UNIVERSITÀ DELLA CALABRIA

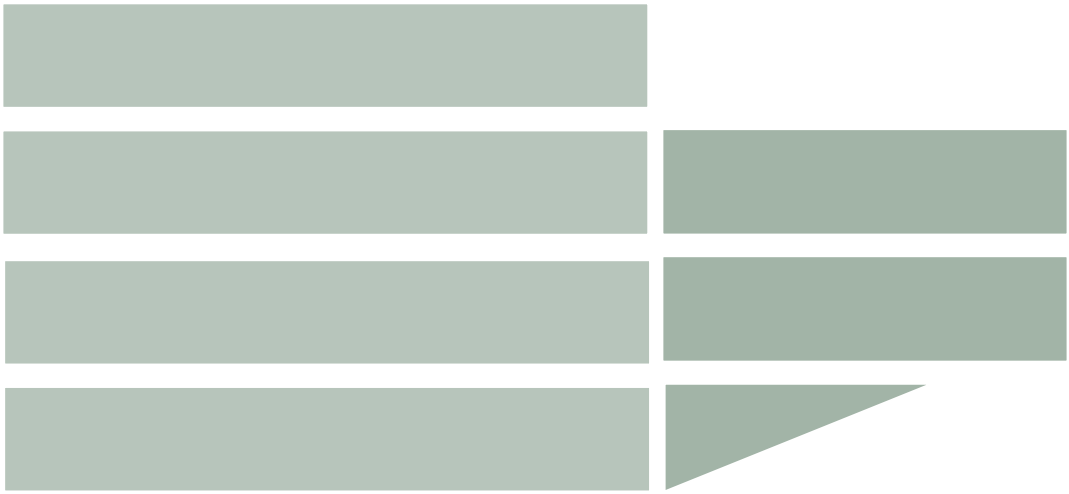
Dipartimento di Elettronica,
Informatica e Sistemistica

Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica
XIX ciclo

Tesi di Dottorato

Swarm-Based Algorithms for Decentralized Clustering and Resource Discovery in Grids

Agostino Forestiero



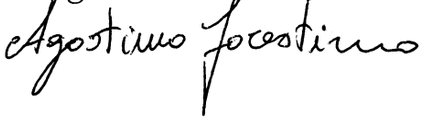
UNIVERSITÀ DELLA CALABRIA
Dottorato di Ricerca in
Ingegneria dei Sistemi e Informatica

XIX ciclo

Tesi di Dottorato

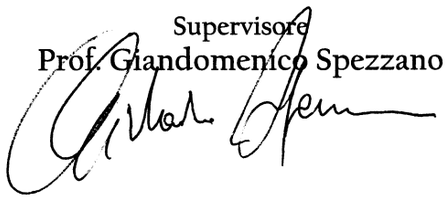
Swarm-Based Algorithms for Decentralized Clustering and Resource Discovery in Grids

Agostino Forestiero



Coordinatore
Prof. Domenico Talia

Supervisore
Prof. Giandomenico Spezzano



DIPARTIMENTO DI ELETTRONICA, INFORMATICA E SISTEMISTICA
Settore Scientifico Disciplinare: ING-INF/05

To my wife and partner, Sara

Acknowledgments

I wish to express my sincere gratitude to my thesis advisor Ing. Giandomenico Spezzano for guiding me through every step of the thesis and providing me direction and insight on numerous occasions during the course of this work. I would like to thank my colleagues (friends) Ing. Gianluigi Folino and Ing. Carlo Mastroianni for their precious support and patience. Special thanks go to my parents and my brother Francesco. I would also like to thank my wife, Sara, for her loving support and appreciation. Moreover, I cannot forget to thank my friends and colleagues at ICAR-CNR institute, in particularly Giuseppe Papuzzo and Oreste Verta.

Contents

1

Introduction

1.1 Background and Motivations

In recent times, distributed computing has considerably changed due to new developments of information technology. New environments has emerged such as massively large-scale, wide-area computer networks and mobile ad-hoc networks. These environments represent an enormous potential for future applications: they enable communication, storage, and computational services to be built in a bottom-up fashion (e.g. *peer-to-peer system* and *Grid Computing*). However, these environments present a few problems because they can be extremely dynamic and unreliable. Traditional approaches to the design of distributed systems which assume reliable components or based on central and explicit control are not applicable for these environments. Furthermore, central control introduces a single-point-of-failure which should be avoided whenever possible [4]. In order to tackle these issues, an artificial intelligence technique based on the study of collective behavior in decentralized, self-organized systems, namely **Swarm Intelligence** [7], appears particularly suitable. Swarm Intelligence systems are typically made up of a population of simple agents interacting locally with one another and with their environment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. Examples of systems like these can be found in nature, including ant colonies, bird flocking, animal herding, bacteria molding and fish schooling. Swarms offer several advantages compared to traditional systems:

- robustness,
- flexibility,
- scalability,
- adaptability,
- suitability for analysis.

Simple agents are less likely to fail than more complex ones. If they do fail, they can be pulled out entirely or replaced without significantly impacting the overall performance of the system. Distributed systems are therefore, tolerant of agent error and failure. They are also highly scalable: increasing the number of agents or task size does not greatly affect performance. The inherent parallelism and scalability make the swarm-based algorithms very suitable to be used in environment whose structure dynamically changes. In systems using central control, the high communications and computational costs required to coordinate agent behavior limit the size of the system to at most a few dozen agents. The simplicity of agent's interactions with other agents make swarms amenable to quantitative mathematical analysis. There are multiple examples of complex collective behavior among social insects: trail formation in ants, hive building by bees and mound construction by termites are just few examples. The apparent success of these organisms has inspired computer scientists and engineers to design algorithms and distributed problem-solving systems modelled after them [8], especially for systems which are characterized by decentralized control, large scale and extreme dynamism of their operating environment as peer-to-peer system, Grid Computing, etc..

1.2 Main Contributions of the Thesis

In this thesis, some novel algorithms based on swarm intelligent paradigm are proposed. In particular, the swarm agents, was exploited to tackle the following issues:

- **P2P Clustering**. A swarm-based algorithm is used to cluster distributed data in a peer-to-peer environment through a small worlds topology. Moreover, to perform spatial clustering in every peer, two novel algorithms are proposed. They are based on the stochastic search of the flocking algorithm and on the main principles of two popular clustering algorithms, DBSCAN and SNN.
- **Resource discovery in Grids**. An approach based on ant systems is exploited to replicate and map Grid services information on Grid hosts according to the semantic classification of such services. To exploit this mapping, a semi-informed resource discovery protocol which makes use of the ants' work has been achieved. Asynchronous query messages (agents) issued by clients are driven towards "representative peers" which maintain information about a large number of resources having the required characteristics.

1.3 Thesis Organization

The thesis organized as follows:

Chapter 2 I present some of the critical notions of Swarm Intelligence and the research work that has addressed them. These notions, organized around the concept of problem-solving which is one of the most overall characteristics that an SI exhibits.

Chapter 3 presents an adaptive flocking algorithm based on the biology-inspired paradigm of a flock of birds. We used swarming agents (SA), i.e. a population of simple agents interacting locally with each other and with the environment. They provide models of distributed adaptive organization and they are useful to solve difficult optimization, classification, distributed control problems, etc...

Chapter 4 describes P-SPARROW, a algorithm for distributed clustering of data in peer-to-peer environments combining a smart exploratory strategy based on a flock of birds with a density-based strategy to discover clusters of arbitrary shape and size in spatial data.

Chapter 5 shows as an Ant-based Replication and MApping Protocol (ARMAP) is used to disseminate resource information by a decentralized mechanism, and its effectiveness is evaluated by means of an entropy index. Information is disseminated by agents - ants - that traverse the Grid by exploiting P2P interconnections among Grid hosts. A mechanism inspired by real ants' pheromone is used by each agent to autonomously drive its behavior on the basis of its interaction with the environment.

Chapter 6 proposes a semi-informed discovery protocol (namely ARDIP, Ant-based Resource DIscovery Protocol) to exploit the logical resource organization achieved by ARMAP.

2

Swarm-Based Systems

2.1 Introduction

The term *Swarm*, in a general sense, refers to any such loosely structured collection of interacting agents. The classic example of a swarm is a swarm of bees, but the metaphor of a swarm can be extended to other systems with a similar architecture. An ant colony can be thought of as a swarm whose individual agents are ants, a flock of birds is a swarm whose agents are birds, traffic is a swarm of cars, a crowd is a swarm of people, an immune system is a swarm of cells and molecules, and an economy is a swarm of economic agents. A high-level view of a swarm suggests that the N agents in the swarm are cooperating to achieve some goal. This apparent *collective intelligence* seems to emerge from what are often large groups of relatively simple agents. The agents use simple local rules to govern their actions and via the interactions of the entire group, the swarm achieves its objectives. A type of *self-organization* emerges from the collection of actions of the group. Swarm intelligence is the emergent collective intelligence of groups of simple autonomous agents. Here, an autonomous agent is a subsystem that interacts with its environment, which probably consists of other agents, but acts relatively independently from all other agents. The autonomous agent does not follow commands from a leader, or some global plan.

