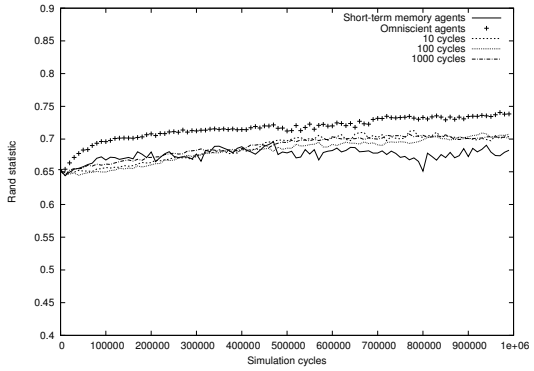
The Rand statistic scores over time for all tested algorithms on the Iris Plant and Wine databases using 10 agents using a periodic laying policy are shown in figure 5.3.



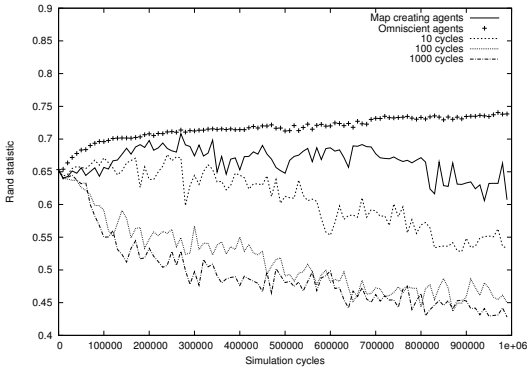
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

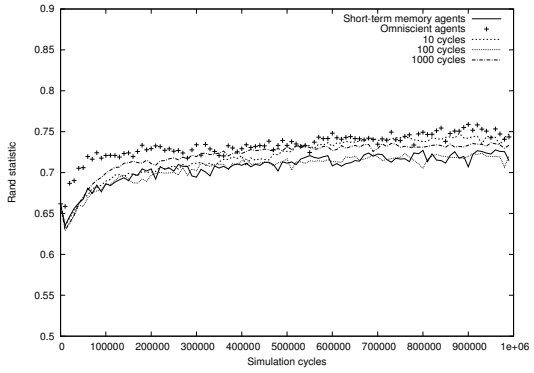


(d) Growing Neural Gas model on Wine database

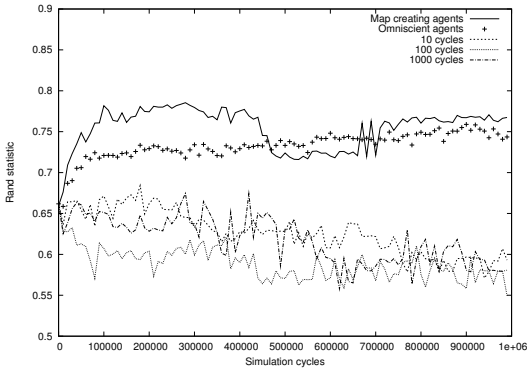
Figure 5.3: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using a periodic laying policy. A score value of 1 means perfect classification.

With the short-term memory model, the best performing algorithm is the one with omniscient agents. All other algorithms perform almost the same. With the growing neural gas model, with the Iris database, the best algorithm is the simple map creating agents algorithm and with the Wine database, the best algorithm is again the omniscient agents algorithm. Again it is clear how the higher the laying frequency the better when using 10 agents and with the Wine database.

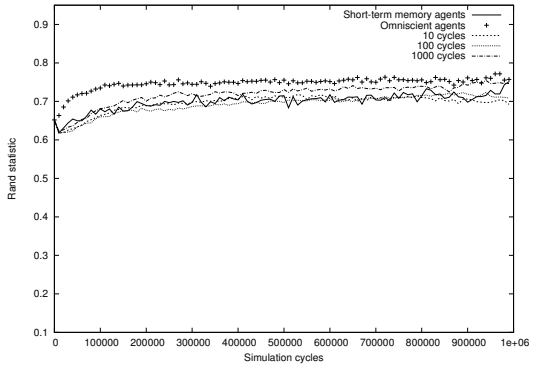
Figure 5.4 shows the Rand statistic scores with 30 agents using a periodic information laying policy. The behavior pattern observed in figure 5.3 is also observed in this figure.



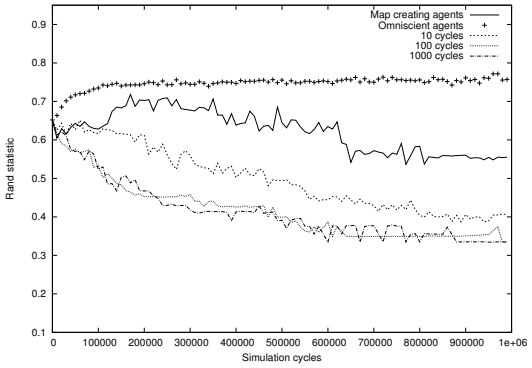
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

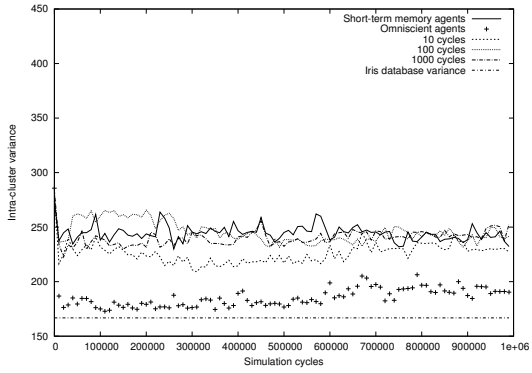


(d) Growing Neural Gas model on Wine database

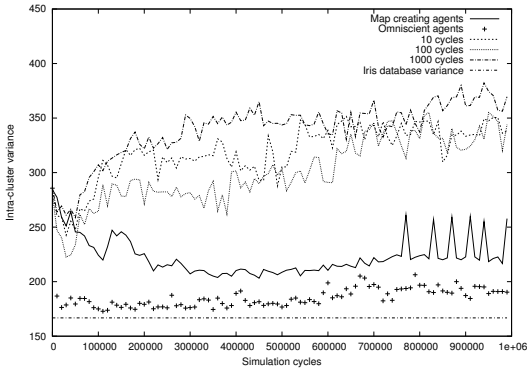
Figure 5.4: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using a periodic laying policy. A score value of 1 means perfect classification.

5.3.3 Intra-cluster Variance

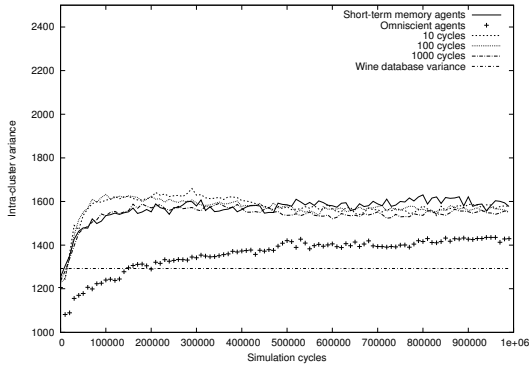
The plots seen in figure 5.5 correspond to the intra-cluster variance scores for all the tested algorithms with 10 agents using a periodic laying policy.



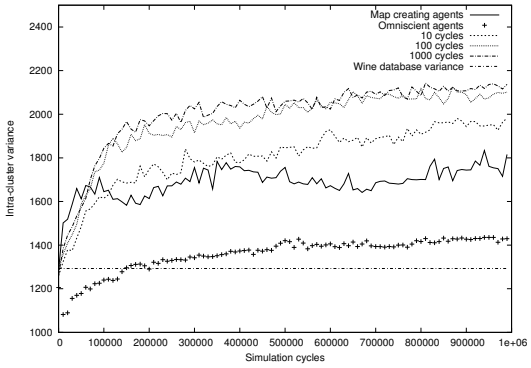
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

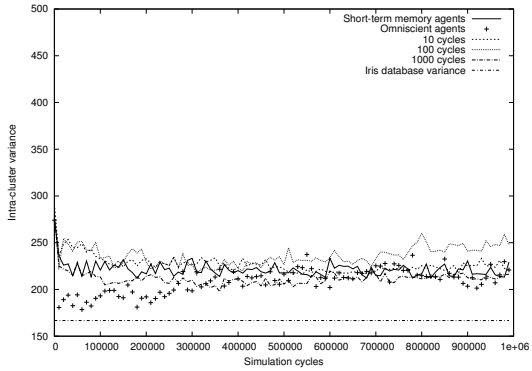


(d) Growing Neural Gas model on Wine database

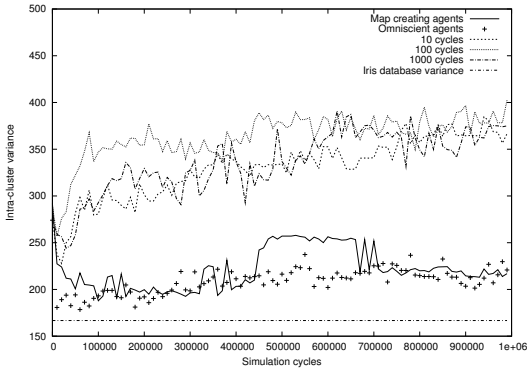
Figure 5.5: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using a periodic laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.

In all four graphs the best performing algorithm is the omniscient agents algorithm. With the both databases, using the short-term memory model, all other four algorithms perform the same. With the growing neural gas model, the omniscient agents algorithm is followed by the simple noncommunicating agents algorithm. The other three algorithms follow the pattern observed before, i.e., a frequency of 10 cycles is better than 100 and 1000 cycles.

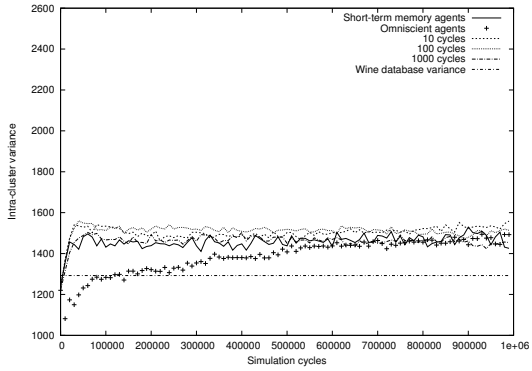
As before, in figure 5.6 we can see the same behavior pattern.



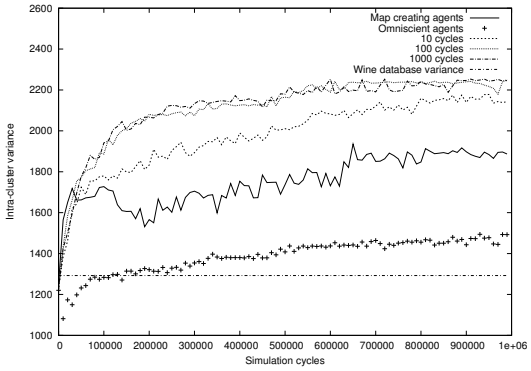
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

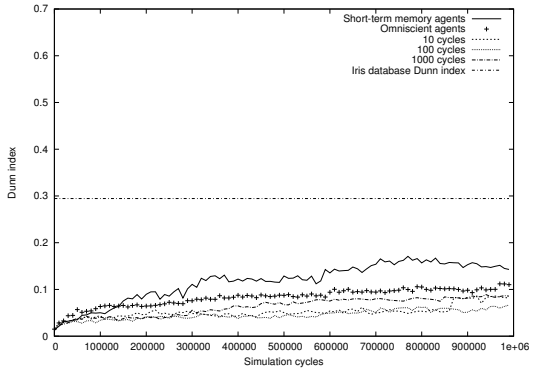


(d) Growing Neural Gas model on Wine database

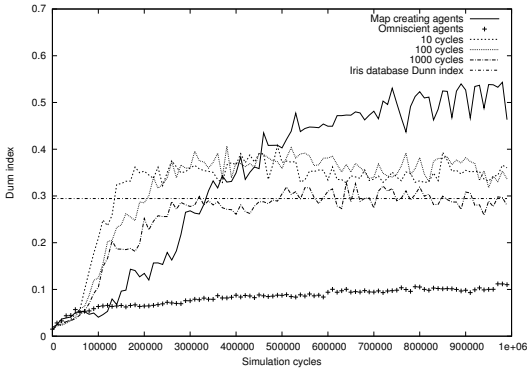
Figure 5.6: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using a periodic laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.

5.3.4 Dunn index

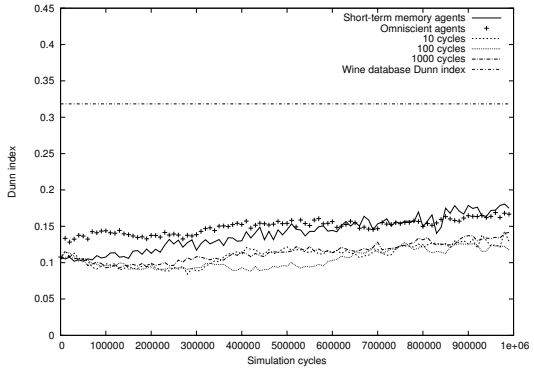
Figure 5.7 shows the Dunn index scores for all the tested algorithms with 10 agents on the test databases.



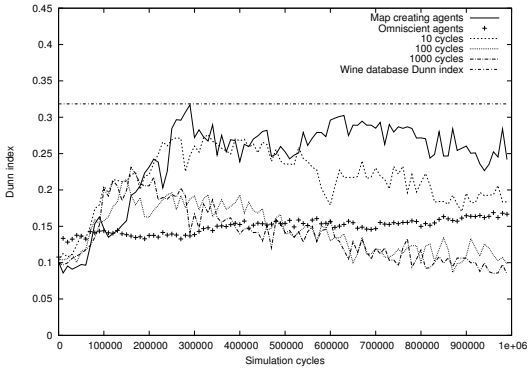
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



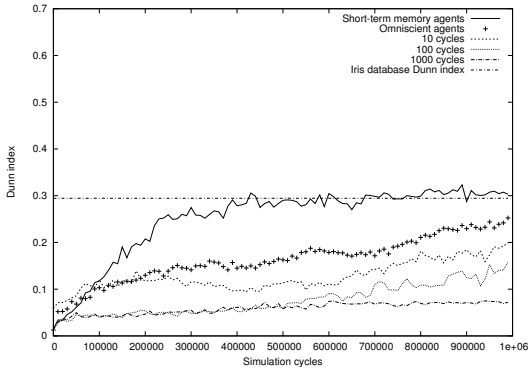
(d) Growing Neural Gas model on Wine database

Figure 5.7: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using a periodic laying policy. As a reference, the Dunn index of the correct clustering is shown.

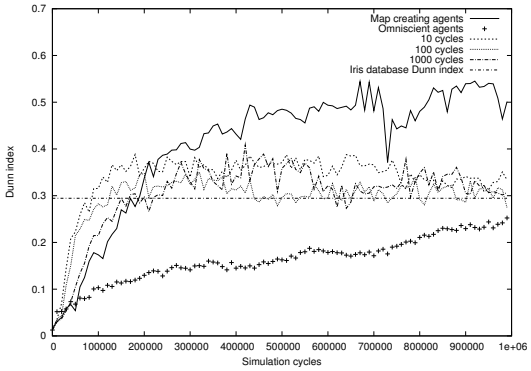
With the short-term memory model, all algorithms perform almost the same. although its clear that the best performing algorithm in this case is the simple noncommunicating agents algorithm. With the growing neural gas model, the best performing algorithm is the simple noncommunicating agents algorithm. In this case, with the Iris database, the behavior pattern exhibited by the communicating agents algorithms shows up again. Notice the difference in their absolute performance during the first 200,000 cycles.

Figure 5.8 shows the Dunn index scores for the tested algorithms using 30 agents. With the short-term memory model and the both databases, the same behavior pattern reappears. With the growing neural gas model, the communicating agents algorithms perform terribly

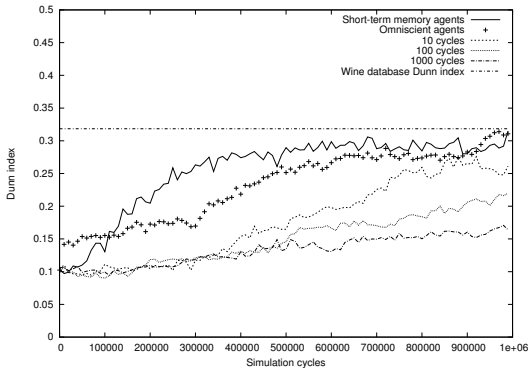
bad with the Wine database.