For example, for a bird to participate in a flock, it only adjusts its movements to coordinate with the movements of its flock mates, typically its neighbors that are close to it in the flock. A bird in a flock simply tries to stay close to its neighbors, but avoid collisions with them. Each bird does not take commands from any leader bird since there is no lead bird. Any bird can fly in the front, center and back of the swarm. Swarm behavior helps birds take advantage of several things including protection from predators (especially for birds in the middle of the flock), and searching for food (essentially each bird is exploiting the eyes of every other bird). Researchers try to examine how collections of animals, such as flocks, herds and schools, move in a way that appears to be orchestrated. In 1987, Reynolds created a boid model, which

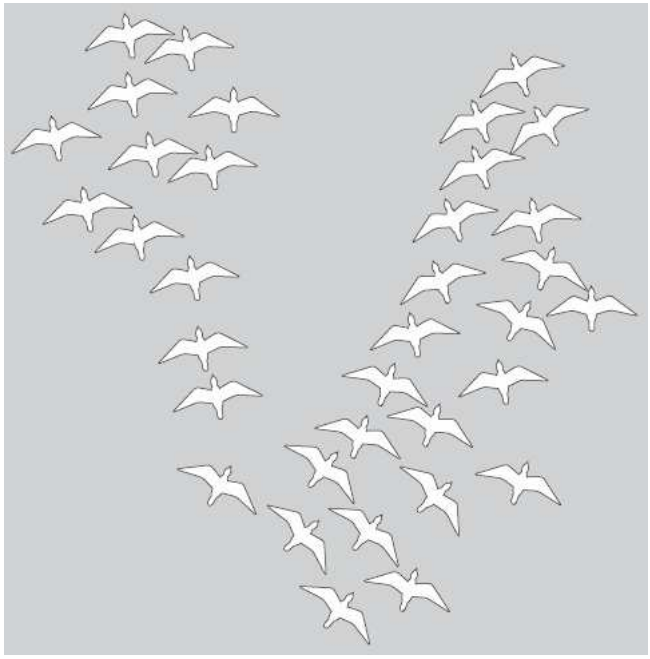


Fig. 2.1. Reynolds showed that a realistic bird flock could be programmed by implementing three simple rules: match your neighbors velocity, steer for the perceived center of the flock, and avoid collisions.

is a distributed behavioral model, to simulate on a computer the motion of a flock of birds [103] (see in Fig. 2.1). Each boid is implemented as an independent actor that navigates according to its own perception of the dynamic environment. A boid must observe the following rules. A boid must:

- move away from boids that are too close;
- follows direction and velocity of the flock;
- move towards boids that are too distant.

The swarm behavior of the simulated flock is the result of the dense interaction of the relatively simple behaviors of the individual boids. Swarm robotics is currently one of the most important application areas for swarm intelligence. Swarms provide the possibility of enhanced task performance, high reliability (fault tolerance), low unit complexity and decreased cost over traditional robotic systems. They can accomplish some tasks that would be impossible for a single robot to achieve. Swarm robots can be applied to many fields, such as flexible manufacturing systems, spacecraft, inspection/maintenance, construction, agriculture, and medicine work [7].

2.2 Self-Organization

Self-Organization is a set of dynamical mechanism whereby structures appear at the global level of system from interactions among its lower-level components. The rules specifying the interactions among the system's constituent unit are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence. For example, the emerging structures in the case of foraging in ants include spatiotemporally organized networks of pheromone trails. Self-organization relies on four basic ingredients:

- *Positive feedback* amplifies a certain behavior, e.g. bees may recruit other bees to follow them to good food sources.
- *Negative feedback* on the other hand limits a behavior, e.g. if a food source is too crowded.
- *The amplification of fluctuations* enables discovery of a new collective behavior resulting from random walks or errors of individuals.
- *Multiple interactions* between individuals are necessary for a new behavior to be adopted by the swarm.

When a given phenomenon is self-organized, it can usually be characterized by a few properties:

1. The creation of spatiotemporal structures in an initially homogeneous medium. Such structures include nest architectures, foraging trail, or social organization. For example, a characteristic well-organized pattern develops on the combos honeybee colonies, i.e. three concentric regions.
2. The possible coexistence of several stable states (multistability). Because structures emerge by amplification of random deviations, any such deviation can be amplified, and the system converges to one among several possible stable state, depending on initial conditions.
3. The existence of bifurcations when some parameters are varied. The behavior of a self-organized system changes dramatically at bifurcations.

2.2.1 Stigmergy

Self-Organization in social insect often required interactions among insect: such interactions can be direct or indirect. Direct interactions are the "obvious" interactions: antennation, trophallaxis (food or liquid exchange), mandibular contact, visual contact, chemical contact (the odor of nearby nestmates), etc. Indirect interactions are more suitable: two individuals interact indirectly when one of them modifies the environment and the other respond to the new environment at a later time. The indirect form of communication among individuals was first described by French entomologist Pierre-Paul Grass in the 1950s and denominated **stigmergy** (from the Greek sigma: sting and ergon: work) [40]. Stigmergy has helped researchers understand the connection

between individual and collective behaviour. The basic principle of stigmergy is extremely simple:

- Traces left and modifications made by individuals in their environment may feed back on them.

The colony records its activity in part in the physical environment and uses this record to organize the collective behaviour. Various forms of storage are used:

- gradients of pheromones;
- material structures;
- spatial distribution of colony elements.

Such structures materialize the dynamics of the colonys collective behaviour and constrain the behaviour of the individuals through a feedback loop. Holland and Beckers distinguish between *cue-based* and *sign-based* stigmergy. In cue-based stigmergy, the change in the environment simply provides a cue for the behavior of other actors, while in sign-based stigmergy the environmental change actually sends a signal to other actors. Termite arch-building contains both kinds of stigmergy, with pheromones providing signals while the growing pillars provide cues. Ant corpse-piling has been described as a cue-based stigmergic activity. When an ant dies in the nest, the other ants ignore it for the first few days, until the body begins to decompose. The release of chemicals related to oleic acid stimulates a passing ant to pick up the body and carry it out of the nest. Some species of ants actually organize cemeteries where they deposit the corpses. If dead bodies of these species are scattered randomly over an area, the survivors will hurry around picking up the bodies, moving them, and dropping them again, until soon all the corpses are arranged into a small number of distinct piles. The piles might form at the edges of an area or on a prominence or other heterogeneous feature of the landscape. Deneubourg [24] have shown that ant cemetery formation can be explained in terms of simple rules. The essence of the rule set is that isolated items should be picked up and dropped at some other location where more items of that type are present. A similar algorithm appears to be able to explain larval sorting, in which larvae are stored in the nest according to their size, with smaller larvae near the center and large ones at the periphery, and also the formation of piles of woodchips by termites. In the latter, termites may obey the following rules:

- If you are not carrying a woodchip and you encounter one, pick it up.
- If you are carrying a woodchip and you encounter another one, set yours down.

Thus a woodchip that has been set down by a termite provides a stigmergic cue for succeeding termites to set their woodchips down. If a termite sets a chip down where another chip is, the new pile of two chips becomes probabilistically more likely to be discovered by the next termite that comes past

carrying a chip, since its bigger. Each additional woodchip makes the pile more conspicuous, increasing its growth more and more in an autocatalytic loop. Stigmergy alone is not sufficient to explain collective intelligence, as it only refers to animal-animal interactions. Therefore, it has to be complemented with an additional mechanism that makes use of these interactions to coordinate and regulate the collective task in a particular way. One of the best examples of this mechanism was studied by Grass: the building behaviour of termites. Grass showed that the coordination and regulation of building activities do not depend on the workers themselves but are mainly achieved by the nest structure: A stimulating configuration triggers a building action of a termite worker, transforming the configuration into another configuration that may trigger in turn another (possibly different) action performed by the same termite or any other termite in the colony. The use of stigmergy is not confined to building structures. It also occurs in cooperative foraging strategies such as trail recruitment in ants, where the interactions between foragers are mediated by pheromones put on the ground in quantities determined by the local conditions of the environment. For example, trail recruitment in ant species are able to select and preferentially exploit the richest food source in the neighbourhood or the shortest path between the nest and a food source: foragers are initially evenly distributed between the two sources, but one of the sources randomly becomes slightly favoured, and this difference may be amplified by recruitment, since the more foragers there are at a given source, the more individuals recruited to that source . Michener describes in [85] many activities in bee colonies that result in nest structures, conditions of brood or stored food, to which other bees respond. Referring to this as indirect social interactions, where the construct is made for other primary objectives, not for signalling, although the information content becomes essential for colony integration. In nectar source decision making in honey bees it is less clear if only direct communication through the recruitment dances of the bees produce the self-organizing behaviour, or if also the indirect communication given by the waiting time for downloading the honey is affecting the collective behaviour [108]. As a consequence of stigmergy and self-organization, complex behaviours which had been explained on the basis of certain rules of interaction among individuals were later accounted for even simpler behaviours in the context of environmental stimuli. Stigmergy seems indeed at the root of several collective behaviours of social insects, especially in their building activities. This is certainly a powerful principle, as social insect constructions are remarkable for their complexity, size and adaptive value. However, it is possible to extend the idea easily to other domains; it can be seen as an even more impressive and general account of how simple systems can produce a range of apparently highly organized and coordinated behaviours and behavioural outcomes, simply by exploiting the influence of the environment.

2.3 Foraging behavior of ants

Ant colony expresses a complex collective behavior providing intelligent solutions to problems such as carrying large items, forming bridges and finding the shortest routes from the nest to a food source. A single ant has no global knowledge about the task it is performing. The ant's actions are based on local decisions and are usually unpredictable. The intelligent behavior naturally emerges as a consequence of the self-organization and indirect communication between the ants. This is what is usually called *Emergent Behavior* or *Emergent Intelligence*. Ants use a signaling communication system based on the deposition of **pheromone** over the path it follows, marking a trail. Pheromone is a hormone produced by ants that establishes a sort of indirect communication among them. Basically, an isolated ant moves at random, but when it finds a pheromone trail there is a high probability that this ant will decide to follow the trail. An ant foraging for food lay down pheromone over its route. When this ant finds a food source, it returns to the nest reinforcing its trail. Pheromone evaporates with passing of the time with evaporation rate Ev (see formula 2.1).Other ants in the proximities are attracted by this substance and have greater probability to start following this trail and thereby laying more pheromone on it.

$$\Phi = Ev * \Phi_{i-1} + \phi_i \tag{2.1}$$

This process works as a positive feedback loop system because the higher the intensity of the pheromone over a trail, the higher the probability of an ant start traveling through it.This elementary behavior of real ants can be used to explain how they can find the shortest path which reconnects a broken line after the sudden appearance of an unexpected obstacle has interrupted the initial path (see Fig. 2.2). In fact, once the obstacle has appeared, those ants which are just in front of the obstacle cannot continue to follow the pheromone trail and therefore they have to choose between turning right or left. In this situation we can expect half the ants to choose to turn right and the other half to turn left. The very same situation can be found on the other side of the obstacle. It is interesting to note that those ants which choose, by chance, the shorter path around the obstacle will more rapidly reconstitute the interrupted pheromone trail compared to those which choose the longer path. Hence, the shorter path will receive a higher amount of pheromone in the time unit and this will in turn cause a higher number of ants to choose the shorter path. Due to this positive feedback (autocatalytic) process, very soon all the ants will choose the shorter path. The most interesting aspect of this autocatalytic process is that finding the shortest path around the obstacle seems to be an emergent property of the interaction between the obstacle shape and ants distributed behavior: Although all ants move at approximately the same speed and deposit a pheromone trail at approximately the same rate, it is a fact that it takes longer to contour obstacles on their longer side than on their shorter side which makes the pheromone trail accumulate quicker on the shorter side.