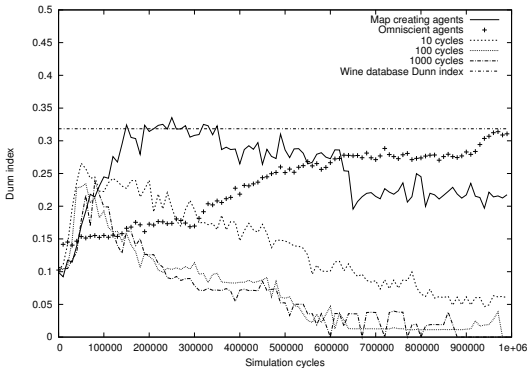
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

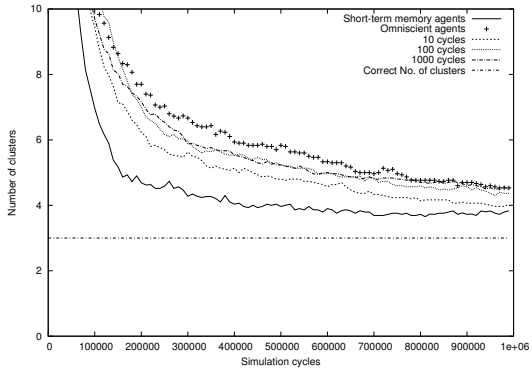


(d) Growing Neural Gas model on Wine database

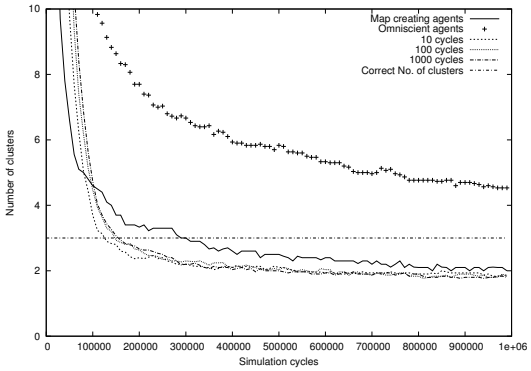
Figure 5.8: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using a periodic laying policy. As a reference, the Dunn index of the correct clustering is shown.

5.3.5 Number of clusters

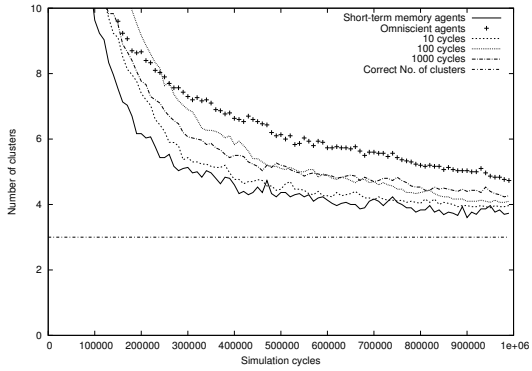
Figures 5.9 and 5.10 show the number of clusters discovered by the tested algorithms over time with 10 and 30 agents respectively.



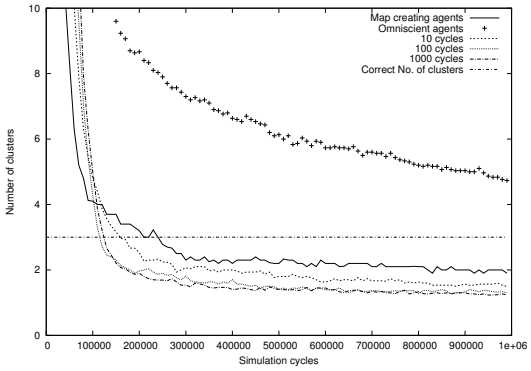
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

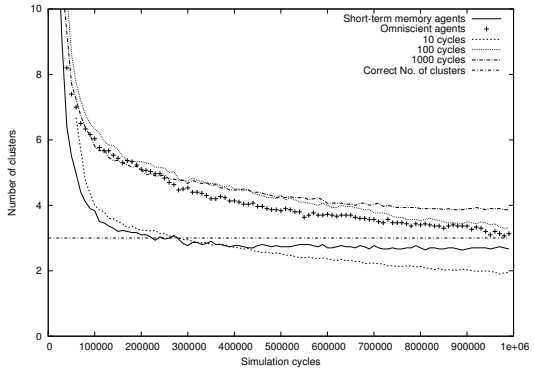


(c) Short-term memory model on Wine database

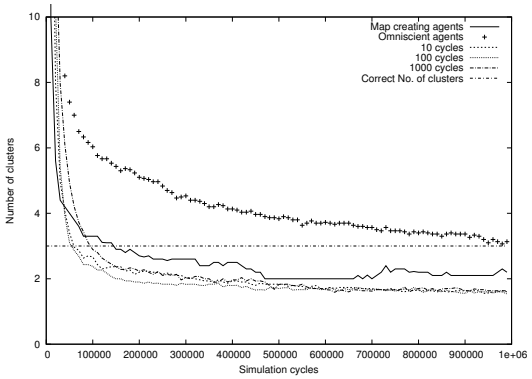


(d) Growing Neural Gas model on Wine database

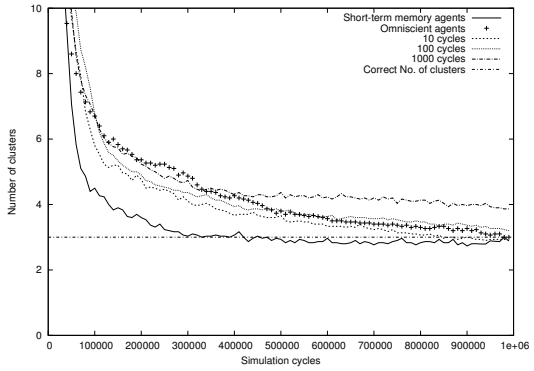
Figure 5.9: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using a periodic laying policy. As a reference, the correct number of clusters is shown.



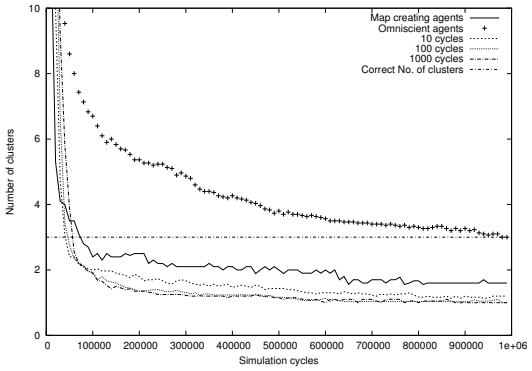
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



(d) Growing Neural Gas model on Wine database

Figure 5.10: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using a periodic laying policy. As a reference, the correct number of clusters is shown.

In all eight graphs, it is clear that the omniscient agents algorithm converges slower than any simple noncommunicating agents algorithm. The communicating agents algorithms converge faster than the omniscient agents algorithm. In some cases they converge to just one cluster.

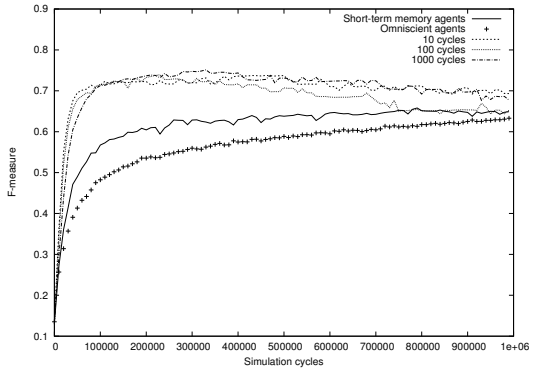
5.4 Results of Experiments on Information Exchange for Behavior Changes using a Periodic Laying Policy

This section presents the results of the ant-based clustering algorithms using agents that change their dropping spot search trajectories based on the information they come across in the environment.

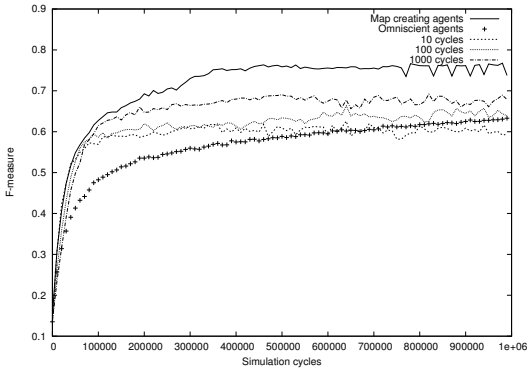
5.4.1 F-Measure

In figure 5.11 we can see the F-Measure scores over time for all tested algorithms on the Iris Plant and the Wine databases using 10 agents. With the short-term memory model the communicating agents algorithms perform better than the simple noncommunicating agents and the omniscient agents algorithms. With the growing neural gas model, the best performing algorithm is the noncommunicating agents algorithm. In this case, the best performing communicating agents algorithm is the one using a laying frequency of 1000 cycles. The worst of them is the one using 10 cycles.

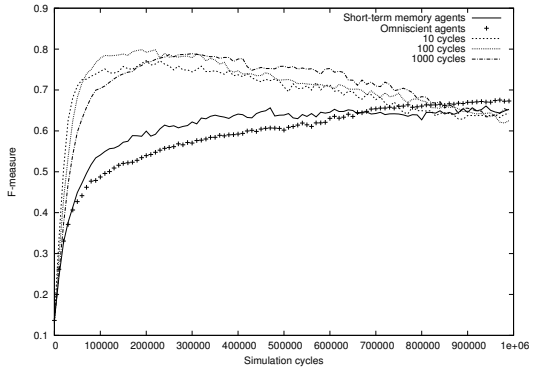
Figure 5.12 shows the results obtained using 30 agents. With the short-term memory model, the performance of all three communicating agents algorithms degrades. The same is true with the growing neural gas model; however, the relative difference among their performance is maintained.



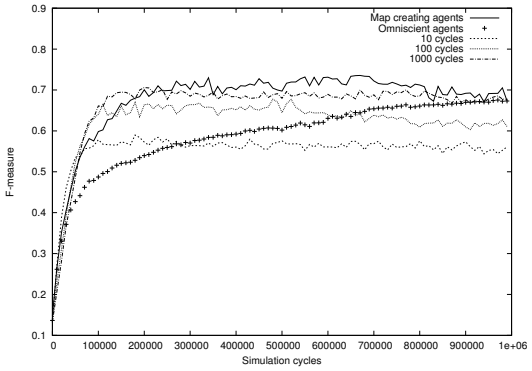
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

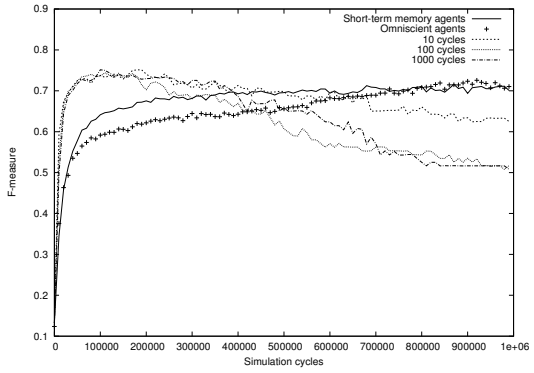


(c) Short-term memory model on Wine database

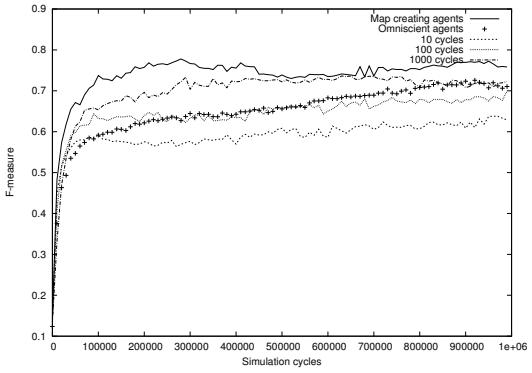


(d) Growing Neural Gas model on Wine database

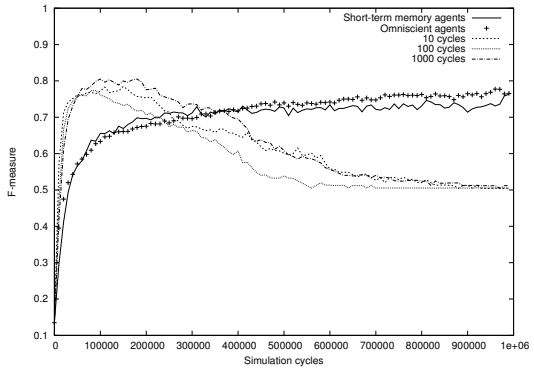
Figure 5.11: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using a periodic laying policy. A score value of 1 means perfect classification.



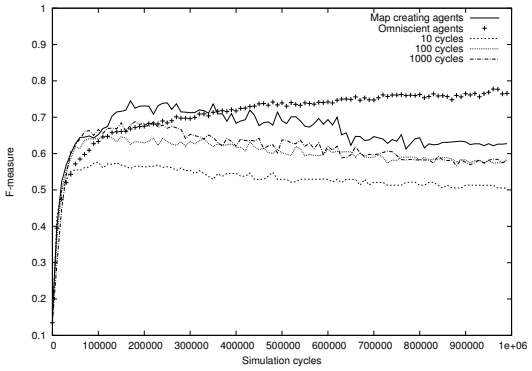
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

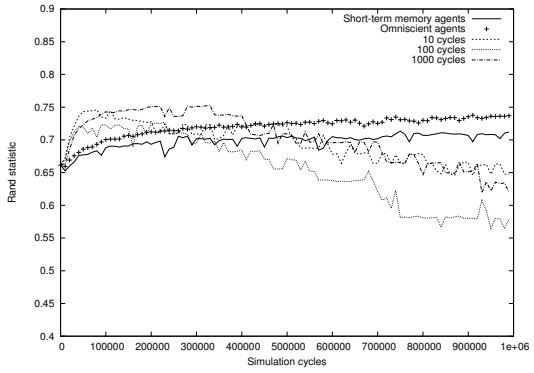


(d) Growing Neural Gas model on Wine database

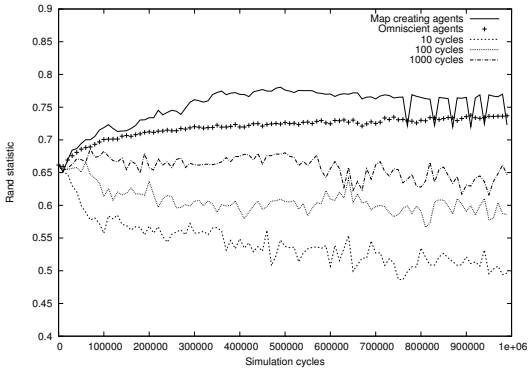
Figure 5.12: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using a periodic laying policy. A score value of 1 means perfect classification.

5.4.2 Rand Statistic

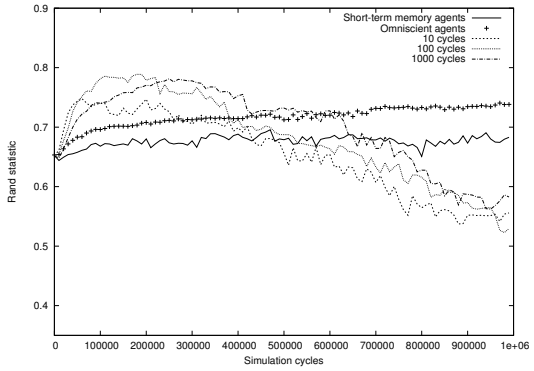
Figure 5.13 shows the Rand statistic scores over time using 10 agents following a periodic information laying policy. With the short-term memory model, the communicating agents algorithms perform terribly bad. At the end of the simulation the omniscient agents algorithm becomes the best performing algorithm. With the growing neural gas model, the communicating agents algorithms show a clear pattern: the more frequently they lay information on the environment the worse.



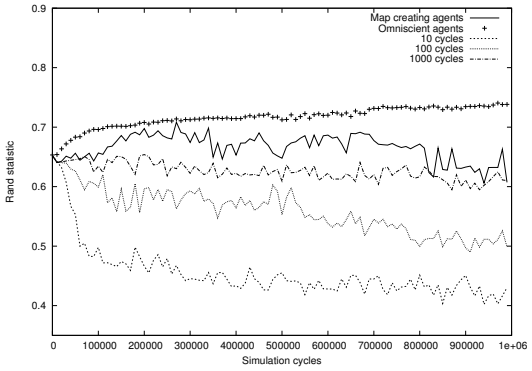
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



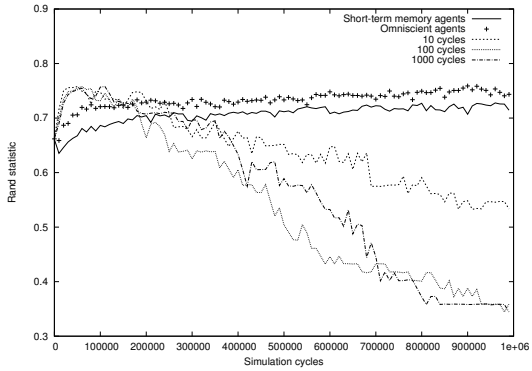
(c) Short-term memory model on Wine database



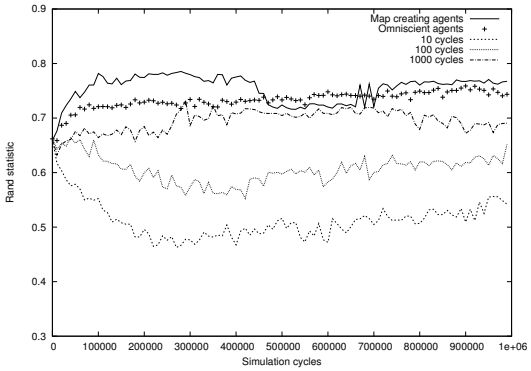
(d) Growing Neural Gas model on Wine database

Figure 5.13: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using a periodic laying policy. A score value of 1 means perfect classification.

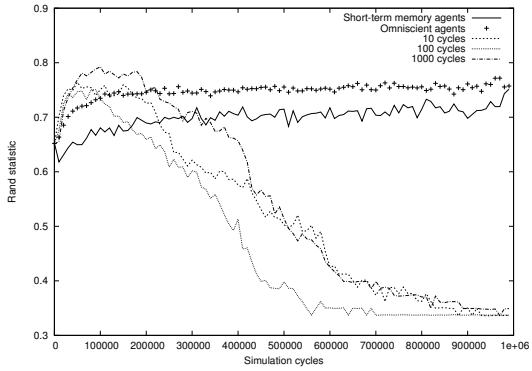
In figure 5.14 the behavior pattern just described shows again. In this case using 30 agents.



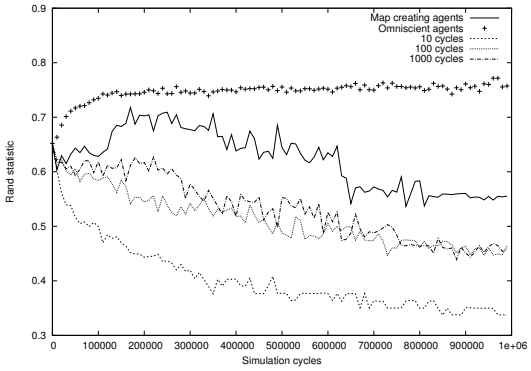
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

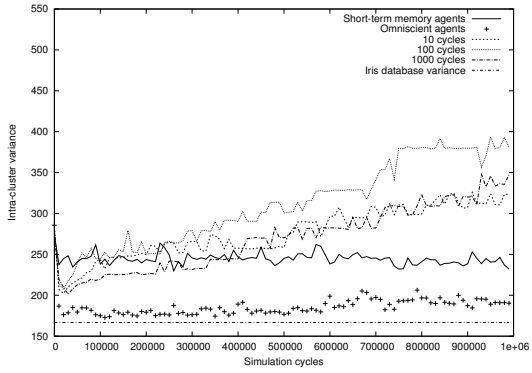


(d) Growing Neural Gas model on Wine database

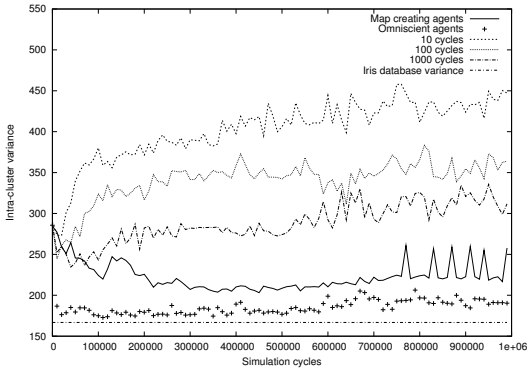
Figure 5.14: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using a periodic laying policy. A score value of 1 means perfect classification.

5.4.3 Intra-cluster Variance

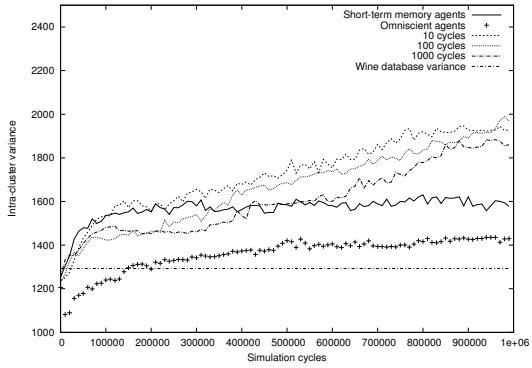
In figures 5.15 and 5.16 we can see the total intra-cluster variance scores over time for all tested algorithms on the Iris Plant and the Wine databases using 10 and 30 agents respectively. The best performing algorithm is the omniscient agents algorithm, followed by the noncommunicating agents algorithm in every case. The algorithm in which agents lay information on the grid every 10 cycles is the worst of all.



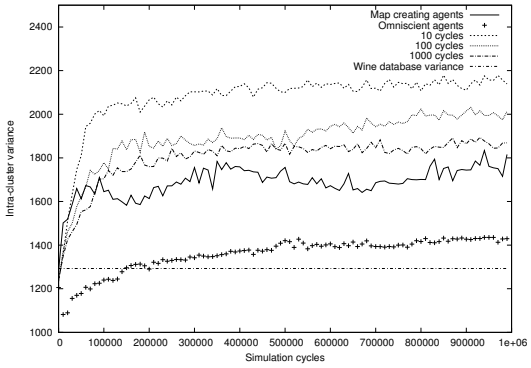
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

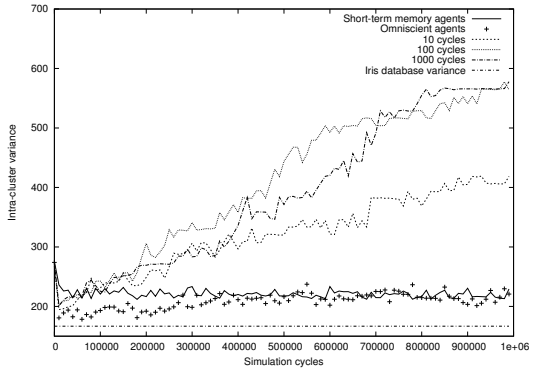


(c) Short-term memory model on Wine database

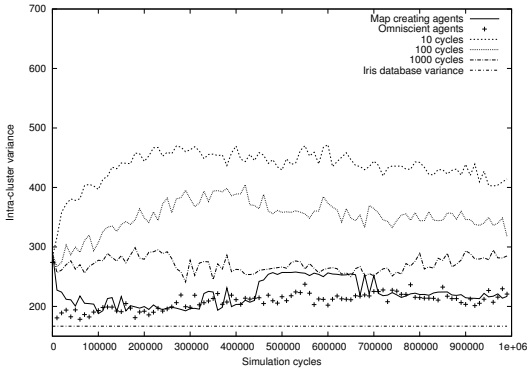


(d) Growing Neural Gas model on Wine database

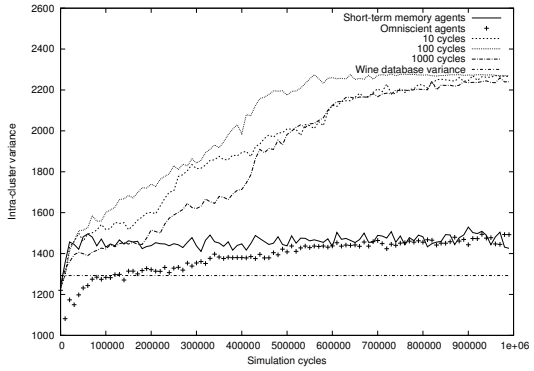
Figure 5.15: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using a periodic laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.



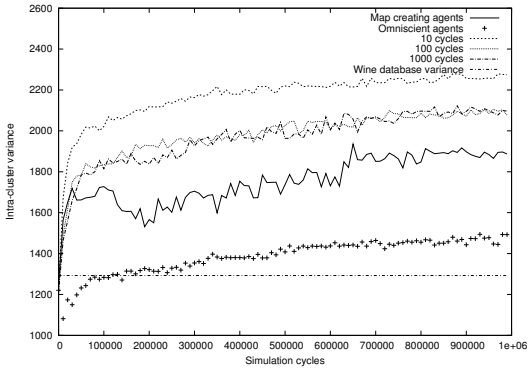
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

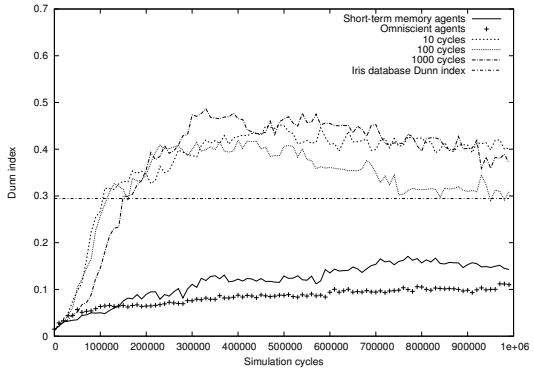


(d) Growing Neural Gas model on Wine database

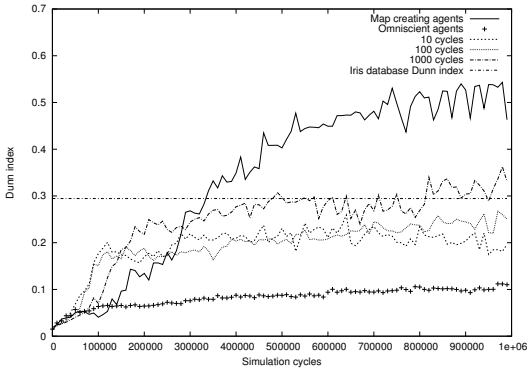
Figure 5.16: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using a periodic laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.

5.4.4 Dunn index

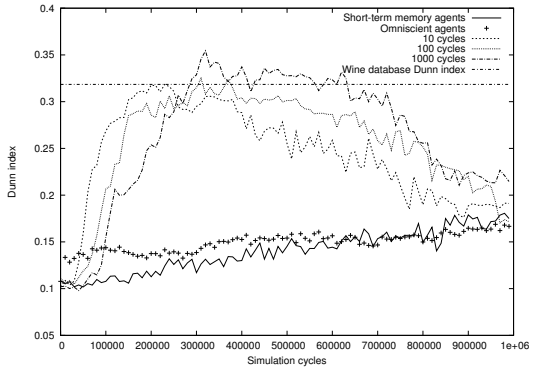
Figure 5.17 shows the Dunn index scores obtained by the tested algorithms with 10 agents. With the short-term memory model, the communicating agents algorithms are the best. With the Wine database, they reach a maximum and start decreasing. With the growing neural gas model and with the Iris database, they perform better than the omniscient agents algorithm but worse than the noncommunicating agents algorithm.



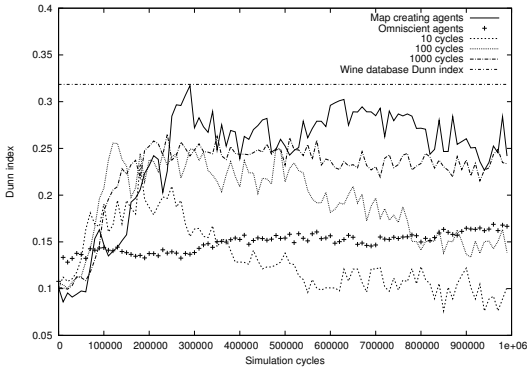
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



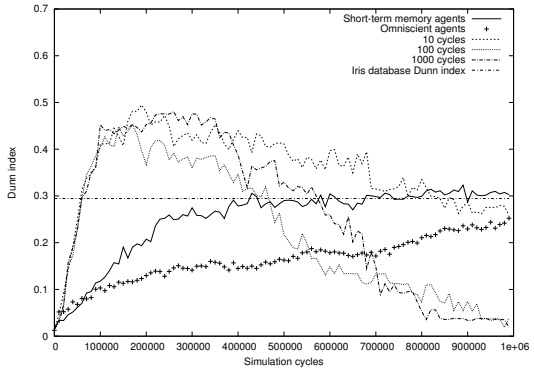
(c) Short-term memory model on Wine database



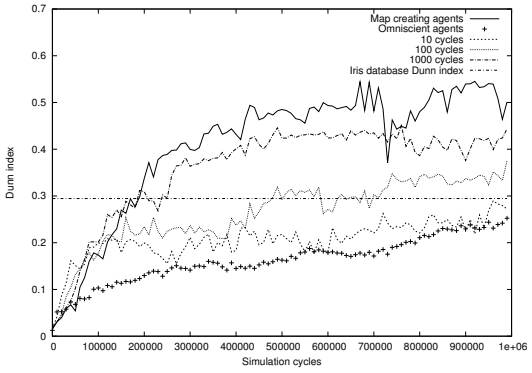
(d) Growing Neural Gas model on Wine database

Figure 5.17: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using a periodic laying policy. As a reference, the Dunn index of the correct clustering is shown.

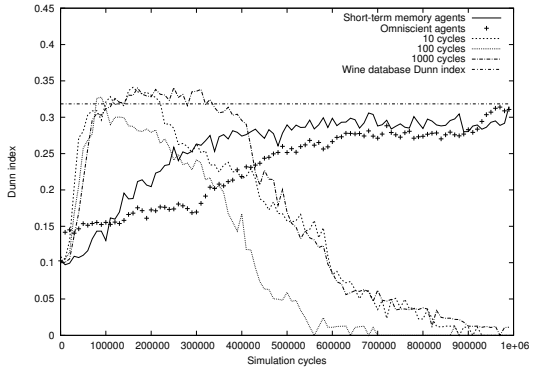
In figure 5.18 the Dunn index scores obtained by the tested algorithms with 30 agents are shown. In all cases except one, the communicating agents algorithms start performing quite well. However, in the end they converge to a value near zero.



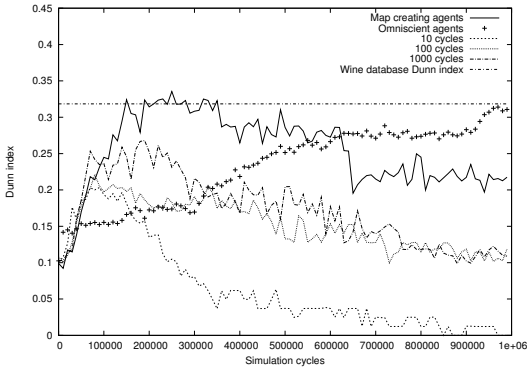
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

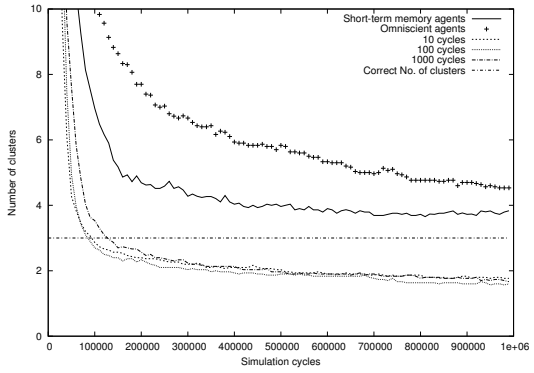


(d) Growing Neural Gas model on Wine database

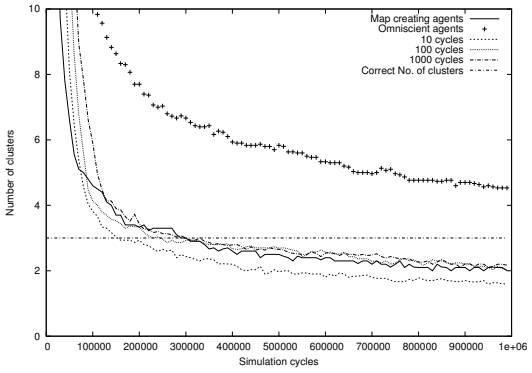
Figure 5.18: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using a periodic laying policy. As a reference, the Dunn index of the correct clustering is shown.

5.4.5 Number of clusters

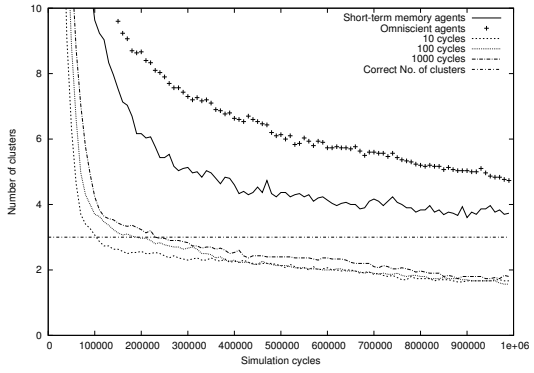
Figures 5.19 and 5.20 show the number of clusters discovered by the tested algorithms over time with 10 and 30 agents respectively.



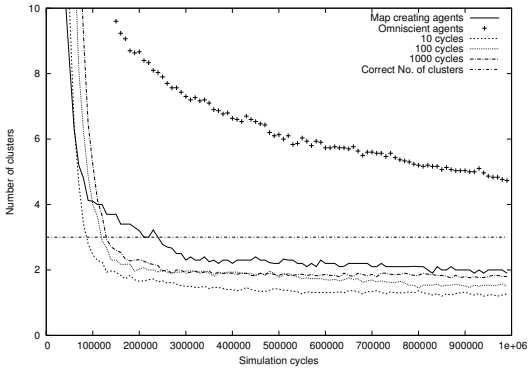
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



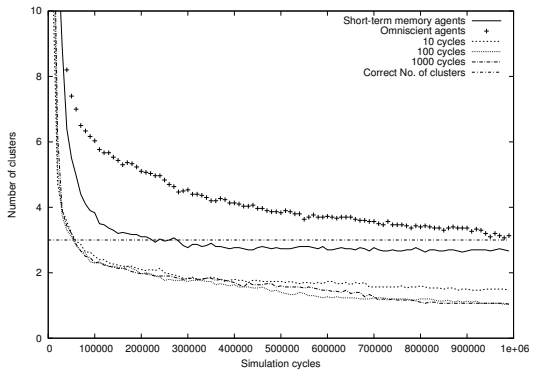
(c) Short-term memory model on Wine database



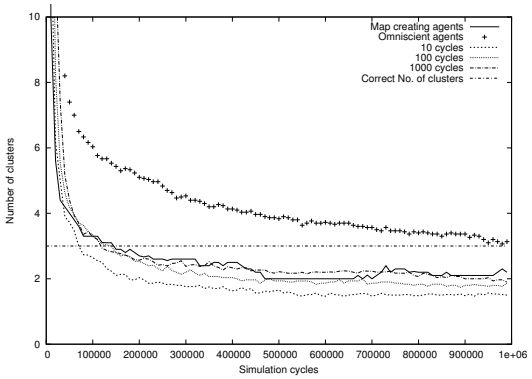
(d) Growing Neural Gas model on Wine database

Figure 5.19: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using a periodic laying policy. As a reference, the correct number of clusters is shown.