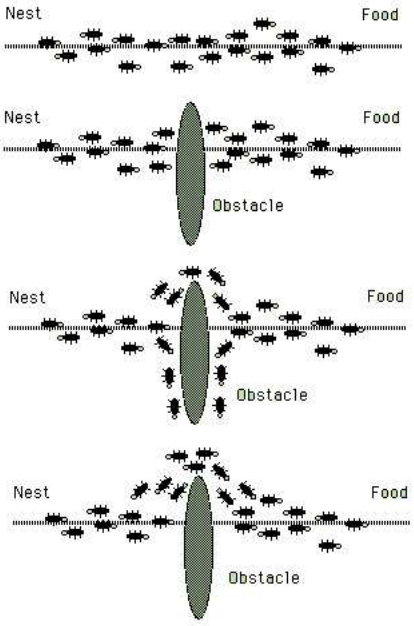


Fig. 2.2. Pheromones accumulate on the shorter path because any ant that sets out on that path returns sooner.

It is the ants preference for higher pheromone trail levels which makes this accumulation still quicker on the shorter path.

2.3.1 Ant colony optimization

"Ant colony optimization" [8] is based on the observation that ants will find the shortest path around an obstacle separating their nest from a target such as a piece of candy simmering on a summer sidewalk. As ants move around they leave pheromone trails, which dissipate over time and distance. The pheromone intensity at a spot, that is, the number of pheromone molecules that a wandering ant might encounter,is higher either when ants have passed over the spot more recently or when a greater number of ants have passed over the spot. Thus ants following pheromone trails will tend to congregate simply from the fact that the pheromone density increases with each additional ant that follows the trail. Ants meandering from the nest to the candy and back will return more quickly, and thus will pass the same points more frequently, when following a shorter path. Passing more frequently, they will lay down a denser pheromone trail. As more ants pick up the strengthened trail, it becomes increasingly stronger (see in Fig. 2.2). In computer adaptation of these behaviors [8] let a population of "ants" search a traveling salesman map stochastically, increasing the probability of following a connection between two

cities as a function of the number of other simulated ants that had already followed that link. By exploitation of the positive feedback effect, that is, the strengthening of the trail with every additional ant, this algorithm is able to solve quite complicated combinatorial problems where the goal is to find a way to accomplish a task in the fewest number of operations. Research on live ants has shown that when food is placed at some distance from the nest, with two paths of unequal length leading to it, they will end up with the swarm following the shorter path. If a shorter path is introduced, though, for instance, if an obstacle is removed, they are unable to switch to it. If both paths are of equal length, the ants will choose one or the other. If two food sources are offered, with one being a richer source than the other, a swarm of ants will choose the richer source; if a richer source is offered after the choice has been made, most species are unable to switch, but some species are able to change their pattern to the better source. If two equal sources are offered, an ant will choose one or the other arbitrarily.The movements of ants are essentially random as long as there is no systematic pheromone pattern; activity is a function of two parameters, which are the strength of pheromones and the attractiveness of the pheromone to the ants. If the pheromone distribution is random, or if the attraction of ants to the pheromone is weak, then no pattern will form. On the other hand, if a too-strong pheromone concentration is established, or if the attraction of ants to the pheromone is very intense, then a suboptimal pattern may emerge, as the ants crowd together in a sort of pointless conformity. In real and simulated examples of insect accomplishments, we see optimization of various types, whether clustering items or finding the shortest path through a landscape, with certain interesting characteristics. None of these instances include global evaluation of the situation: an insect can only detect its immediate environment. Optimization traditionally requires some method for evaluating the fitness of a solution, which seems to require that candidate solutions be compared to some standard, which may be a desired goal state or the fitness of other potential solutions.

2.4 Particle swarm optimization

Particle swarm optimization (PSO) was originally developed by Eberhart and Kennedy in 1995 [64]. It is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each timestep. Over time, particles are accelerated towards those particles within their communication grouping which have better *fitness* values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large number of members that make up the particle swarm make the technique

impressively resilient to the problem of local minima. PSO was inspired by the social behavior of a flock of birds. In the PSO algorithm, the birds in a flock are symbolically represented as particles. These particles can be considered as simple agents "flying" through a problem space. A particles location in the multi-dimensional problem space represents one solution for the problem. When a particle moves to a new location, a different problem solution is generated. This solution is evaluated by a fitness function that provides a quantitative value of the solutions utility. The velocity and direction of each particle moving along each dimension of the problem space will be altered with each generation of movement. In combination, the particles personal experience, P_{id} and its neighbors' experience, P_{gd} influence the movement of each particle through a problem space. The random values $rand_1$ and $rand_2$ are used for the sake of completeness, that is, to make sure that particles explore a wide search space before converging around the optimal solution. The values of c_1 and c_2 control the weight balance of P_{id} and P_{gd} in deciding the particles next movement velocity. At every generation, the particles new location is computed by adding the particles current velocity, v_{id}, to its location, x_{id}. Mathematically, given a multi-dimensional problem space, the i_{th} particle changes its velocity and location according to the following equations [64]:

$$v_{id} = w * v_{id} + c_1 * rand_1 * (p_{id} - x_{id}) + c2 * rand_2 * (p_{gd} - x_{id}) \tag{2.2}$$

$$x_{id} = x_{id} + v_{id} \tag{2.3}$$

where **w** denotes the inertia weight factor; p_{id} is the location of the particle that experiences the best fitness value; p_{gd} is the location of the particles that experience a global best fitness value; c_1 and c_2 are constants and are known as acceleration coefficients; d denotes the dimension of the problem space; $rand_1$, $rand_2$ are random values in the range of (0, 1).

2.5 Multiagent Systems

Multiagent systems are computational systems in which two or more agents interact or work together to perform some set of tasks or to satisfy some set of goals. These systems may be comprised of homogeneous or heterogeneous agents. An agent in the system is considered a locus of problem-solving activity, it operates asynchronously with respect to other agents, and it has a certain level of autonomy. Agent autonomy relates to an agents ability to make its own decisions about what activities to do, when to do them, what type of information should be communicated and to whom, and how to assimilate the information received. Autonomy can be limited by policies built into the

agent by its designer, or as a result of an agent organization dynamically coming to an agreement that specific agents should take on certain roles or adopt certain policies for some specified period. Closely associated with agent autonomy is agent adaptability the more autonomy an agent possesses the more adaptable it is to the emerging problem solving and network context. The degree of autonomy and the range of adaptability are usually associated with the level of intelligence/sophistication that an agent possesses. Agents may also be characterized by whether they are benevolent (cooperative) or self-interested. Cooperative agents work toward achieving some common goals, whereas self-interested agents have distinct goals but may still interact to advance their own goals. In the latter case, self-interested agents may, by exchanging favors or currency, coordinate with other agents in order to get those agents to perform activities that assist in the achievement of their own objectives. For example, in a manufacturing setting where agents are responsible for scheduling different aspects of the manufacturing process, agents in the same manufacturing company would behave in a cooperative way while agents representing two separate companies where one company was outsourcing part of its manufacturing process to the other company would behave in a selfinterested way. Scientific research and practice in multiagent systems, which in the past has been called Distributed Artificial Intelligence (DAI), focuses on the development of computational principles and models for constructing, describing, implementing and analyzing the patterns of interaction and coordination in both large and small agent societies. Multiagent systems research brings together a diverse set of research disciplines and thus there is a wide range of ideas currently being explored. Multiagent systems over the past few years have come to be perceived as crucial technology not only for effectively exploiting the increasing availability of diverse, heterogeneous, and distributed on-line information sources, but also as a framework for building large, complex, and robust distributed information processing systems which exploit the efficiencies of organized behavior. Multiagent systems also provide a powerful model for computing , in which networks of interacting, real-time, intelligent agents seamlessly integrate the work of people and machines, and dynamically adapt their problem solving to effectively deal with changing usage patterns, resource configurations and available sources of expertise and information. Application domains in which multiagent system technology is appropriate typically have a naturally spatial, functional or temporal decomposition of knowledge and expertise. By structuring such applications as a multiagent system rather than as a single agent, the system will have the following advantages:

- speed-up due to concurrent processing;
- less communication bandwidth requirements because processing is located nearer the source of information;
- more reliability because of the lack of a single point of failure;

- improved responsiveness due to processing, sensing and effecting being co-located;
- easier system development due to modularity coming from the decomposition into semiautonomous agents.

Examples of application domains that have used a multiagent approach include:

- Distributed situation assessment, which emphasizes how (diagnostic) agents with different spheres of awareness and control (network segments) should share their local interpretations to arrive at consistent and comprehensive explanations and responses (e.g., network diagnosis [112]; information gathering on the Internet [22] [92]; distributed sensor networks [13].
- Distributed resource scheduling and planning, which emphasizes how (scheduling) agents (associated with each work cell) should coordinate their schedules to avoid and resolve conflicts over resources, and to maximize system output (e.g., factory scheduling [90]; network management [2]; and intelligent environments [50]).
- Distributed expert systems, which emphasize how agents share information and negotiate over collective solutions (designs) given their different expertise and solution criteria (e.g., concurrent engineering [72]; network service restoral [55]).