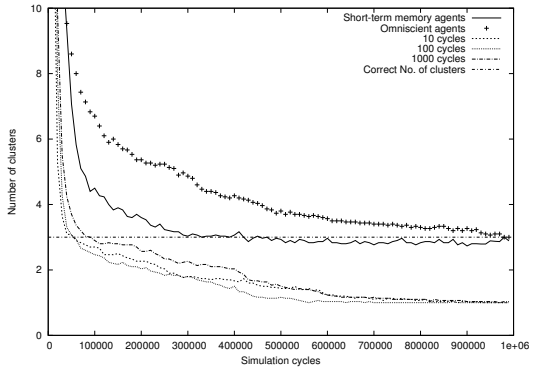
In all cases, the slowest algorithm is the omniscient agents algorithm. It is followed by the noncommunicating agents algorithms. It can be seen how these algorithms converge to two or one cluster.



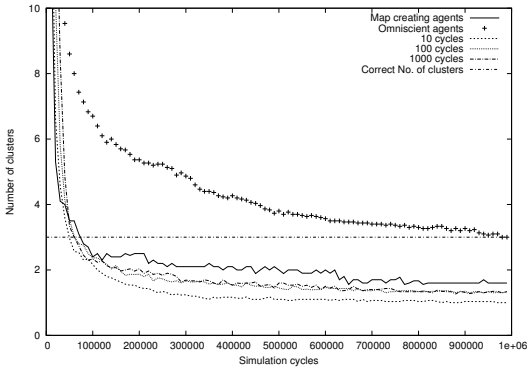
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



(d) Growing Neural Gas model on Wine database

Figure 5.20: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using a periodic laying policy. As a reference, the correct number of clusters is shown.

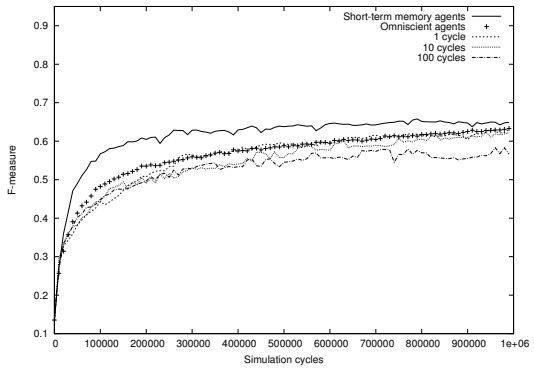
5.5 Results of Experiments on Information Exchange for Environment Representation Updates using an Adaptive Laying Policy

This section presents the results of the ant-based clustering algorithms using agents that update their environment representations with the information packets they find on the

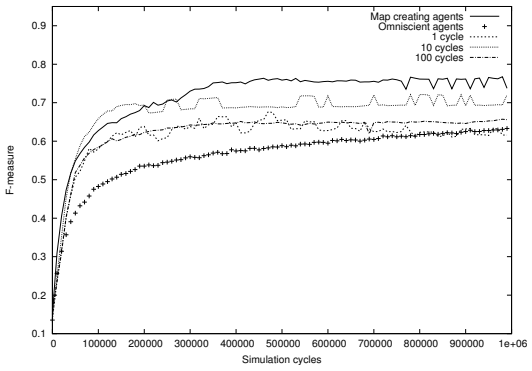
environment as they move through it.

5.5.1 *F*-Measure

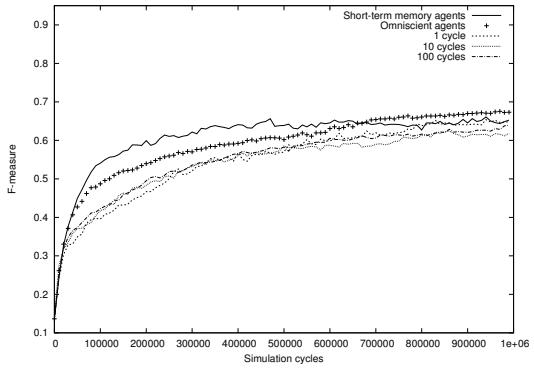
Figure 5.21 shows the *F*-Measure scores obtained by the tested algorithms with 10 agents on the test databases.



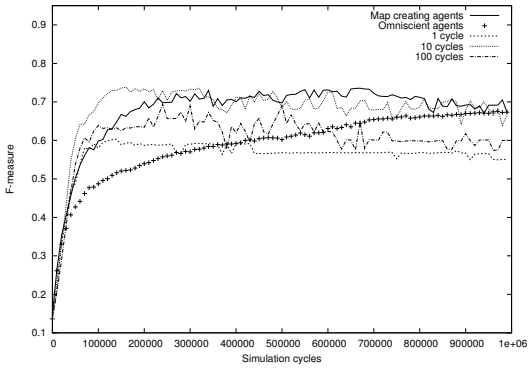
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



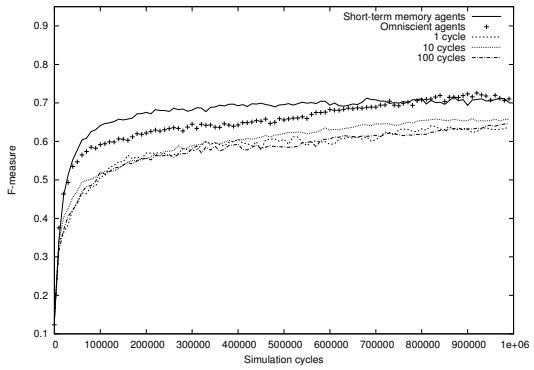
(d) Growing Neural Gas model on Wine database

Figure 5.21: *F*-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using an adaptive laying policy. A score value of 1 means perfect classification.

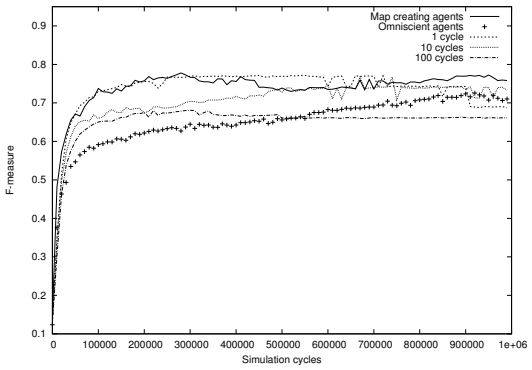
With the short-term memory model, the best performing algorithm is the one with noncommunicating agents. With the growing neural gas model, the best algorithm during the first simulation cycles is the map updating agents algorithm when agents wait for 100 cycles

before laying an information packet after a modification. After 200,000 cycles approximately, the simple noncommunicating agents algorithm becomes the best performing algorithm.

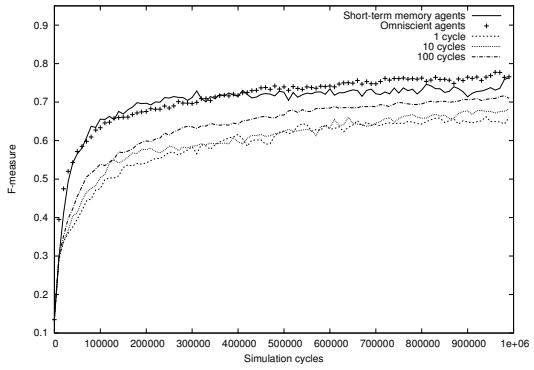
The graphs shown in figure 5.22 follow the same trend.



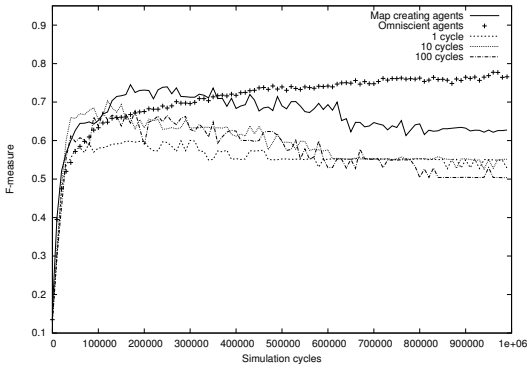
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



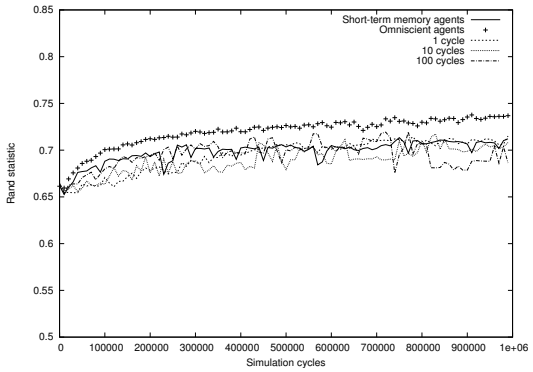
(d) Growing Neural Gas model on Wine database

Figure 5.22: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using an adaptive laying policy. A score value of 1 means perfect classification.

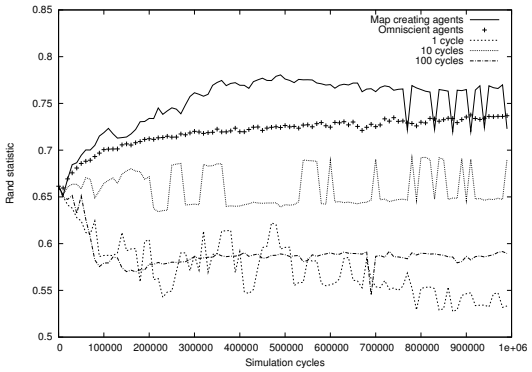
5.5.2 Rand Statistic

In figures 5.23 and 5.24 it can be seen that the omniscient agents algorithm is the best performing algorithm with the short-term model. The worst algorithm is the one with a delay of 1 cycle. With the growing neural gas model it happens the same thing except that, in the Iris database, the best performing algorithm is the noncommunicating agents

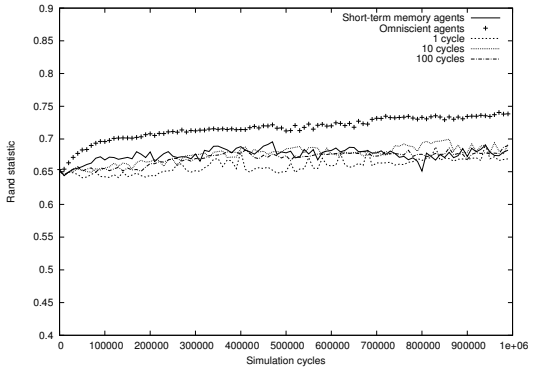
algorithm.



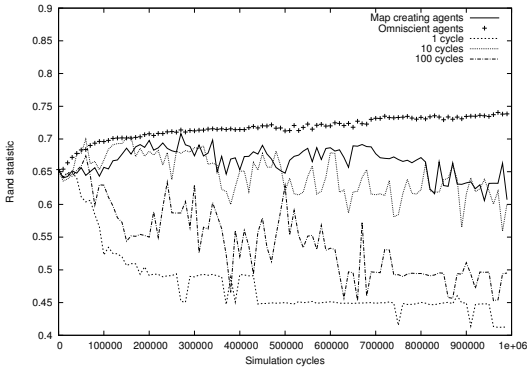
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

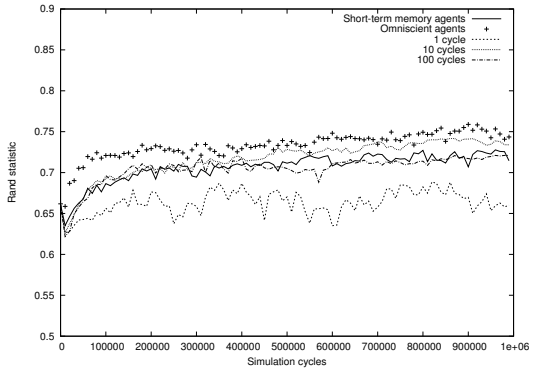


(c) Short-term memory model on Wine database

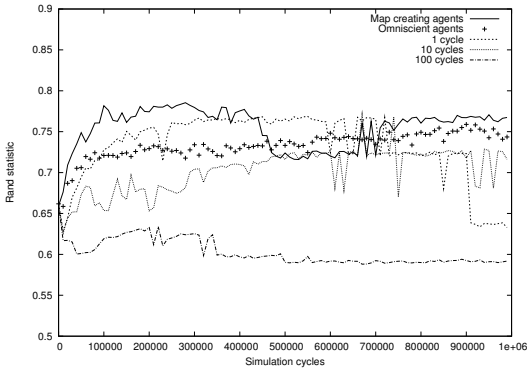


(d) Growing Neural Gas model on Wine database

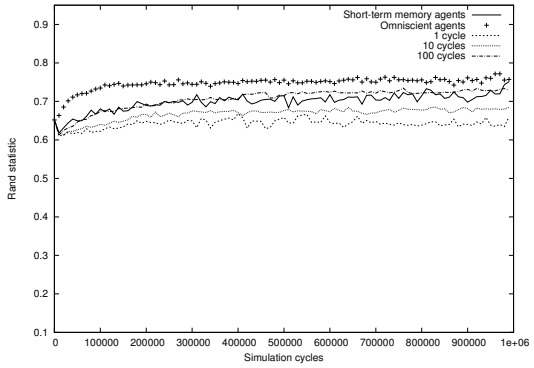
Figure 5.23: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using an adaptive laying policy. A score value of 1 means perfect classification.



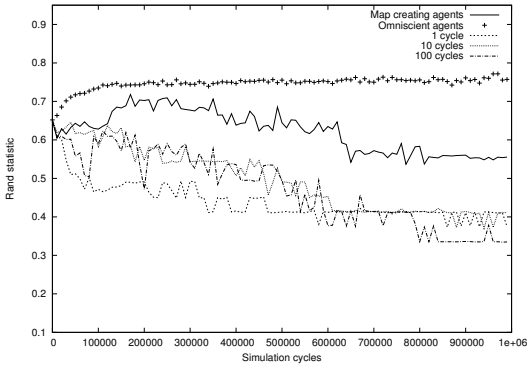
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

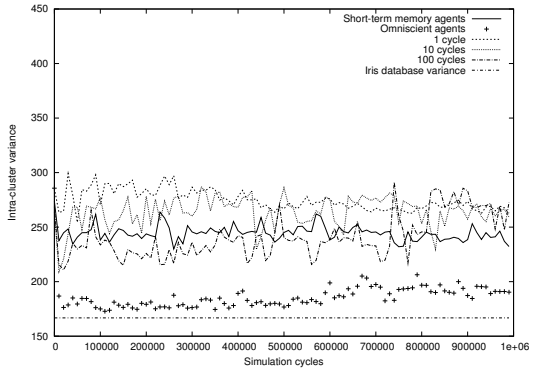


(d) Growing Neural Gas model on Wine database

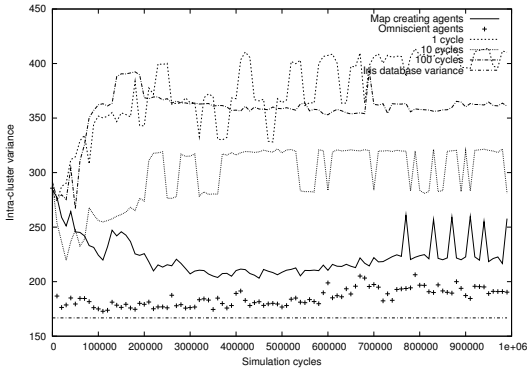
Figure 5.24: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using an adaptive laying policy. A score value of 1 means perfect classification.

5.5.3 Intra-cluster Variance

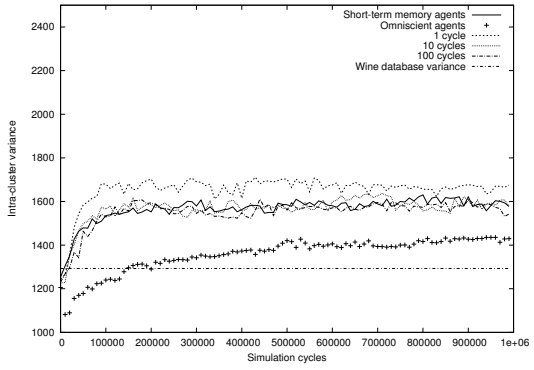
In figures 5.25 and 5.26 we can see the total intra-cluster variance scores over time for all tested algorithms on the Iris Plant and the Wine databases using 10 and 30 agents respectively. The algorithm with the lowest total intra-cluster variance is the omniscient agents algorithm. The second with lowest variance is the noncommunicating agents algorithms. The pattern we have seen before shows up again here, i.e., the algorithm with agents waiting 1 cycle after having manipulated the environment to lay an information packet on the environment is the worst performing algorithm, and the longer the delay, the better.



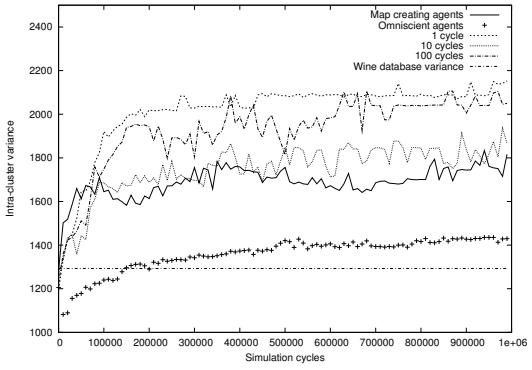
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

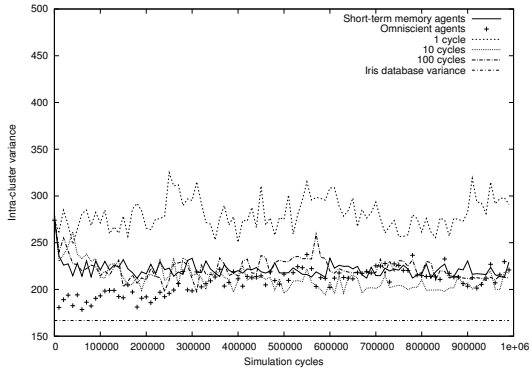


(c) Short-term memory model on Wine database

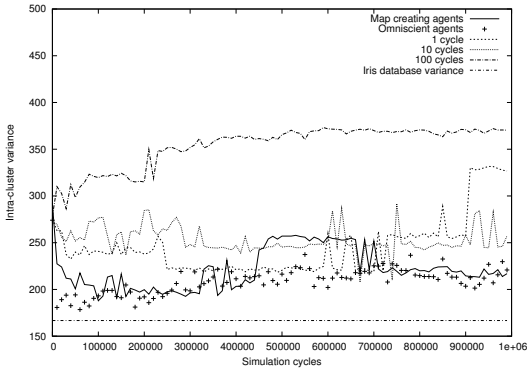


(d) Growing Neural Gas model on Wine database

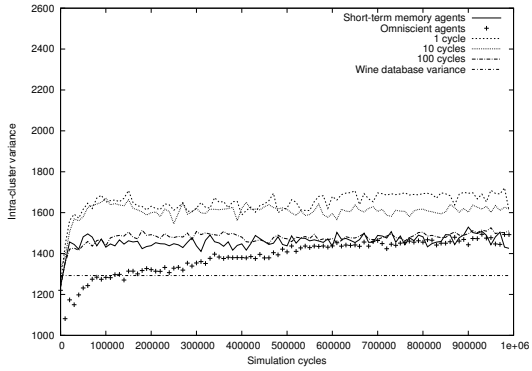
Figure 5.25: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using an adaptive laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.



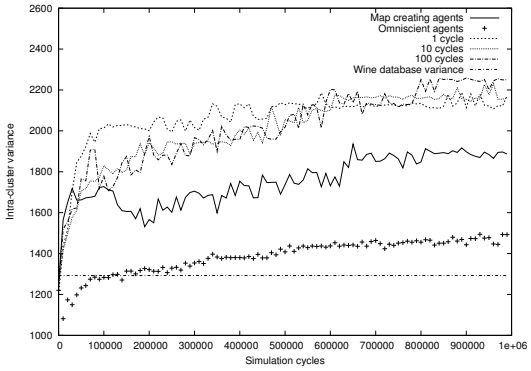
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

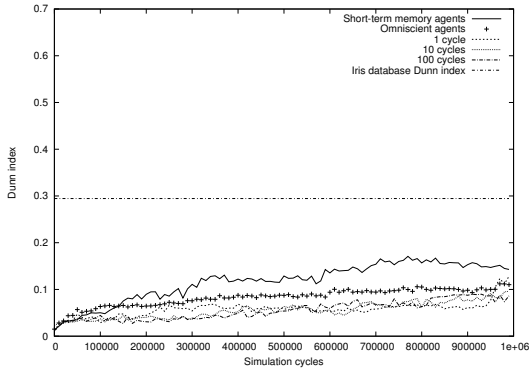


(d) Growing Neural Gas model on Wine database

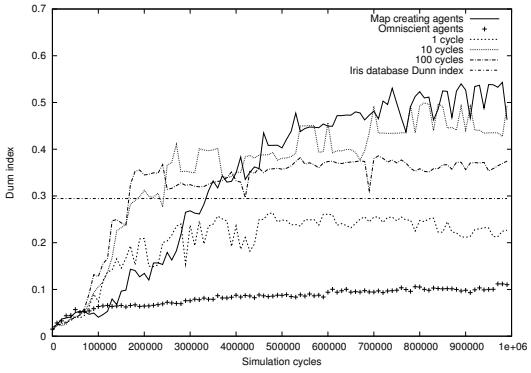
Figure 5.26: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using an adaptive laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.

5.5.4 Dunn index

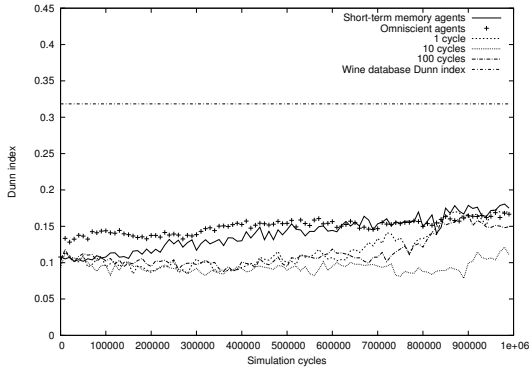
In figure 5.27 it is shown the Dunn index scores when using 10 agents. With the short-term memory the algorithm with highest scores is the noncommunicating short-term memory algorithm. With the Wine database it performs equally well than the omniscient agents algorithm. With the growing neural gas model the noncommunicating agents and the 10 cycles delay agents algorithms are the best in both databases.



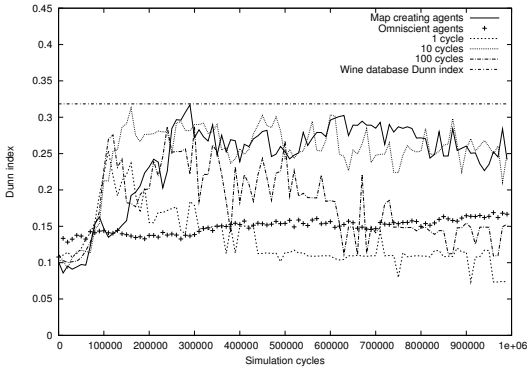
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



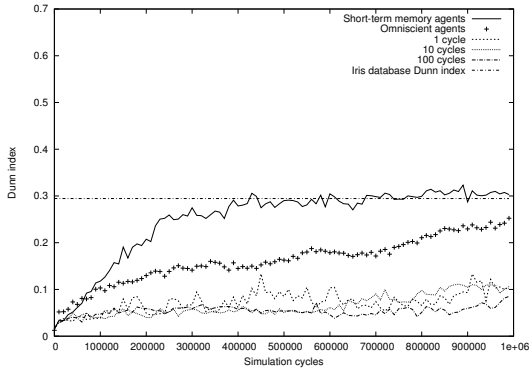
(c) Short-term memory model on Wine database



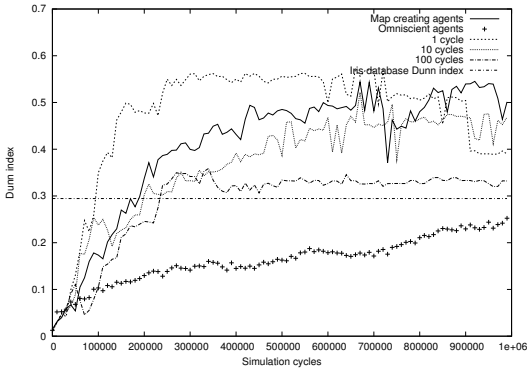
(d) Growing Neural Gas model on Wine database

Figure 5.27: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using an adaptive laying policy. As a reference, the Dunn index of the correct clustering is shown.

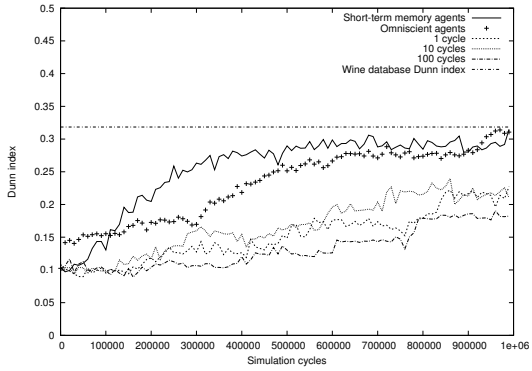
In figure 5.28 one can see how the performance of all algorithms degrades, specially of the communicating agents algorithms.



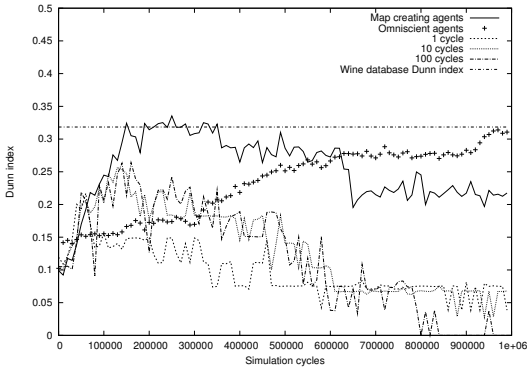
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

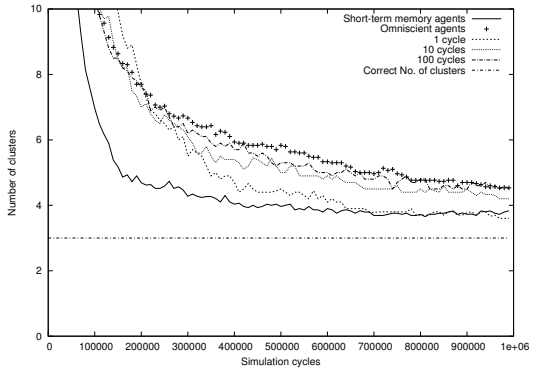


(d) Growing Neural Gas model on Wine database

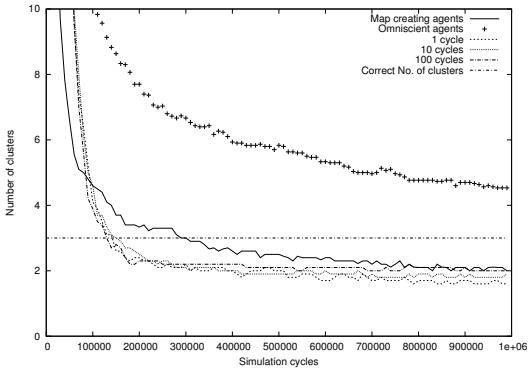
Figure 5.28: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using an adaptive laying policy. As a reference, the Dunn index of the correct clustering is shown.

5.5.5 Number of clusters

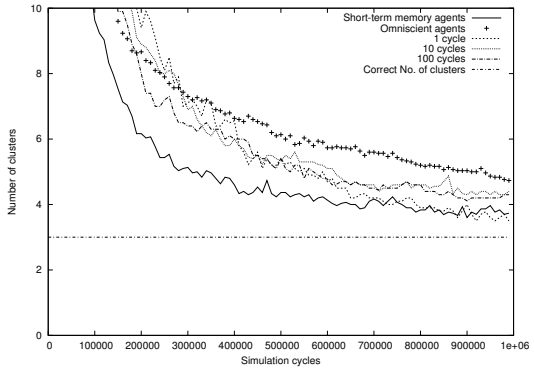
Figures 5.29 and 5.30 show the convergence of all tested algorithms. As was seen before, the slowest algorithm is the omniscient agents algorithm. No clear relationship can be seen among the communicating agents algorithms except, perhaps, when using 30 agents with the the Iris database and the short-term memory model.



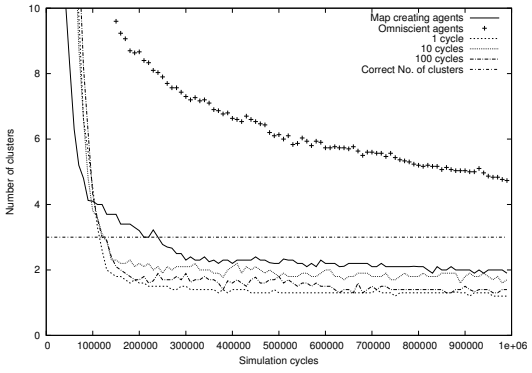
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

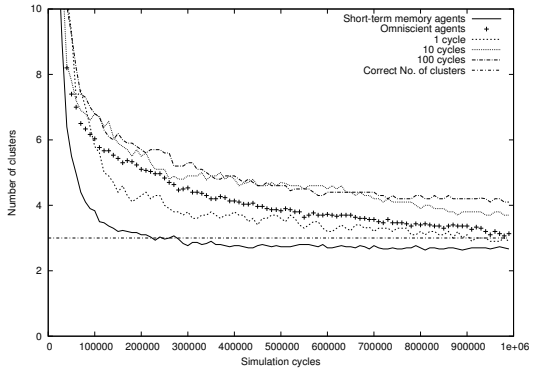


(c) Short-term memory model on Wine database

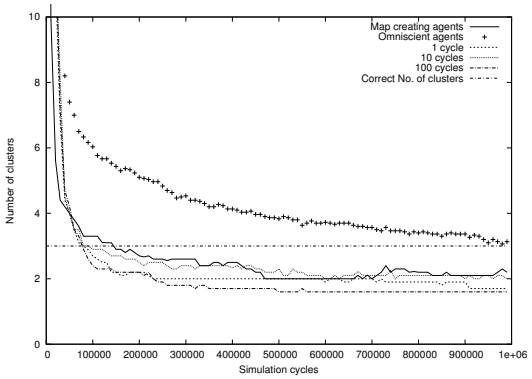


(d) Growing Neural Gas model on Wine database

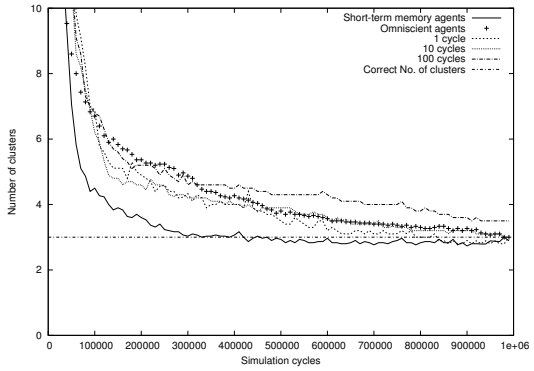
Figure 5.29: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 10 agents using an adaptive laying policy. As a reference, the correct number of clusters is shown.



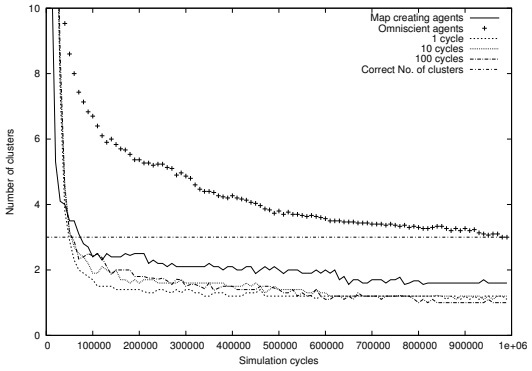
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



(d) Growing Neural Gas model on Wine database

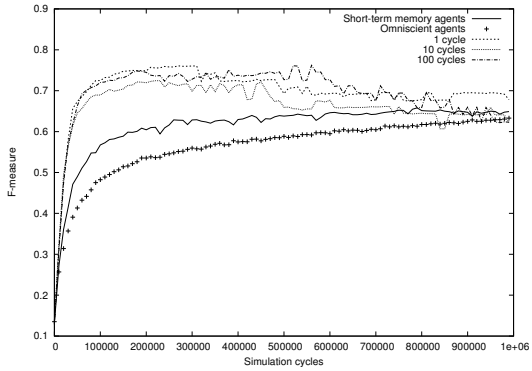
Figure 5.30: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents update their short-term memories, and an algorithm in which agents update their distribution map. The results were obtained using 30 agents using an adaptive laying policy. As a reference, the correct number of clusters is shown.

5.6 Results of Experiments on Information Exchange for Behavior Changes using an Adaptive Laying Policy

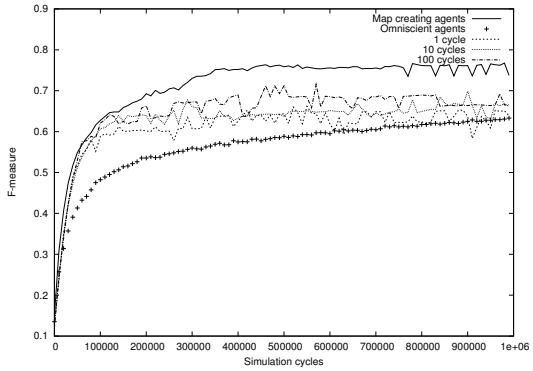
This section presents the results of the ant-based clustering algorithms using agents that change their dropping spot search trajectories based on the information they come across in the environment. The information is laid on the environment after that a given number of simulation cycles have elapsed after a modification to the environment.