The next generation of applications will integrate characteristics of each of these generic domains. The need for a multiagent approach can also come from applications in which agents represent the interests of different organizational entities (e.g., electronic commerce and enterprise integration [6]). Other emerging uses of multiagent systems are in layered systems architectures in which agents at different layers need to coordinate their decisions (e.g., to achieve appropriate configurations of resources and computational processing), and in the design of survivable systems in which agents dynamically reorganize to respond to changes in resource availability, software and hardware malfunction, and intrusions. In general, multiagent systems provide a framework in which both the inherent distribution of processing and information in an application and the complexities that come from issues of scale can be handled in a natural way [74]. There are two ways in order to design the multi-agent systems :

- the traditional paradigm, based on deliberative agents and (usually) central control;
- the swarm paradigm, based on simple agents and distributed control;

In the past, researchers in the Artificial Intelligence and related communities have, for the most part, operated within the first paradigm. They focused on making the individual agents, be they software agents or robots, smarter and more complex by giving them the ability to reason, negotiate and plan action. In these deliberative systems complex tasks can be done either individually or collectively. If collective action is required to complete some

task, a central controller is often used to coordinate group behavior. Swarm Intelligence represents an alternative approach to the design of multi-agent systems. Swarms are composed of many simple agents. These systems are self-organizing, meaning collective behavior emerges from local interactions among agents and between agents and the environment, there is no central controller.

3

Approximate Clustering by Adaptive Flock

3.1 Introduction

Clustering data is the process of grouping similar objects according to their distance, connectivity, or relative density in space [44]. There are a large number of algorithms for discovering natural clusters in a data set, but they are usually implemented in a centralized way. These algorithms can be classified into partitioning methods [62], hierarchical methods [61], density-based methods [105], grid-based methods [123]. Han, Kamber and Tung's paper [45] is a good introduction to this subject. Many of these algorithms work on data contained in a file or database. In general, clustering algorithms focus on creating good compact representation of clusters and appropriate distance functions between data points. For this purpose, they generally need one or more parameters chosen by a user that indicate the characteristics of the expected clusters. Centralized clustering is problematic if we have large data to explore or data are widely distributed. Parallel and distributed computing is expected to relieve current mining methods from the sequential bottleneck, providing the ability to scale to massive datasets, and improving the response time. Achieving good performance on today's high performance systems is a non-trivial task. The main challenges include synchronization and communication minimization, work-load balancing, finding good data decomposition, etc. Some existing centralized clustering algorithms have been parallelized and the results have been encouraging. Centralized schemes require high level of connectivity, impose a substantial computational burden, are typically more sensitive to failures than decentralized schemes, and are not scalable. Recently, innovative algorithms based on biological models [24] [70] [79] [89] have been introduced to solve the clustering problem in a decentralized fashion. These algorithms are characterized by the interaction of a large number of simple agents that sense and change their environment locally. Furthermore, they exhibit complex, emergent behavior that is robust with respect to the failure of individual agents. Ant colonies, flocks of birds, termites, swarms of bees etc. are agent-based insect models that exhibit a collective intelligent behavior

(swarm intelligence) [8]. SI models have many features in common with Evolutionary Algorithms (EA). Like EA, SI models are population-based and the system is initialized with a population of individuals (i.e., potential solutions). These individuals are then manipulated over many iteration steps by mimicking the social behavior of insects or animals, in an effort to find the optima in the problem space. Unlike EAs, SI models do not explicitly use evolutionary operators such as crossover and mutation. A potential solution simply 'flies' through the search space by modifying itself according to its past experience and its relationship with other individuals in the population and the environment. In these models, the emergent collective behavior is the outcome of a process of self-organization, in which insects are engaged through their repeated actions and interaction with their evolving environment. Intelligent behavior frequently arises through indirect communication between the agents using the principle of stigmergy [40]. This mechanism is a powerful principle of cooperation in insect societies. According to this principle, an agent deposits something in the environment that makes no direct contribution to the task being undertaken but is used to influence the subsequent behavior that is task related. The advantages of SI are twofold. Firstly, it offers intrinsically distributed algorithms that can use parallel computation quite easily. Secondly, these algorithms show a high level of robustness to change by allowing the solution to dynamically adapt itself to global changes by letting the agents self-adapt to the associated local changes. In this section, I present a new algorithm based on a flock of birds that move together in a complex manner using simple local rules, to explore spatial data for searching interesting objects. The flocking algorithm, inspired by the principles of Macgill [81], was used to design two novel clustering algorithms based on the main principles of two popular clustering methods: DBSCAN and SNN. We consider clustering as a search problem in a multi-agent system in which individuals agents have the goal of finding specific elements in the search space, represented by a large data set of tuples, by walking efficiently through this space. Our methods take advantage of the parallel search mechanism a flock implies, by which if a member of a flock finds an area of interest, the mechanics of the flock will draw other members to scan that area in more detail. Our algorithms select interesting subsets of tuples without inspecting the whole search space guaranteeing a fast placing of points correctly in the clusters. We have applied this strategy as a data reduction technique to perform efficiently approximate clustering [67]. In the algorithm, each agent can use hierarchical, partitioned, density-based and grid-based clustering methods to discovery if a tuple belongs to a cluster. Particle Swarm Optimization [64], inspired to the behavior of flocks of birds, school of fish, etc.., is a population based optimization tool, quite different from our approach. PSO consists of a swarm of particles, each of them representing the solution of an optimization problem, moving in the problem search space. At the beginning, each particle has a random position and velocity. A particle moves on the basis of its own experience (its best past position) and of the best of particles in the swarm. Note that, not differently

from our model, variants of PSO consider the best of a local neighborhood instead of all the swarm. Furthermore, the best particle is an attractor for the other particles as the red birds of our algorithm are the attractors for the other birds. However, the similarities with our model ended here since our flock moves for searching representative points and merge them in order to find clusters, while each PSO particle represents itself a solution of a problem and as a consequence it is less flexible in searching clusters (see related works section for more details). Moreover, in the flock algorithm, there is a more strict interrelation, while particles only look at the best particle in the group. Then PSO does not use any repulsion force to avoid uninteresting zones, while our white agents are repulsive for the others.

To better illustrate the usefulness of the method, we present two algorithms: SPARROW (SPAtial ClusteRing AlgoRithm thrOugh SWarm Intelligence) and SPARROW-SNN (Shared Nearest-Neighbor similarity). SPARROW combines the flocking algorithm with the density-based DBSCAN algorithm [32]. SPARROW-SNN couples the flocking algorithm with a shared nearest neighbor (SNN) cluster algorithm [31] to discover clusters with differing sizes, shapes and densities in noise, high dimensional data. We have built a SWARM [86] simulation of both algorithms to investigate the interaction of the parameters that characterize them.

The rest of this chapter is organized as follows. Section 3.2 presents the multi agent adaptive flocking algorithm, first introducing the classical model of Reynolds and then the modified behavioral rules of our algorithm that add an adaptive behavior. Section 3.3 shows how the stochastic search of the adaptive flocking algorithm can be used as a basis for clustering spatial data, combining our strategy respectively with DBSCAN and SNN to constitute the SPARROW and SPARROW-SNN algorithms. Section 3.4 presents the obtained results for both the approximate clustering algorithms. Section 3.5 presents an entropy-based model to theoretically explain the behavior of the algorithm. Finally, section 3.6 discusses some related works concerning clustering algorithms based on the Swarm Intelligence paradigm.

3.2 Flocking algorithm

In this section, we will present a multi-agent stochastic search based on a classical flocking algorithm that has the advantage of being easily implementable on parallel and distributed machines and is robust compared to the failure of individual agents. We first introduce the Reynolds' flock model that describes the standard movement rules of birds. Then, we illustrate the modified behavioral rules of the swarm agents that, adding an adaptive behavior, make the flock more effective in searching points having some desired properties in the space.

3.2.1 The Reynolds' model

The flocking algorithm was originally proposed by Reynolds [103] as a method for mimicking the flocking behavior of birds on a computer both for animation and as a way to study emergent behavior. Flocking is an example of emergent collective behavior: there is no leader, i.e., no global control. Flocking behavior emerges from the local interactions. In the flock algorithm each agent has direct access to the geometric description of the whole scene, but reacts only to flock mates within a certain small radius. The basic flocking model consists of three simple steering behaviors:

Separation gives an agent the ability to maintain a certain distance from others nearby. This prevents agents from crowding too closely together, allowing them to scan a wider area.

Cohesion supplies an agent the ability to cohere (approach and form a group) with other nearby agents. Steering for cohesion can be computed by finding all agents in the local neighborhood and computing the average position of the nearby agents. The steering force is then applied in the direction of that average position.

Alignment gives an agent the ability to align with other nearby characters. Steering for alignment can be computed by finding all agents in the local neighborhood and averaging together the 'heading' vectors of the nearby agents.

3.2.2 Searching objects in spatial data

Different techniques can be used to cope with the problem of searching interesting object in spatial data. We introduced a multi-agent adaptive algorithm able to discover these points in parallel. Our algorithm uses a modified version of standard flocking algorithm, described above, that incorporates the capacity for learning that we can find in many social insects. In this algorithm, the agents are transformed into *hunters* with a foraging behavior that allow them to explore efficiently spatial data. Our algorithm starts with a fixed number of agents that occupy a randomly generated position in this space. Each agent moves around the spatial data testing the neighborhood of each location in order to verify if a point can have some desired properties. Each agent follows the rules of movement described in Reynolds' model. In addition, our model considers four different kinds of agents, classified on the basis of some properties of data in their neighborhood. Different agents are characterized by a different color: red, revealing interesting patterns in the data, green, a medium one, yellow, a low one, and white, indicating a total absence of patterns.

The main idea behind our approach is to take advantage of the colored agent in order to explore more accurately the most interesting regions (signaled by the red agents) and avoid the ones without interesting points (signaled by the white agents). Red and white agents stop moving in order to signal this type of regions to the others, while green and yellow ones fly to

find more dense zones. Indeed, each flying agent computes its heading by taking the weighted average of alignment, separation and cohesion (as illustrated in figure 3.1). The following are the main features which make our model

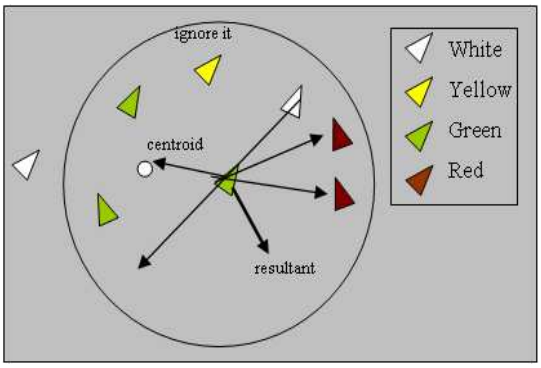


Fig. 3.1. Computing the direction of a green agent.

different from Reynolds' model:

- *Alignment* and *cohesion* do not consider yellow agents, since they move in a not very attractive zone.
- *Cohesion* is the resultant of the heading towards the average position of the green flockmates (centroid), of the attraction towards reds, and of the repulsion from whites, as illustrated in figure 3.1.
- A *separation* distance is maintained from all the agents, apart from their color.

Now, we give a more formal description of our extension of the flocking algorithm. Consider a multidimensional space with dimension d. Each bird k can be represented by its position in this space $x_{k1}, x_{k2}, \ldots, x_{kd}$, by its direction $dir_{k1}, dir_{k2}, \ldots, dir_{kd}$, where dir_{ki} represent the component along the axis i of the direction of bird k and by a color (white, yellow, green or red), indicating the type of the bird. Let B be set of the birds and $dist\_max$ and $dist\_min$ respectively the radius indicating the limited sight of the birds and the minimum distance that must be maintained among them. We denoted by $Neigh(k)$, the neighborhood of bird k, i.e. the set $\{\alpha \in B \mid dist(k, \alpha) \leq dist\_max\}$, that is the set of the birds visible from the bird k and by $Neigh(col, k)$ the set $\{\alpha \in B \mid dist(k, \alpha) \leq dist\_max, color(\alpha) = col\}$.

Each bird moves with speed v_k, depending on the color of the agents (green agents' speed is slower, because they are exploring interesting zones). Then, for each iteration t, the new position of the bird k can be computed as:

$$\forall i = 1 \ldots d \qquad x_{ki}(t+1) = x_{ki}(t) + v_k \times dir_{ki} \tag{3.1}$$

Note also that for each iteration the new direction of the agent is obtained summing the three components of alignment, separation and cohesion (i.e. $dir_{ki} = dir\_al_{ki} - dir\_sep_{ki} + dir\_co_{ki}$). and they can be computed using the following formulas (considering as $dir(a, b)$ the normalized direction of the vector between the bird a and the bird b):

$$dir\_al_{ki} = \frac{1}{|Neigh(green, k)|} \cdot \sum_{\alpha \in Neigh(green,k)} dir_{\alpha i} \tag{3.2}$$

and considering $centr(green, k)$ as the centroid of the green agents in the neighborhood of k with generic coordinate i:

$$\frac{1}{|Neigh(green, k)|} \cdot \sum_{\alpha \in Neigh(green,k)} x_{\alpha i} \tag{3.3}$$

then:

$$dir\_co_{ki} = \Delta + \Phi - \Omega \tag{3.4}$$

where:

$$\Delta = dir(centr(green, k), k)_i \tag{3.5}$$

$$\Phi = \sum_{\alpha \in Neigh(red,k)} dir(\alpha, k)_i \tag{3.6}$$

$$\Omega = \sum_{\alpha \in Neigh(white,k)} dir(\alpha, k)_i \tag{3.7}$$

and:

$$dir\_sep_{ki} = \sum_{\alpha \in Neigh(k), dist(\alpha, kappa) < dist\_min} dir(alpha, k)_i \tag{3.8}$$

In figure 3.2, we summarized the pseudo code of the overall algorithm.

Yellow and green agents will compute their direction, according to the rules previously described, and will move following this direction and with the speed corresponding to their color.

Agents will move towards the computed direction with a speed depending from their color: green agents more slowly than yellow agents since they will explore more interesting regions. An agent will speed up to leave an empty or uninteresting region whereas will slow down to investigate an interesting region more carefully.

The variable speed introduces an adaptive behavior in the algorithm. In fact, agents adapt their movement and change their behavior (speed) on the basis of their previous experience represented from the red and white agents. Red and white agents will stop signaling to the others respectively the interesting and desert regions. Note that, for any agent has become red or white, a new agent will be generated in order to maintain a constant number of agents

```
for i=1 ... MaxIterations
        foreach agent (yellow, green)
                age=age+1;
                if (age > Max_Life)
                        generate_new_agent();die();
                endif
                if (not visited (current_point))
                        property = compute_local_property(current_point);
                        mycolor= color_agent(property);
                endif
        end foreach
        foreach agent (yellow, green)
                dir= compute_dir();
        end foreach
        foreach agent (all)
                switch (mycolor){
                    case yellow, green: move(dir, speed(mycolor)); break;
                    case white: stop(); generate_new_agent(); break;
                    case red: stop(); generate_new_close_agent(); break; }
        end foreach
end for
```

Fig. 3.2. The pseudo-code of the adaptive flocking algorithm.

exploring the data. In the first case (red), the new agent will be generated in a close random point, since the zone is considered interesting, while in the latter it will be generated in a random point over all the space. In case the agent falls in the same position of an older it will be regenerated using the same policy described above.

Note that the color of the agent is assigned on the basis of the desired property of the point in which it falls; the assignment is made on a scale going from white ($property = 0$) to red ($property > threshold$), passing for yellow and green, corresponding to intermediate values.

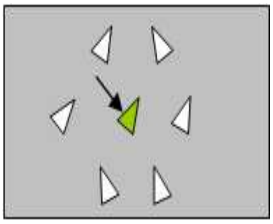


Fig. 3.3. The cage effect.

Yellow and green agents try to avoid the '*cage effect*' (see figure 3.3); in fact, some agents could remain trapped inside regions surrounded by red or white agents and would have no way to go out, wasting useful resources for the exploration. So, a limit was imposed on their life. When their age exceeded a determined value (Max_Life) they die and are regenerated in a new randomly chosen position of the space.

3.3 Clustering spatial data

In this section, we show how the stochastic search of the adaptive flocking described in the previous section can be used as a basis for implementing algorithms for clustering spatial data. Our approach has a number of nice properties. It can be easily implemented on parallel computers and is robust compared to the failure of individual agents. It can also be applied to perform efficiently *approximate clustering* since the points, that are visited and analyzed by the agents, represent a significant (in *ergodic* sense) subset of the entire dataset. The subset reduces the size of the dataset while keeping the accuracy loss as small as possible.

In particular, we implemented SPARROW and SPARROW-SNN that respectively combine our flocking strategy with the DBSCAN heuristics for discovering clusters with differing sizes, shapes in noise data, and with the SNN heuristics in order to discover clusters with differing densities.

In the following, the DBSCAN algorithm and the SPARROW are described, then, SNN and SPARROW SNN are illustrated.

3.3.1 The DBSCAN algorithm

One of the most popular spatial clustering algorithms is DBSCAN, which is a density-based spatial clustering algorithm. A complete description of the algorithm and its theoretical basis is presented in the paper by Ester et al. [32]. In the following, we briefly present the main principles of DBSCAN. The algorithm is based on the idea that all points of a data set can be regrouped into two classes: *clusters* and *noise*. Clusters are defined as a set of dense connected regions with a given radius (*Eps*) and containing at least a minimum number (*MinPts*) of points. Data are regarded as noise if the number of points contained in a region falls below a specified threshold. The two parameters, Eps and MinPts, must be specified by the user and allow to control the density of the cluster that must be retrieved. The algorithm defines two different kinds of points in a clustering: *core points* and *non-core points*. A core point is a point with at least MinPts number of points contained in an Eps-neighborhood of the point. The non-core points in turn are either *border points* if are not core points but are density-reachable from another core point or *noise points* if they are not core points and are not density-reachable from other points. To find the clusters in a data set, DBSCAN starts from an arbitrary point and

retrieves all points with the same density reachable from that point using Eps and MinPts as controlling parameters. A point p is density reachable from a point q, if the two points are connected by a chain of points such that each point has a minimal number of data points, including the next point in the chain, within a fixed radius. If the point is a core point, then the procedure yields a cluster. If the point is on the border, then DBSCAN goes on to the next point in the database and the point is assigned to the noise. DBSCAN builds clusters in sequence (that is, one at a time), in the order in which they are encountered during space traversal. The retrieval of the density of a cluster is performed by successive spatial queries. Such queries are supported efficiently by spatial access methods such as R*-trees.

3.3.2 SPARROW algorithm

SPARROW is a multi-agent adaptive algorithm able to discover clusters in parallel. It uses the modified version of standard flocking algorithm that incorporates the capacity for learning that can find in many social insects. Flocking agents are transformed into hunters with a foraging behavior that allow them to explore the spatial data and to search for clusters.

SPARROW starts with a fixed number of agents that occupy a randomly generated position. Each agent moves around the spatial data testing the neighborhood of each location in order to verify if the point can be identified as a *core* (representative) point. Such a case, a temporary label is assigned to all the points of the neighborhood of the core point. The labels are updated concurrently as multiple clusters take shape. Contiguous points belonging to the same cluster take the label corresponding to the smallest label in the group of contiguous points.

The algorithm follows the same pseudo-code described in figure 5.1, but the *compute_property* function is derived from DBSCAN and computes the local density of the points of clusters in the data belonging to the neighborhood of the current point, then the *myColor* procedure chooses the color and the speed of the agents with regard to this local density. The algorithm is based on the same parameters used in the DBSCAN algorithm: *MinPts*, the minimum number of points to form a cluster and *Eps*, the maximum distance that the agents can look at. In practice, the agent computes the local density (density) in a circular neighborhood (with a radius determined by its limited sight, i.e. Eps) and then it chooses the color in accordance to the following simple rules:

$$
\begin{array}{lll}
density > MinPts & \Rightarrow mycolor = red & (speed = 0) \\
\frac{MinPts}{4} < density \leq MinPts & \Rightarrow mycolor = green & (speed = 1) \\
0 < density \leq \frac{MinPts}{4} & \Rightarrow mycolor = yellow & (speed = 2) \\
density = 0 & \Rightarrow mycolor = white & (speed = 0)
\end{array}
$$

In the running phase, the yellow and green agents will compute their direction, according to the rules previously described, and will move following this direction and with the speed corresponding to their color.