5.6.1 F-Measure

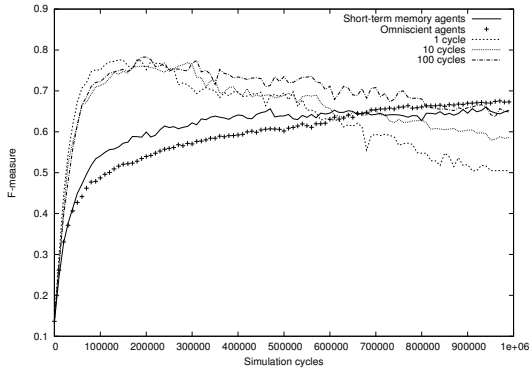
Figure 5.31 shows the F-Measure scores over time for the tested algorithms.



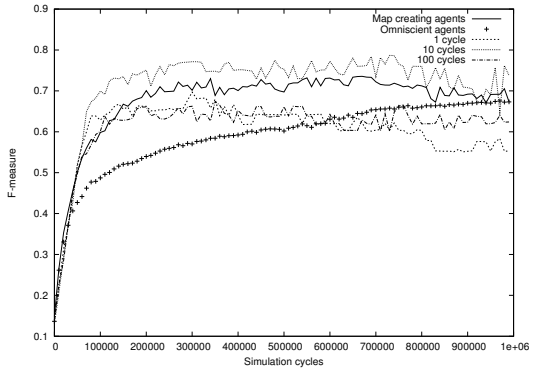
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



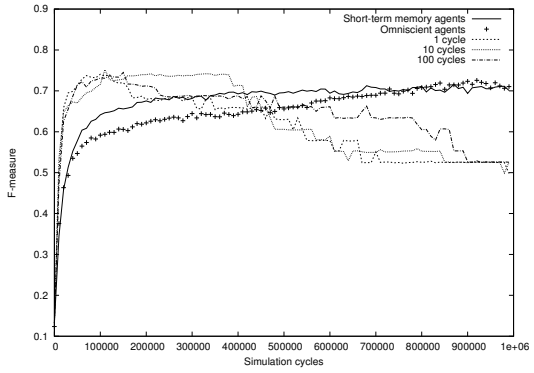
(d) Growing Neural Gas model on Wine database

Figure 5.31: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using an adaptive laying policy. A score value of 1 means perfect classification.

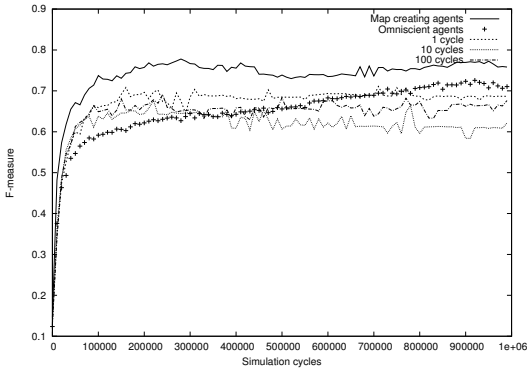
With the short-term memory model, the performance of the communicating agents algorithms is better than the omniscient agents and the noncommunicating agents algorithms. However, the performance drops steadily after 200,000 simulation cycles approximately. The performance of the information laying algorithms with the growing neural model is located

between that of the simple noncommunicating map creating agents and the omniscient agents algorithm.

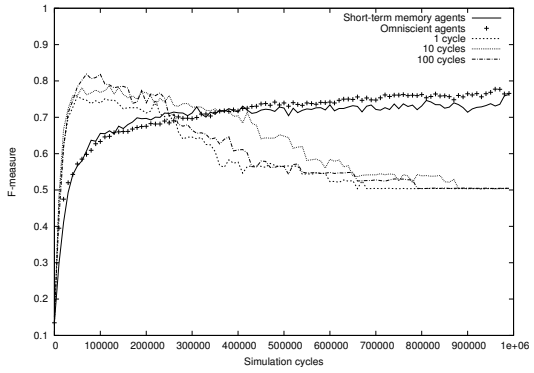
In figure 5.32 we can see how the performance of the communicating agents algorithms drops substantially.



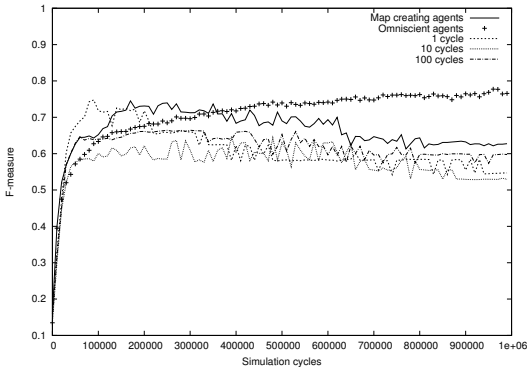
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

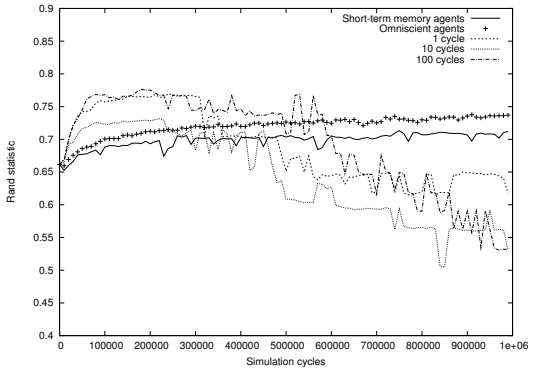


(d) Growing Neural Gas model on Wine database

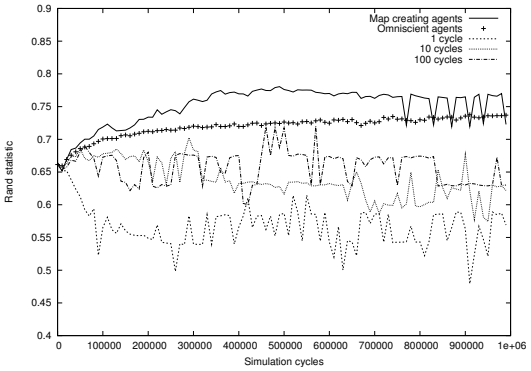
Figure 5.32: F-Measure scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using an adaptive laying policy. A score value of 1 means perfect classification.

5.6.2 Rand Statistic

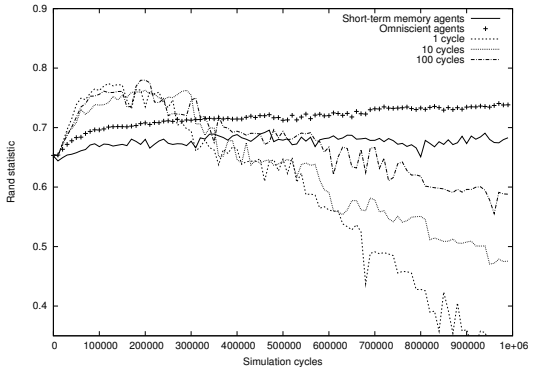
In figures 5.33c and 5.34c we can see how the behavior pattern observed before appears again. The observed pattern suggests that the more information available to the agents the worse. We will talk more on this issue in section 5.7.



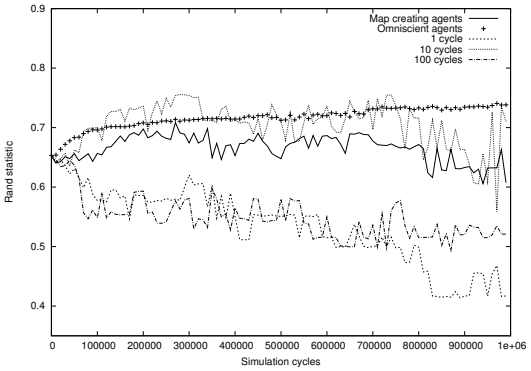
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

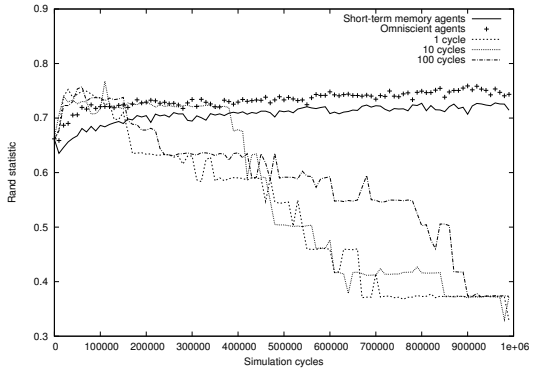


(c) Short-term memory model on Wine database

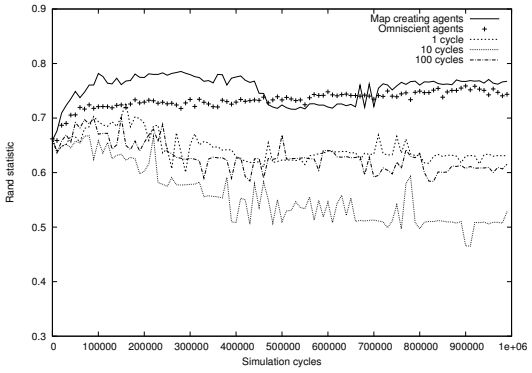


(d) Growing Neural Gas model on Wine database

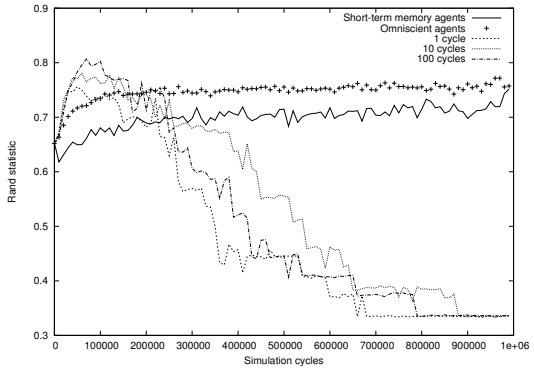
Figure 5.33: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using an adaptive laying policy. A score value of 1 means perfect classification.



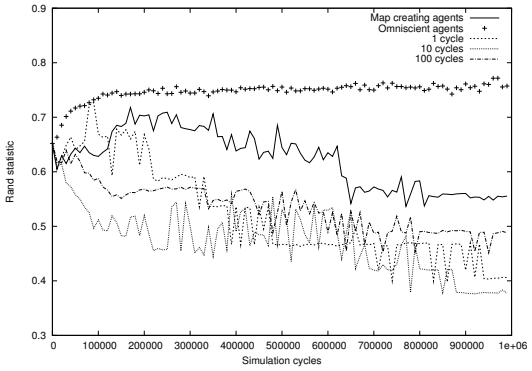
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

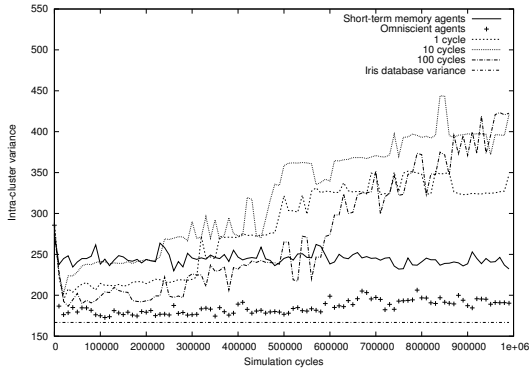


(d) Growing Neural Gas model on Wine database

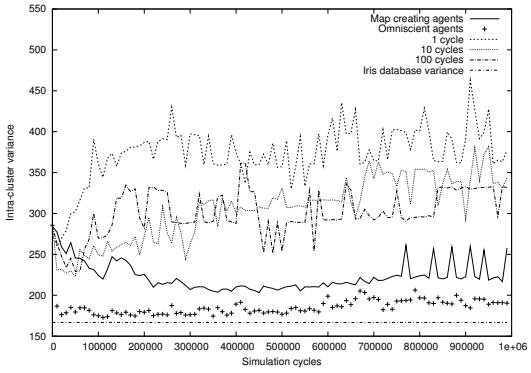
Figure 5.34: Rand statistic scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using an adaptive laying policy. A score value of 1 means perfect classification.

5.6.3 Intra-cluster Variance

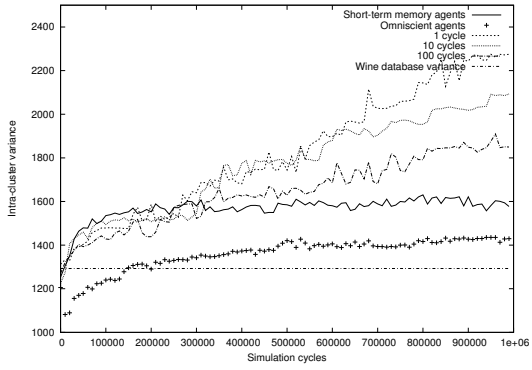
The pattern just mentioned appears again with this validity measure. See figure 5.35c. Besides this, we can see that the best performing algorithm in these eight graphs is the omniscient agents algorithm.



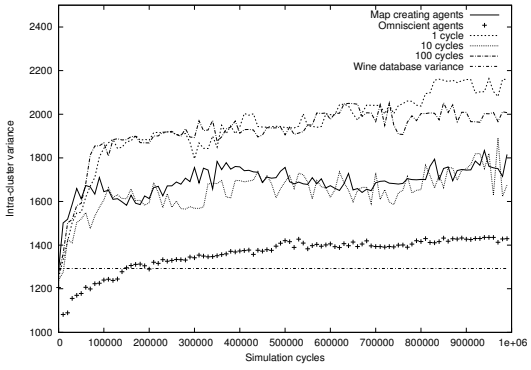
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

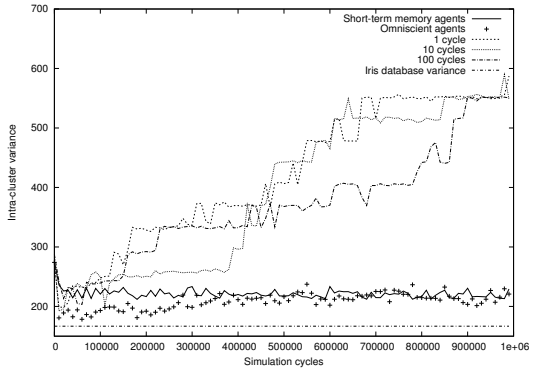


(c) Short-term memory model on Wine database

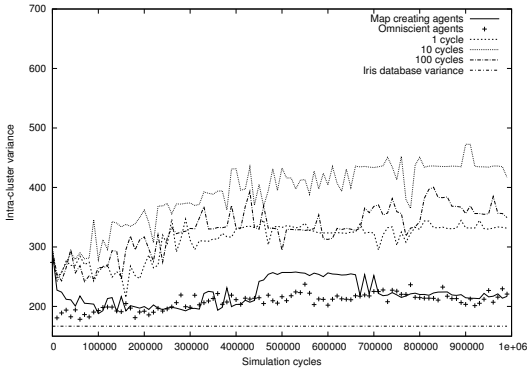


(d) Growing Neural Gas model on Wine database

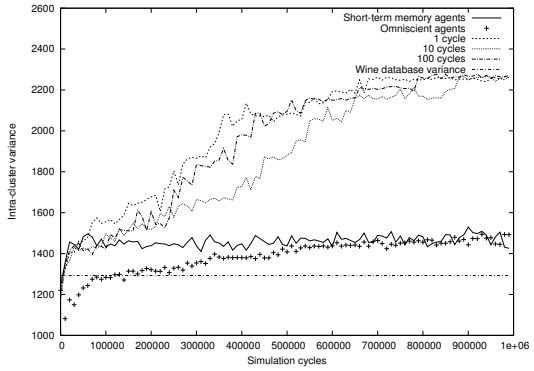
Figure 5.35: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using an adaptive laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.



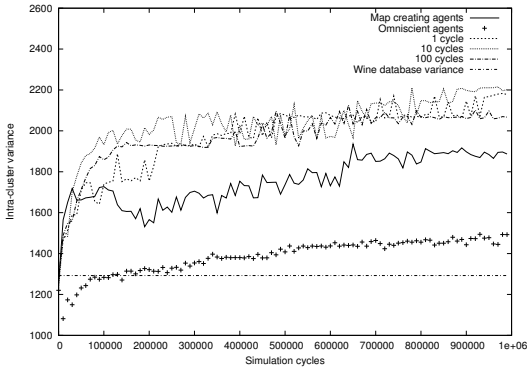
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

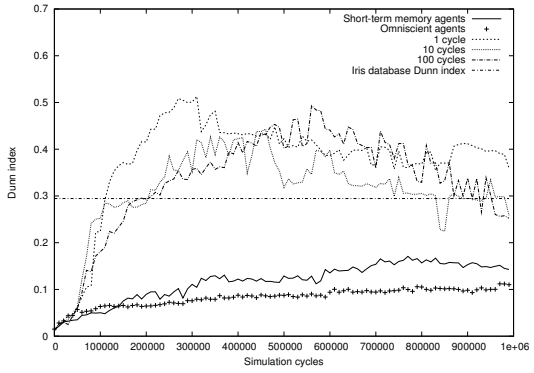


(d) Growing Neural Gas model on Wine database

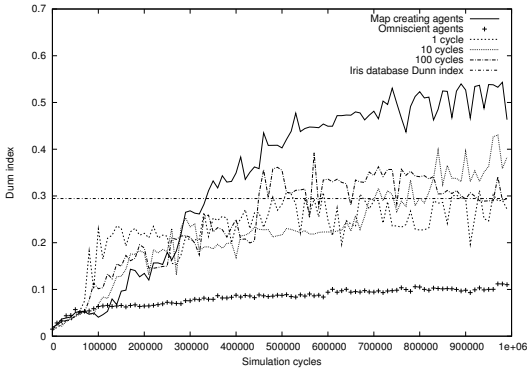
Figure 5.36: Total intra-cluster variance scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using an adaptive laying policy. As a reference, the total intra-cluster variance of the correct clustering is shown.

5.6.4 Dunn index

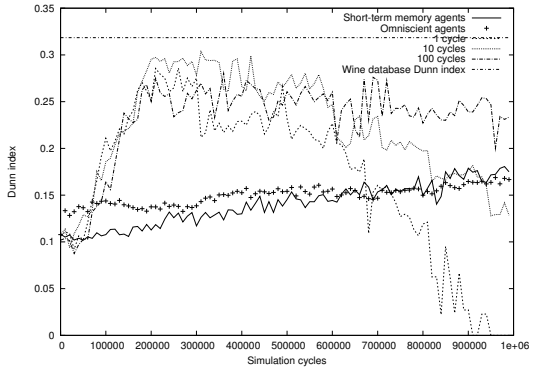
Figure 5.37 shows the Dunn index scores obtained by the tested algorithms with 10 agents. With the short-term memory model, the communicating agents algorithms are the best. With the Wine database, they reach a maximum and start decreasing. With the growing neural gas model and with the Iris database, they perform better than the omniscient agents algorithm but worse than the noncommunicating agents algorithm.



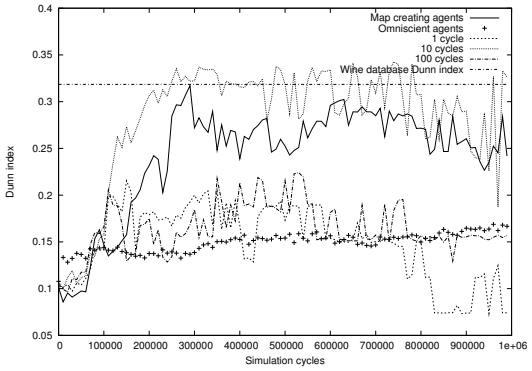
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



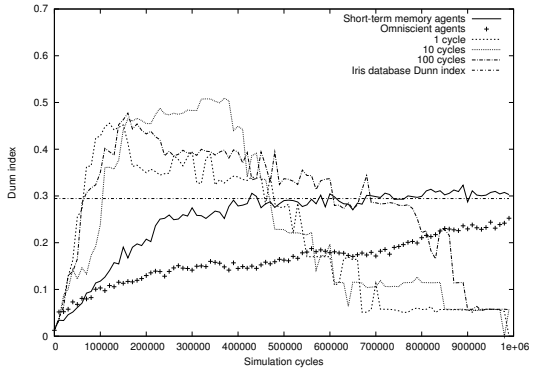
(c) Short-term memory model on Wine database



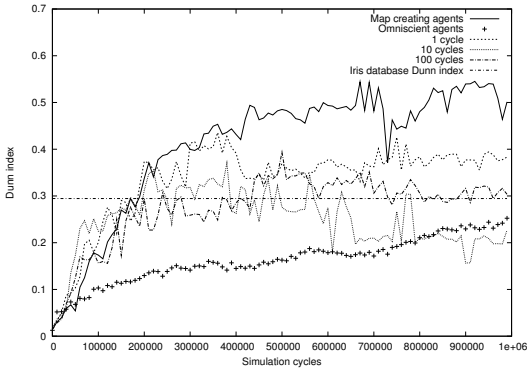
(d) Growing Neural Gas model on Wine database

Figure 5.37: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using an adaptive laying policy. As a reference, the Dunn index of the correct clustering is shown.

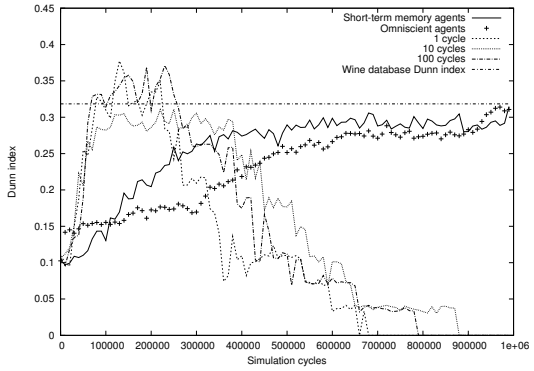
In figure 5.38 the Dunn index scores obtained by the tested algorithms with 30 agents are shown. In all cases except one, the communicating agents algorithms start performing quite well. However, in the end they converge to a value near zero.



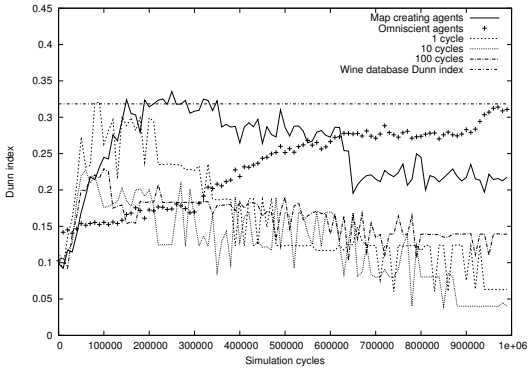
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database

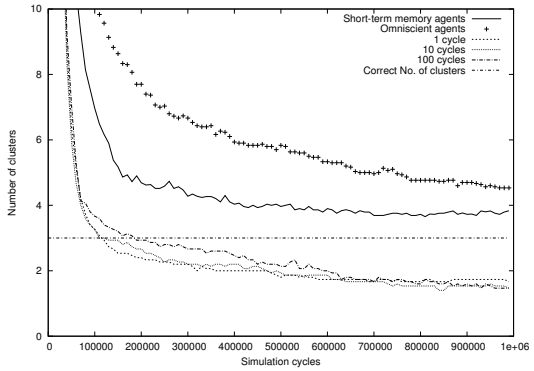


(d) Growing Neural Gas model on Wine database

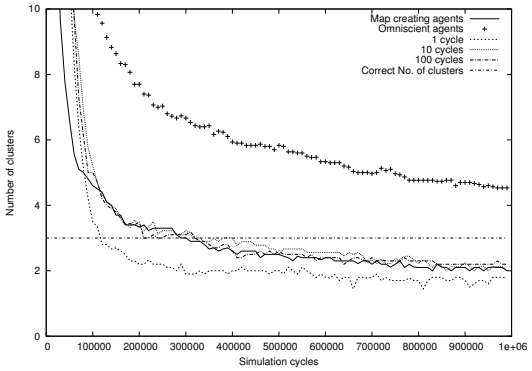
Figure 5.38: Dunn index scores over time for the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using an adaptive laying policy. As a reference, the Dunn index of the correct clustering is shown.

5.6.5 Number of clusters

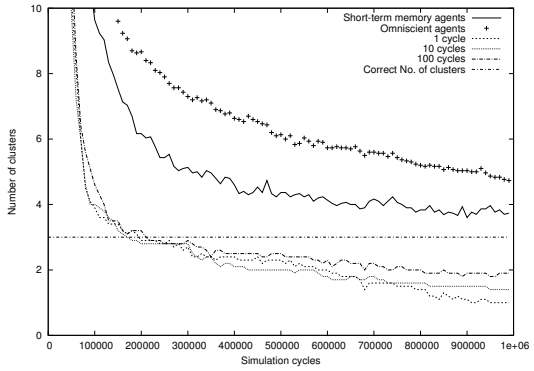
Figures 5.39 and 5.40 show the number of clusters discovered by the tested algorithms over time with 10 and 30 agents respectively.



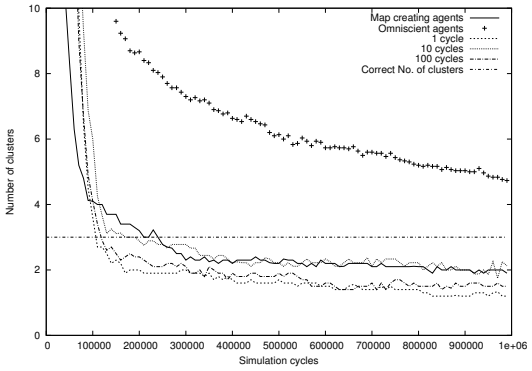
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database

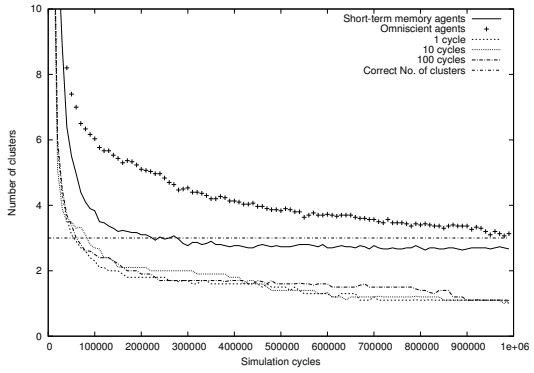


(c) Short-term memory model on Wine database

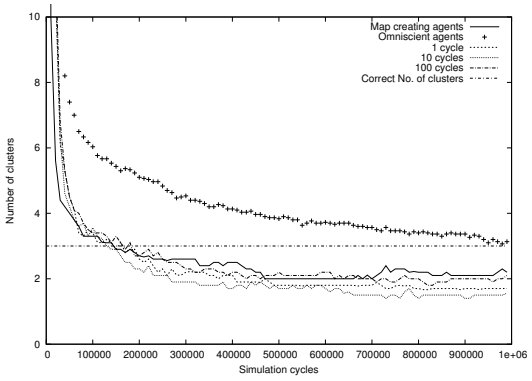


(d) Growing Neural Gas model on Wine database

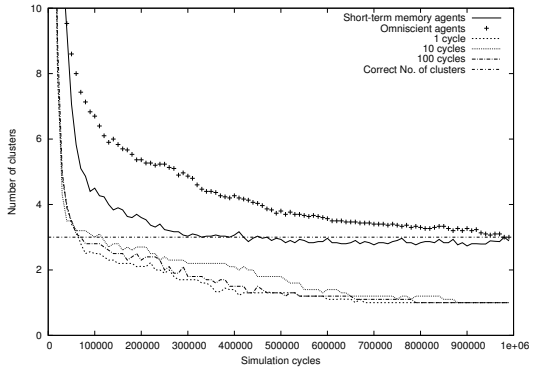
Figure 5.39: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 10 agents using an adaptive laying policy. As a reference, the correct number of clusters is shown.



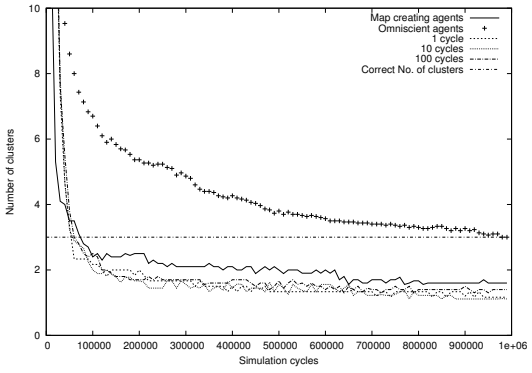
(a) Short-term memory model on Iris database



(b) Growing Neural Gas model on Iris database



(c) Short-term memory model on Wine database



(d) Growing Neural Gas model on Wine database

Figure 5.40: Number of found clusters over time in the Iris Plant and Wine databases of Lumer and Faieta's short-term memory agents algorithm, the same algorithm but with map creating agents, an omniscient agents algorithm, an algorithm in which agents change their dropping spot search trajectories based on the contents of other agents' short-term memories, and an algorithm in which agents change their dropping spot search trajectory based on other agents' environment maps. The results were obtained using 30 agents using an adaptive laying policy. As a reference, the correct number of clusters is shown.

5.7 Discussion of Results

In contrast with the previous chapter where only two series of experiments were run, in this chapter they were four. This is because we tried with two different laying policies that proved very similar in the end. In the first two series of experiments, a periodic laying policy was tested. With it, every given number of simulation cycles, an agent drops a reference to the inner structure that holds its environment representation, in our case a short-term

memory and a growing neural gas network. In the second series of experiments, an adaptive laying policy was tried. By an adaptive laying policy, we mean a policy that changes all the time according to a given circumstance an agent may be. In this case, it was decided that laying an information packet on the environment would be sound only after an agent had manipulated somehow (dropped or picked an object) the environment. Since data on the environment begin to concentrate into clusters, the absolute laying frequency using the adaptive approach decreases over time.

In the periodic laying policy, three different frequencies were tested: 10, 100, and 1000 cycles. In the adaptive laying policy we also tried three delays: 1, 10, and 100 cycles. They are different because it does not make much sense to notify the other agents about a modification of the environment long after it has been done. Of course, a quick delay provokes a rather local notification radius, i.e., only an agent that was already nearby is going to consume that information. Indeed, this fact makes the results more interesting.

Now, let us begin our analysis, and for that purpose we will divide it into two parts, one for each laying policy.

5.7.1 Periodic Laying Policy

There are two identifiable behavior patterns. When the information is used for updating or enriching the environment state representation, a frequent laying policy is better than an infrequent one. See for example in figure 5.1d how the performance of the algorithm with agents laying information every 10 cycles is better than the algorithm with agents laying information every 100 cycles, but this in turn has a better performance than the algorithm with agents laying information every 1000 cycles. This is confirmed in figures 5.8a and 5.8c. The other pattern emerges when agents use the information to change their dropping spot search trajectories. It is, interesting enough, opposite to the pattern just described. In this case, the less frequent the deposition of information, the better. This is confirmed in figures 5.11d, 5.12b, 5.13b, 5.13d, etc. It is evident.

The overall performance however, is not encouraging since these algorithms converge to one or two clusters in the best case.

5.7.2 Adaptive Laying Policy

With this laying policy, things change a bit. In the case of agents updating their environment state representation, the shorter the deposition delay, the worse. This result which is confirmed in figures 5.21d, 5.23d, 5.24, etc. together with the result obtained with the periodic laying policy, shows that the local component we mentioned earlier pays a key role with the representation update model. Indeed, it is logical since the usefulness of the information for the purpose of updating or enriching an environment state representation depends on how much *new* information is received. If the information received reflects the state of the region in which an agent is already, then it is not likely to be very useful.

For the same reason, the behavior pattern when agents use the information to decide whether to change their dropping trajectory is the same. That is, the longer the delay, the better. This is confirmed in figure 5.35c and 5.37c.

5.8 Conclusions

In this chapter, the effects of intentional indirect communication among agents in ant-based clustering algorithms were explored. The main motivation of this exploration was to increase the number of exchanges among agents without incrementing their number. Two information laying policies were studied: a periodic laying policy and an adaptive laying policy. With the periodic laying policy, an agent drops information packets every given number of simulation cycles. With the adaptive laying policy, an agent drops information after it has modified the environment and a given number of simulation cycles have passed.

Both policies were tested with two different information exploitation strategies: for updating the agents' internal environment representations and for changing their dropping spot search trajectories. With the first strategy and the periodic laying policy, agents behave better with high laying frequencies. This fact supports the hypothesis that the closer the environment representation to the real environment state, the better. The first exploitation strategy when used with the adaptive laying policy makes agents behave better with long deposition delays than with short ones. This is because short delays provoke agents to drop information near the region where they modified the environment and if other agent comes across this information, it is unlikely that this information will be useful to it since the information it contains, i.e., the recent change in the environment, is readily available to the consumer agent. With the second exploitation strategy, the more information available to the agents, the worse. That is, when using a high frequency laying or with a small delay after manipulation, the performance of the algorithm degrades.

Chapter 6

Conclusions

This thesis investigated the effects of communication among agents on the clustering quality and on the convergence speed of ant-based clustering algorithms. In total, four different information exchange strategies were studied:

1. Direct information exchange for updating agents' environment representations.
2. Direct information exchange for changing agents' dropping spot search trajectories.
3. Intentional indirect information exchange for updating agents' environment representations.
4. Intentional indirect information exchange for changing agents' dropping spot search trajectories.

To design the experiments, questions such as: What information will agents exchange? When will they exchange it? How will they do it? What will they do with that information? had to be answered. Although there are no general answers to them, we tried to explore four issues when we proposed answers to them. First, what are the effects on the algorithms' performance when agents exchange information from different levels of abstraction?. Second, what is the impact on the algorithms' performance when agents exchange information in different ways?. Third, what happens when agents use the information for different purposes?. And fourth, what happens when agents choose to use immediately or after some delay the available information?