In addition, new red agents will run the merge procedure, which will merge the neighboring clusters. The merging phase considers two different cases: when we have never visited points in the circular neighborhood and when we have points belonging to different clusters. In the first case, the points will be labeled and will constitute a new cluster; in the second case, all the points will be merged into the same cluster, i.e. they will get the label of the cluster discovered first.

SPARROW suffers the same limitation as DBSCAN, i.e. it can not cope with clusters of different densities. A new algorithm SPARROW-SNN, introduced in the next subsection, is more general and overcomes these drawbacks. It can be used to discover clusters with differing sizes, shapes and densities in noise data.

3.3.3 The SNN algorithm

SNN is a clustering algorithm developed by Ertöz, Steinbach and Kumar [31] to discover clusters with differing sizes, shapes and densities in noise, high dimensional data. The algorithm extends the nearest-neighbor non-hierarchical clustering technique by Jarvis-Patrick [54] redefining the similarity between pairs of points in terms of how many nearest neighbors the two points share. Using this new definition of similarity, the algorithm eliminates noise and outliers, identifies representative points, and then builds clusters around the representative points. These clusters do not contain all the points, but rather represent relatively uniform group of points. The SNN algorithm starts performing the Jarvis-Patrick scheme. In the Jarvis-Patrick algorithm a set of objects is partitioned into clusters on the basis of the number of shared nearest-neighbors. The standard implementation is constituted by two phases. The first is a pre-processing stage that identifies the K nearest-neighbors of each object in the dataset. In the subsequent clustering stage, two objects i and j join the same cluster if:

- i is one of the $Knearest - neighbors$ of j;
- j is one of the $Knearest - neighbors$ of i;
- i and j have at least K_{min} of their $Knearest - neighbours$ in common;

where K and K_{min} are used-defined parameters. For each pair of points i and j is defined a link with an associate weight. The strength of the link between i and j is defined as:

$$strength(i,j) = \sum (k+1-m)(k+1-n) \qquad where \qquad i_m = j_n$$

In the equation above, k is the nearest neighbor list size, m and n are the positions of a shared nearest neighbor in i and j's lists. At this point,

clusters can be obtained by removing all edges with weights less than a user specified threshold and taking all the connected components as clusters. A major drawback of the Jarvis-Patrick algorithm is that, the threshold needs to be set high enough since two distinct set of points can be merged into same cluster even if there is only link across them. On the other hand, if a high threshold is applied, then a natural cluster will be split into many small clusters due to the variations in the similarity in the cluster.

SNN addresses these problems introducing the following steps.

1. For every node (data point) calculates the total strength (connectivity) of links coming out of the point;
2. Identify representative points by choosing the points that have high density ($> core\_threshold$);
3. Identify noise points by choosing the points that have low density ($< noise\_threshold$) and remove them;
4. Remove all links between points that have weight smaller than a threshold ($merge\_threshold$);
5. Take connected components of points to form clusters, where every point in a cluster is either a representative point or is connected to a representative point.

The number of clusters is not given to the algorithm as a parameter. Also note that not all the points are clustered.

3.3.4 SPARROW-SNN algorithm

Sparrow-SNN follows the pseucode of figure 3.2 and starts a fixed number of agents that will occupy a randomly generated position. From their initial position, each agent moves around the spatial data testing the neighborhood of each location in order to verify if the point can be identified as a *representative* (or core) point.

The *compute_property* function represents the connectivity of the point as defined in SNN algorithm. In practice, when an agent falls on a data point A, not yet visited, it computes the connectivity, *conn(A)*, of the point, i.e. computes the total number of strong links the points has, according to the rules of the SNN algorithm. Note that SPARROW-SNN computes for each data element the nearest-neighbor list using a *similarity-threshold* that reduces the number of data elements to take in consideration. The introduction of the similarity-threshold produces variable-length nearest-neighbor lists and therefore i and j must have at least P_{min} of the shorter nearest-neighbor list in common; where P_{min} is a user-defined percentage.

Points having connectivity smaller than a fixed threshold ($noise\_threshold$) are classified as noise and are considered for removal from clustering. Then a color is assigned to each agent, on the basis of the value of the connectivity computed in the visited point, using the following procedure (called *color_agent*() in the pseudocode):

$$
\begin{array}{lll}
conn > core\_threshold & \Rightarrow mycolor = red & (speed = 0) \\
noise\_threshold < conn \leq core\_threshold & \Rightarrow mycolor = green & (speed = 1) \\
0 < conn < noise\_threshold & \Rightarrow mycolor = yellow & (speed = 2) \\
conn = 0 & \Rightarrow mycolor = white & (speed = 0)
\end{array}
$$

The colors assigned to the agents are: red, revealing representative points, green, border points, yellow, noise points, and white, indicating an obstacle (uninteresting region). After the coloration step, the green and yellow agents compute their movement observing the positions of all other agents that are at most at some fixed distance (*dist_max*) from their and applying the same rules as SPARROW. In any case, each new red agent (placed on a representative point) will run the merge procedure, as described in the Sparrow subsection, so that it will include, in the final cluster, the representative point discovered, and together to the points that share with it a significant (greater that Pmin) number of neighbors and that are not noise points. The merging phase considers two different cases: when we have visited none of these points in the neighborhood and when we have points belonging to different clusters. In the former, the same temporary label will be assigned and a a new cluster will be constituted; in the latter, all the points will be merged into the same cluster, i.e. they will get the label corresponding to the smallest label. So clusters will be built incrementally.

3.4 Experimental results

The following section shows the experimental results obtained for the two clustering algorithms. Both algorithms have been implemented using Swarm, a multi-agent software platform for the simulation of complex adaptive systems. In the Swarm system the basic unit of simulation is the swarm, a collection of agents executing a schedule of actions. **Swarm** provides object oriented libraries of reusable components for building models and analyzing, displaying, and controlling experiments on those models. More information about Swarm can be obtained from [86].

All the experiments were averaged over 30 tries. Where not differently specified, we run our algorithm using 50 agents and we set the visibility radius of the agents (dist_max) to 6 and the minimum distance among the agents (dist_min) to 2.

3.4.1 SPARROW results

We evaluated the accuracy of the solution supplied by SPARROW in comparison with the one of DBSCAN and the performance of the search strategy of SPARROW in comparison with the standard flocking search strategy and with the linear randomized search. Furthermore, we studied the impact of the

number of agents on foraging for clusters performance. Results are compared with the ones obtained using a publicly available version of DBSCAN. Our algorithm uses the same parameters as DBSCAN. Therefore, if we visited all the points of the dataset, we would obtain the same results. Then, in our experiments we consider as 100% the cluster points found by DBSCAN (note DBSCAN visit all the points). We want to verify how we come close to this percentage visiting only a portion of the entire dataset. For the experiments, we used two synthetic data sets and one real, shown in figure 3.4. The first data set, called GEORGE, consists of 5463 points. The second data set, called DS4, contains 8843 points. Each point of the two data sets has two attributes that define the x and y coordinates. Furthermore, both data sets have a considerable quantity of noise. The third called SEQUOIA is composed by 62556 names of landmarks (and their coordinates), and was extracted from the US Geological Survey's Geographic Name Information System. We set eps to 9 for all the datasets and MinPts to 20 for George and DS4 and 40 for Sequoia.

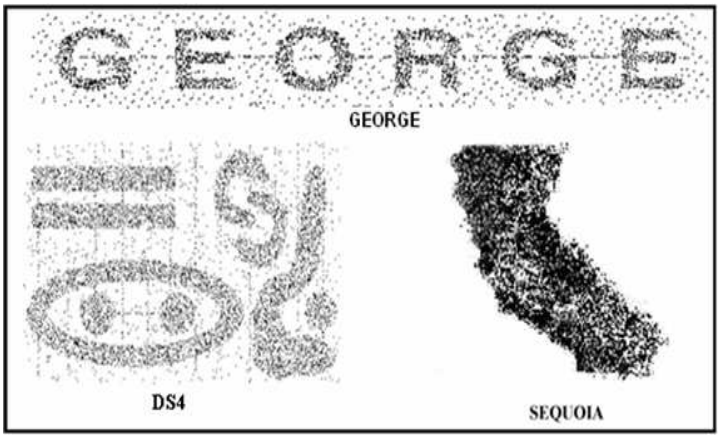


Fig. 3.4. The three data sets used in our experiments.

Although DBSCAN and SPARROW would produce the same results if we examined all points of the data set, our experiments showed that SPARROW can obtain, with an average accuracy about 93% on GEORGE dataset and about 78% on DS4, the same number of clusters with a slightly smaller percentage of points for each cluster using only 22% of the spatial queries used by DBSCAN. The same results cannot be obtained by DBSCAN because of the different strategy of attribution of the points to the clusters. In fact, if we stopped DBSCAN before which it had performed the spatial queries on all the points, we should obtain a correct number of points for the clusters already individuated and probably a smaller number of points for the cluster that it was building, but obviously we will not discover all the clusters.

	Perc. of data points for cluster			
	7%	12%	22%	60%
G	57.6 %	83.2 %	92.4 %	99.4 %
E	58.2 %	71.1 %	91.3 %	99.7 %
O	61.3 %	85.8 %	94.2 %	99.6 %
R	50.6 %	72.7 %	93.3 %	99.1%
G	48.8 %	77.2 %	89.7 %	98.8 %
E	61.1 %	81.2 %	94.5 %	99.6 %

Table 3.1. Number of clusters and number of points for clusters for GEORGE data set (percentage in comparison to the total point for cluster found by DBSCAN) when SPARROW analyzes 7%, 12%, 22% and 60% points.

	Perc. of data points for cluster			
	7%	12%	22%	70%
1	51.16%	70.99%	78.86%	95.76 %
2	45.91%	64.74%	74.40%	95.45 %
3	40.68%	59.36%	81.95%	97.55 %
4	44.21%	60.66%	81.67%	98.05 %
5	54.65%	58.72%	71.54%	94.99 %
6	48.77%	59.91%	78.10%	97.76 %
7	54.29%	66.43%	79.18%	96.12 %
8	51.16%	70.99%	78.86%	96.33 %
9	45.91%	64.74%	74.40%	95.25 %

Table 3.2. Number of clusters and number of points for clusters for DS4 data set (percentage in comparison to the total point for cluster found by DBSCAN) when SPARROW analyzes 7%. 12%. 22% and 70% points.