Let us discuss how we coped with these issues and what were our results. The information agents exchanged in the experiments belong to two different levels of abstraction:

- Memorized grid locations on which an agent had dropped a data object, and
- Pointers to promising dropping locations

The first choice was needed in order to compare the performance of the algorithms with and without communicating agents. In fact, this model is just a simple extension of Lumer an Faieta's short-term memory agents model. The second choice tries to implement the idea of

map exchanging agents. Maps were represented as a set of vectors simultaneously distributed in the grid and in the attribute space of data objects. Maps provide more information than memorized dropping spots because they can adapt to changes in the spatial distribution of objects.

With regard to the moment in which agents should exchange information, we decided to couple this issue with the way agents were going to exchange information. That is, agents decided to exchange information depending on the exchange method used. When agents exchanged information directly, they did it whenever they met on the grid. When they exchanged information indirectly, they did it whenever an agent came across an information packet on the grid.

One of the most critical part in the experiments design was to decide what agents should do with the exchanged information. Our hypothesis was that agents with information about the spatial distribution of data on the grid, would be able to choose the best location on which to drop an object (if they were loaded), or the best regions of the environment to explore (if they were unloaded). With informed decisions, agents could create better clusters in a faster way. We therefore decide to explore the idea by (i) letting agents represent their environment and after every exchange, update or enrich their representations, and (ii) changing their dropping spot search trajectories, i.e., they were allowed to "change their minds" regarding their supposed best dropping spot on the grid.

The following sections present in detail the conclusions drawn from the experimental results with each of the tested strategies.

6.1 Direct information exchange

In the experiments run for this part of the thesis, direct information exchange occurs only when two or more agents meet a location on the grid. Hence, the probability of an encounter between two agents moving randomly raises as the number of agents is increased (assuming a constant size of the grid). In robotic experiments [16] this fact is confirmed. In these experiments, we tried to take advantage of this fact and use it to study the effect of increasing the information exchange frequency among agents. This is the reason of using two different sizes of agent populations in all the experiments.

The actual effect of information exchange depends not only on the frequency of encounters but also on their effectiveness, i.e., information exchange is beneficial as long as the exchanged information is *useful*. In the ant-based clustering context, information usefulness depends very much on how well an agent's environment representation approximates the *current* environment state and/or how it uses the available information. The environment is not really well approximated during the first stages of the clustering process because of the high environment dynamism. When this phase finishes, once that some initial clusters exist, information exchange is more effective as the environment is less dynamic.

6.1.1 When used for updating environment representations

The results obtained in the first series of experiments, those with agents updating their environment representations, are somewhat discouraging. The worst performing algorithm is the one with map updating agents, and the second worst is the memory updating agents algorithm. In the first case, further experimentation must be done to know what went wrong in the experiments. It is very likely that the parameter set used in the experiments was not the correct one. We leave this issue as future work. In the case of the memory updating agents algorithm, the relatively poor performance is not as discouraging as is the obtained with map creating agents. The negative offset observed in the response plots with respect to the omniscient agents algorithm response plots, is an indication of the correct, although not perfect, approximation to the real state of the environment.

6.1.2 When used for changing dropping spot search trajectories

The results obtained with dropping spot search trajectory changing agents are quite the opposite from the previous ones. In this case, the clustering quality is improved by the communicating agents algorithms. They produce even better clusters than the version with omniscient agents. This is true only when using 10 agents. With 30 agents, the relative performance of these algorithms is deteriorated. This suggests the existence of some critical number of agents that should behave this way with a given database. Determining the optimum number of agents for a given clustering task remains a research issue.

6.2 Indirect information exchange

In the experiments run for exploring the effects of intentional indirect information exchange among agents in ant-based clustering algorithms, agents lay packets which contain information about data distribution on the environment for others to pick and use. This strategy is inspired by the anal trophallaxis phenomenon among social insects but it also has other reasons. Direct communication among agents in ant-based clustering has two disadvantages: (i) even when the number of exchanges increases, we cannot expect many of them to happen since the number of agents must be kept small (for performance reasons), and (ii) many exchanges do not have any effect since agents walk in a randomly fashion, i.e., two agents coincide many times, over and over again, before they follow different trajectories. So the idea is that if we let agents lay information on their environment, it could be possible to increase dramatically the number of exchanges without even increasing the number of agents.

Two information laying policies were studied: a periodic laying policy and an adaptive laying policy. With the periodic laying policy, an agent drops information packets every given number of simulation cycles. With the adaptive laying policy, an agent drops information after it has modified the environment and a given number of simulation cycles have passed.

6.2.1 When used for updating environment representations

The results show that the more information available to the agents, the better the performance of the algorithms with environment representations updates. With a periodic laying frequency, the higher the frequency, the better. And with the adaptive laying policy, the shorter the delay, the better. This results confirm the intuition which says that to maintain an up-to-date environment representation, an agent has to acquire fresh information all the time.

6.2.2 When used for changing dropping spot search trajectories

When agents use the information only to decide whether to change their dropping spot search trajectory, the more information available to them, the worse. This result is also sound if we think of agents as "changing their minds" based on the information provided by other agents. An agent may receive contradicting information or may be misleaded to a nonpromising region in the environment. Therefore, with this information exploitation strategy, high laying frequencies and short delays have negative impact on the algorithms performance.

6.3 Discussion

In this thesis, the effects on clustering quality and convergence speed of direct and indirect communication among agents in ant-based clustering algorithms were studied. The results show that nonstigmergic communication among agents in these algorithms has effects on the final clustering obtained by the algorithm. The final effects depend on the type of information exchanged, its use, its availability and the number of participating agents.

The hypothesis that guided our work was:
In ant-based clustering algorithms, it is possible to reduce convergence time and improve clustering quality by letting agents exchange information about the spatial distribution of data and their type.

The underlying assumption that is the core of this hypothesis is: if an agent is loaded with a data object, the best place on the grid it can drop that data object is next to, or at least near, the location of its nearest neighbor in attribute space.

An loaded omniscient agent then, would move directly to the location on the grid of the nearest neighbor of the data object it is carrying. This behavior was programmed but it did not obtained the best clustering quality scores. Why? One possible explanation is that agents fail to drop the data objects they are carrying next to their nearest neighbors. This is a fact in our experiments since agents do not "jump". They change their walking direction only, and therefore, they can drop its load somewhere on the grid between its load original location and their supposed destination. Future work should confirm or reject this hypothesis. Another possible explanation is that our omniscient agents are only an

approximation to real omniscient agents, that is, in our experiments these agents know the exact location of all the objects on the grid but they do not know the location of the objects other agents are carrying. So, their choice of the nearest neighbor is not necessarily the true nearest neighbor. It can be solved if the algorithm runs with a single agent, but in this case, communication among agents is no longer meaningful.

To select the best location on which to drop a data object, when agents do not have full observability of the environment, we tested two strategies: (i) to enrich/update an internal representation, and (ii) to change the dropping spot search trajectory. In the former, high availability of information has a positive impact. In the latter, it was shown that high availability of information has negative consequences. These conclusions are drawn from all our experiments, regardless the actual representation scheme used or the way information was exchanged. Future work involves testing these conclusions with experiments with user-controlled databases. This last thing in order to eliminate the bias introduced by the used test databases.

Two facts motivated this work. One one hand the lack of knowledge about the effects of direct and intentional indirect communication among agents in ant-based clustering algorithms. On the other hand, the slow convergence speed of the classical stigmergic version of the algorithm. This thesis studied both. The hypothesis was that in ant-based clustering algorithms it is possible to reduce convergence time and improve clustering quality by letting agents exchange information about the spatial distribution of data and their type. The hypothesis is true when agents do not update their environment state representation and the number of participating agents is small. The convergence speed is improved in almost all cases but the quality of the final clustering is not satisfactory.

This thesis tries to fill a space in literature regarding agent communication issues in swarm intelligent systems. Communication among agents in swarm intelligent systems and more generally in multi-agent systems, is crucial in order to coordinate their activities so that a particular goal at the collective level is achieved. From an individual perspective, the problem consists in establishing communication policies that determine *what*, *when*, and *how* to communicate with others. In general, communication policies will depend on the nature of the problem being solved. This means that the solvability of problems by swarm intelligent systems depends, among other things, on the agents communication policies, and setting an incorrect set of policies into the agents may result in finding poor solutions or even in the unsolvability of problems. This is confirmed with our results.

6.4 Contributions

Two previously unexplored problems in the area of ant-based clustering algorithms were studied in this thesis. Namely, the use of direct and intentional indirect communication among agents. This thesis reports experimental evidence that show what behavior patterns agents exhibit when using different information communication and exploitation strategies.

Table 6.1 summarizes our contributions to the state of the art in ant-based clustering algorithms.

Table 6.1: Contributions to the state of the art in ant-based clustering algorithms

Aspect	Contribution
Direct communication among agents in ant-based clustering algorithms	No previous works are known that study the area of direct communication among agents in ant-based clustering algorithms. This work is a first approach to it and our results show that it is promising when certain condition are met.
Intentional indirect communication among agents in ant-based clustering algorithms	Ant-based clustering works because of the indirect communication among agents through local modifications of the environment. However, intentional indirect communication among agents in ant-based clustering algorithms had not been explored before this thesis.

6.4.1 Publications derived from the work on this thesis

Table 6.2: Publications derived from this work

Publication
Montes de Oca M. A., Garrido L., and Aguirre J.L. *A first approach to study the effects of direct information exchange between agents in ant-based clustering*. International Journal on Lateral Computing. International Federation on Lateral Computing. Volume 1. Number 1. May 2005.
Montes de Oca M. A., Garrido L., and Aguirre J.L. *Efectos de la comunicación directa entre agentes en los algoritmos de agrupación de clases basados en el comportamiento de insectos sociales*. Inteligencia Artificial. Revista de la Asociación Española de Inteligencia Artificial (AEPIA). ISSN 1137-3601. Volume 9. Number 25. 1st quarter 2005 (To appear).
Montes de Oca M. A., Garrido L., and Aguirre J.L. *An hybridization of an ant-based clustering algorithm with growing neural gas networks for classification tasks* In 20th Annual ACM Symposium on Applied Computing. Santa Fe, N.M. USA. March 2005. pp 9-13.
Montes de Oca M. A., Garrido L., and Aguirre J.L. *A first approach to study the effects of direct information exchange between agents in ant-based clustering*. Abstract Compilation of the XXXV Congreso de Investigación y Desarrollo Tecnológico. Tecnológico de Monterrey. Monterrey, N. L. México. January 2005. pp. 280.
Montes de Oca M. A., Garrido L., and Aguirre J.L. *A first approach to study the effects of direct information exchange between agents in ant-based clustering*. In Proceedings of the First World Congress on Lateral Computing. Bangalore, India. December 2004.
Montes de Oca M. A., Garrido L., and Aguirre J.L. *On the effects of direct communication between agents in ant-based clustering algorithms*. In Proc. of the Fifth Iberoamerican Workshop on Multi-Agent Systems. Iberagents 2004. Puebla, México. November 2004.

6.5 Future Work

Quite a lot of work remains to be done in the area of communication among agents in ant-based clustering algorithms. Let us first list those that are directly related to the work done in this thesis. First, we have already mentioned that work is needed to determine the performance of an ant-based clustering algorithm with real omniscient agents. A single agent is capable to perform clustering and it would be really omniscient in the sense that knows exactly where the objects are located on the grid all the time. However, omniscience in the sense of knowing what to do in order to maximize its performance measure would still be a research issue since it is not evident, even to us humans, what to do in a totally observable environment in which one has to cluster data objects. Second, it is necessary to explore the behavior of the algorithms with different parameter settings. Unfortunately, there is no theory that could explain how ant-based clustering really works, and therefore, we are forced to experiment with different parameter settings. Third, it is of great importance to explore the effects of a *mixed* communication strategy, i.e., using at the same time a direct and an intentional indirect communication strategy or using agents that change their communication policies during the clustering process.

In chapter 3, we described an improved version of the Lumer and Faieta's ant-based clustering algorithm authored by Handl et. al. Future work should focus on this algorithm to study the effects of different communication policies among agents, since its performance is far superior than that of Lumer and Faieta's algorithm.

With regard to swarm intelligent systems, and in spite of their success, we need to consider the question of whether agents should/could communicate in other ways to achieve organization or better solutions to problems. There is no general answer nor general communication policy that will apply equally well to all problems. This is why we need to formalize the effects of letting agents use different communication policies. With this knowledge and considering the characteristics of a particular problem, we could either improve the performance or permit the application of a swarm intelligent system to solve it. Moreover, a swarm intelligent system may find adequate to change communication policies *during* problem solving. This imposes the need of formally studying problems and their behavior when solved by swarm intelligent systems. From this, we can finally conclude that Swarm Intelligence is an exciting research area and much work is yet to be done. The pay off could be really high if we can move from an experimental research strategy to a theoretical one.

Bibliography

[1] Martinoli A. and F. Mondada. Collective and cooperative group behaviours: Biologically inspired experiments in robotics. In Khatib. O. and Salisbury. J.K., editors, *Proceedings of the Fourth International Symposium on Experimental Robotics ISER-95*, pages 3–10. Springer Verlag, 1995.

[2] Periklis Andritsos. Data clustering techniques, 2002.

[3] G. Beni and J. Wang. Swarm intelligence. In *Proceedings of the Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428, 1989.

[4] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press, 1999.

[5] Blake C.L. and Merz C.J. Uci repository of machine learning databases. [http://www.ics.uci.edu/∼mlearn/mlrepository.html]. University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

[6] J.-L. Deneubourg, S. Goss, N. Franks, Sendova-Franks A., Detrain C., and L. Chretien. The dynamics of collective sorting: Robot-like ants and ant-like robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 356–365. MIT Press, 1991.

[7] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[8] Richard J. Elzinga. *Fundamentals of entomology*. Prentice Hall, 2000.

[9] Brian S. Everitt, Sabine Landau, and Morven Leese. *Cluster Analysis*. Arnold, a member of the Hodder Headline Group, fourth edition, 2001.

[10] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

[11] Pierre-Paul Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la theorie de la stigmergie: Essai d'interpretation des termites constructeurs. *Insectes Sociaux*, 6(1):41–83, 1959.

[12] Julia Handl, Joshua Knowles, and Marco Dorigo. On the performance of ant-based clustering. Design and application of hybrid intelligent systems. Frontiers in Artificial Intelligence and Applications 104. IOS Press, Amsterdam, The Netherlands, 2003.

[13] Julia Handl, Joshua Knowles, and Marco Dorigo. Strategies for the increased robustness of ant-based clustering. In *Self-Organising Applications: Issues, challenges and trends. LNCS 2977*, pages 90–104. Springer, 2003.

[14] Julia Handl, Joshua Knowles, and Marco Dorigo. Ant-based clustering and topographic mapping. *Artificial Life*, 11(2), 2005. In press.

[15] Julia Handl and Bernd Meyer. Improved ant-based clustering and sorting in a document retrieval interface. In *Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature (PPSN VII). LNCS 2439*, pages 913–923. Springer, 2002.

[16] Owen Holland and Chris Melhuish. Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life*, 5:173–202, 1999.

[17] Frank Hoppner, Frank Klawonn, Rudolf Kruse, and Thomas Runkler. *Fuzzy Cluster Analysis. Methods for classification, data analysis and image recognition*. Wiley & Sons, Ltd., 1999.

[18] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[19] T. Kohonen, T.S. Huang, and M.R. Schroeder. *Self-organizing maps*. Springer Verlag, third edition, 2000.

[20] C.R. Kube and H. Zhang. Collective robotic intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 460–468, 1992.

[21] Kristina Lerman and Aram Galstyan. A general methodology for mathematical analysis of multi-agent systems. Technical report, Information Sciences Institute, Univ. of Southern California, 2001.

[22] Erick D. Lumer and Baldo Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, pages 501–508. MIT Press, 1994.

[23] P.S. Nagpaul. Non-parametric measures of bivariate relationships. http://www.unesco.org/webworld/idams/advguide/chapt4_2.htm, 2004.

[24] M.J. Pearce. *Termites. Biology and Pest Management*. CAB International Publishing, 1997.

[25] Edward O. Wilson. *The Insect Societies*. The Belkap Press of Harvard University Press, 1971.

Vita

Marco Antonio Montes de Oca Roldán was born in Mexico City, Mexico on April 6th, 1979. He was awarded a B.S. in Computer Engineering from the School of Computer Engineering of the National Polytechnic Institute in 2001. By that time, he was already working for the National Center for the Arts, and continued working there for two more years after his graduation. He also worked for the Public Ministry of Education from 2001 to 2003.

In August 2003, he enrolled in the graduate program in intelligent systems at the Monterrey Institute of Technology. He is expecting to start his PhD at the Université Libre de Bruxelles in September 2005.

E-mail Address: **marco.montes.de.oca@gmail.com**

This thesis was typesetted with LaTeX[1] by Marco Antonio Montes de Oca Roldán.

[1]The macros package, `ITESMtesis.sty`, that was used to format this thesis was written by Dr. Horacio Martínez Alfaro <`hma@itesm.mx`>, Associate Professor of the Center for Intelligent Systems. Monterrey Institute of Technology, Monterrey Campus.