	Perc. of data points for cluster			
	7%	12%	22%	70%
S. Francisco	48.12%	66.22%	79.32%	98.88%
Sacramento	44.03%	61.11%	80.34%	97.56%
Los Angeles	51.21%	68.65%	81.92%	98.32%

Table 3.3. Number of clusters and number of points for clusters for Sequoia data set (percentage in comparison to the total point for cluster found by DBSCAN) when SPARROW analyzes 7%. 12%. 22% and 70% points.

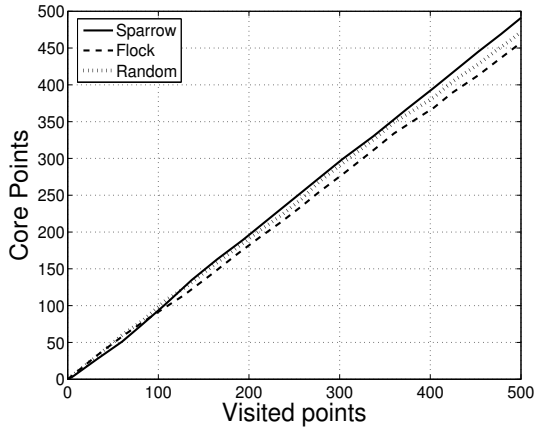


Fig. 3.5. Number of core points found for SPARROW, random and flock strategy vs. total number of visited points for the DS4 dataset.

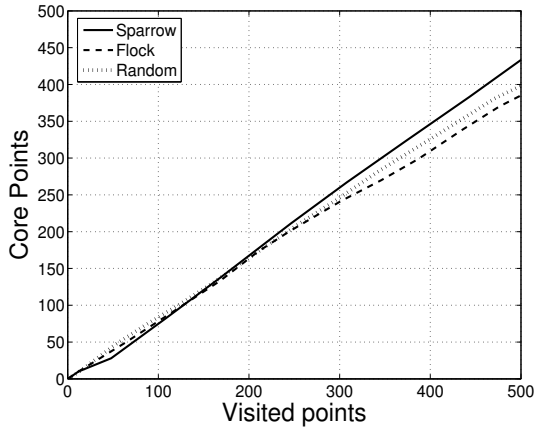


Fig. 3.6. Number of core points found for SPARROW, random and flock strategy vs. total number of visited points for the GEORGE dataset.

Table 1 and table 2 show, for the two data sets, the number of clusters and the percentage of points for each cluster found by DBSCAN and SPARROW.

To verify the effectiveness of the search strategy, we have also compared SPARROW with the random-walk search (RWS) strategy of the Reynolds' flock algorithm and with the linear randomized search (LRS) strategy.

Figure 3.5, 3.6 and 3.7 show the number of core points found for the three different strategies versus number of visited points respectively for the DS4, GEORGE and Sequoia data set. At the beginning, the random strategy, and also (to a minor extent) the flock, overcomes SPARROW, but, after 200-250 visited points SPARROW presents a superior behavior on both the search

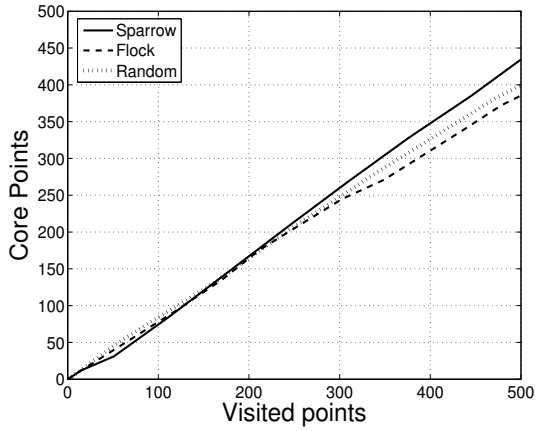


Fig. 3.7. Number of core points found for SPARROW, random and flock strategy vs. total number of visited points for the Sequoia dataset.

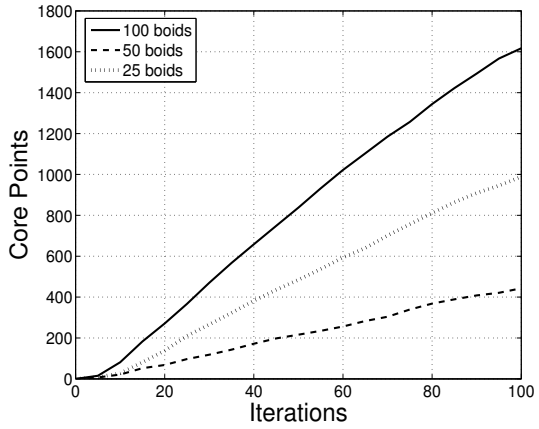


Fig. 3.8. The impact of the number of agents on the foraging for clusters strategy (DS4).

strategies because of the adaptive behavior of the algorithm that allows agents to learn on their previous experience.

Finally, we present the impact of the number of agents on the foraging for clusters performance. Figure 3.8, 3.9 and 3.10 give the number of core points found in 100 time steps (iterations) for 25, 50 and 100 agents. A comparative analysis reveals that a 100-agents population discovers a larger number of points than the other two populations with a smaller number of agents (the scalability is almost linear). This scalable behavior of the algorithm determines a faster completion time because a smaller number of iterations are necessary to produce the solution.

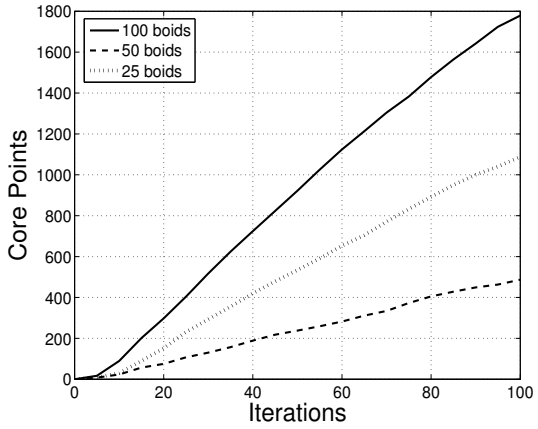


Fig. 3.9. The impact of the number of agents on the foraging for clusters strategy (GEORGE).

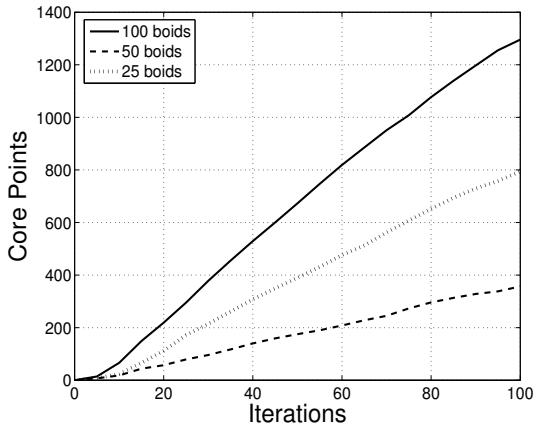


Fig. 3.10. The impact of the number of agents on the foraging for clusters strategy (Sequoia).

3.4.2 SPARROW-SNN results

This section presents an experimental evaluation of SPARROW-SNN algorithm. We intent to explore the following issues:

1. determine the accuracy of the approximate solution that we would obtain if we run our cluster algorithm on only a small per cent of points, as opposed to running the SNN clustering algorithm on the entire dataset;
2. determine the effects of using SPARROW-SNN searching strategy as opposed to the random-walk search strategy of the Reynolds' flock algorithm

and with the linear randomized search strategy, in order to identify clusters;
3. determine the impact of the number agents on the foraging for clusters performance.

For the experiments we used a synthetic dataset and two real life datasets. The structures of these data sets are shown in figure 3.11 a, b and c.

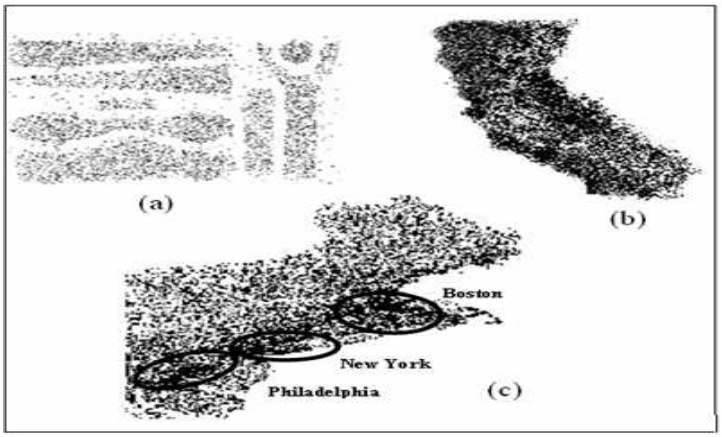


Fig. 3.11. The three datasets used in our experiments a) DS1, b) Sequoia, c) North-East (circles surround the three towns of the North-East dataset).

	Perc. of data points for cluster			
	7%	12%	22%	70%
1	41.35 %	58.31 %	70.37 %	98.25 %
2	30.08 %	53.58%	60.72 %	97.02 %
3	29.28 %	40.99 %	53.02 %	91.10 %
4	20.9 %	30.5 %	51.41 %	89.65 %
5	53.38 %	65.36 %	76.56%	99.46 %
6	56.89 %	69.87 %	73.63 %	98.90 %
7	33.89 %	43.5 %	61.58 %	99.59 %

Table 3.4. Number of clusters and number of points for clusters for the DS1 data set (percentage in comparison to the total point for cluster found by SNN) when SPARROW-SNN analyzes 7%, 12%, 22% and 70% points.

The first dataset, called DS1, contains 8000 points and presents different densities in the clusters. The second, Sequoia, is the same described in subsection 5.1. The third dataset, called North-East, contains 123593 points

	Perc. of data points for cluster			
	7%	12%	22%	70%
Philadelphia	42.5 %	65.2 %	79.4 %	99.65 %
New York	38.7 %	52.3 %	67.6 %	96.61 %
Boston	46.5 %	68.6 %	82.3 %	99.94 %

Table 3.5. Number of clusters and number of points for clusters for the DS4 data set (percentage in comparison to the total point for cluster found by SNN) when SPARROW-SNN analyzes 7%, 12%, 22% and 70% points.

	Perc. of data points for cluster			
	7%	12%	22%	70%
S. Francisco	49,11%	67,75%	80,38%	99,55%
Sacramento	44,81%	61,72%	80,86%	98,22%
Los Angeles	51,03%	71,99%	84,68%	98,86%

Table 3.6. Number of clusters and number of points for clusters for the Sequoia data set (percentage in comparison to the total point for cluster found by SNN) when SPARROW-SNN analyzes 7%, 12%, 22% and 70% points.

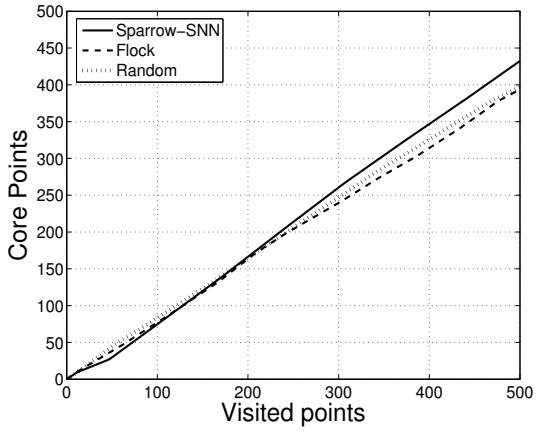


Fig. 3.12. Number of core points found for SPARROW-SNN, random and flock strategy vs. total number of visited points for the DS1 dataset.

representing postal addresses of three metropolitan areas (New York, Boston and Philadelphia) in the North East States. We set noise_threshold and core_threshold respectively to 5 and 10 for the DS4 dataset and to 10 and 20 for Sequoia and North East. K and P_{min} were set respectively to 30 and 0.8 for all the datasets.

We first illustrate the accuracy of our SPARROW-SNN algorithm in comparison with SNN algorithm when SPARROW-SNN is used as a technique for approximate clustering. To this purpose, we implemented SNN and we

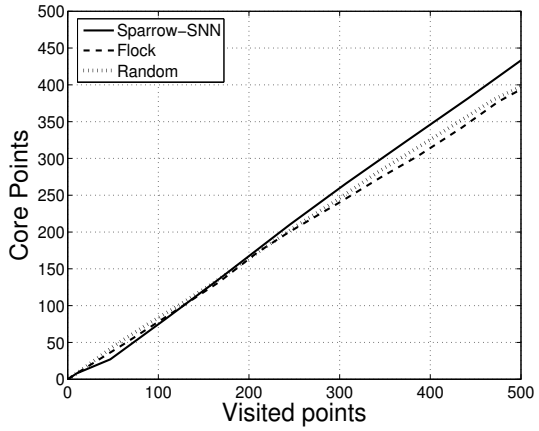


Fig. 3.13. Number of core points found for SPARROW-SNN, random and flock strategy vs. total number of visited points for the North-East dataset.

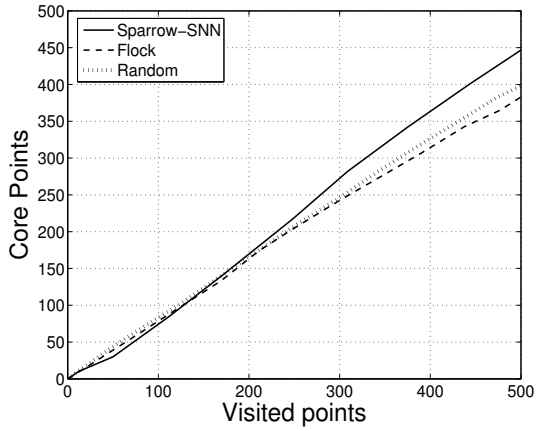


Fig. 3.14. Number of core points found for SPARROW-SNN, random and flock strategy vs. total number of visited points for the Sequoia dataset.

computed the number of clusters and the number of points for cluster for the above described datasets. Tables 3.4, 3.5 and 3.6 present a comparison of these results with respect to ones obtained from SPARROW-SNN when a population of 50 agents has visited respectively 7%, 12% and 22% and 70% of the entire dataset.

Note that with only 7% of points we can have a cler vision of the found clusters and with a few more points we can obtain the nearly totality of the points. This trend is well marked in the North-East dataset. For the DS1 dataset, the results are not so well defined because the clusters called number 3 and 4 have very few points, so they are very hard to discover. For the real

datasets, we only reported the results for the three main clusters representing respectively the towns of Boston, New York and Philadelphia (Sequoia) and the regions of S. Francisco, Sacramento and Los Angeles (North-East).

We can explain the good results thought the adaptive search strategy of SPARROW-SNN that requires to the individual agents to first explore the data searching for representative points whose position is not known a priori, and then, after the representative points are located, all the flock member are steered to move towards the representative points, that represent the interesting regions, in order to help them, avoiding the uninteresting areas that are instead marked as obstacles and adaptively changing their speed.

To verify the effectiveness of the search strategy we have compared SPARROW-SNN with the random-walk search strategy and with the standard flocking search strategy. Figures 13, 14 and 15 show the number of representative points found with SPARROW-SNN and those found with the random search and with the standard flock vs. the total number of visited points. All the figures reveal that the number of representative points discovered at the beginning (and until about 170 visited points for DS1, 150 for North-East and 190 for Sequoia dataset) from the random strategy, and also (to a minor extent) for the flock, is greater than those discovered by SPARROW-SNN.

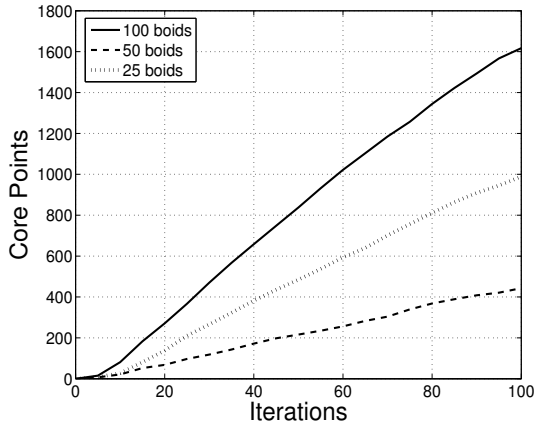


Fig. 3.15. The impact of the number of agents on the foraging for clusters strategy (DS1).

Finally, in order to study the scalability of our approach, we present the impact of the number of agents on the foraging for clusters performance. Figures 3.15, 3.16 and 3.17 give, respectively for the DS1, North-East and Sequoia dataset, the number of clusters found in 100 time steps (iterations) for 25, 50 and 100 agents. A comparative analysis reveals that a 100-agents population discovers a larger number of clusters (almost linear) than the other two populations with a smaller number of agents. This scalable behavior of

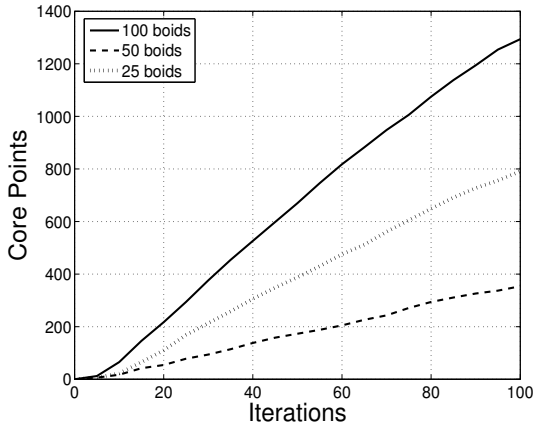


Fig. 3.16. The impact of the number of agents on the foraging for clusters strategy (North-East).

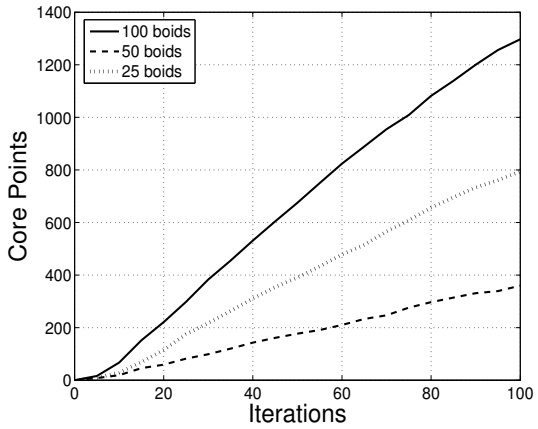


Fig. 3.17. The impact of the number of agents on the foraging for clusters strategy (Sequoia).

the algorithm determine a faster completion time because a smaller number of iterations is necessary to produce the solution. We try to partially give a theoretically explanation of the behavior observed in figure 3.12, 3.13 and 3.14, i.e. the improvement in accuracy of the SPARROW-SNN strategy, introducing an entropy based model in the next section.

3.5 Entropy model

In order to verify and explain the behavior of our system, we used a model based on the entropy introduced in [95] by Parunak and Brueckner. The authors adopted a measure of entropy to analyze emergence in multi-agent systems. Their fundamental claim is that the relation between self-organization based on emergence in multi-agent systems and concepts as entropy is not just a loose metaphor, but it can provide quantitative and analytical guidelines for designing and operating agent systems. These concepts can be applied in measuring the behavior of multi-agent systems. The main result, that the above cited paper suggests, concerns the principle that the key to reduce disorder in a multi-agent system and to achieve a coherent global behavior is coupling that system to another in which disorder increases. This corresponds to a macro-level where the order increases, i.e. a coherent behavior arises, and a micro-level where an increase in disorder is the cause for this coherent behavior at the macro-level.

A multi-agent system follows the second law of thermodynamics "Energy spontaneously disperses from being localized to becoming spread out if it is not hindered", if agents move without any constriction. However, if we add information in an intelligent way, the agents' natural tendency to maximum entropy will be contrasted and the system will go towards self-organization.

3.5.1 Spatial Entropy

In the following, we analyze the behavior of our flocking algorithm; all the considerations can be applied to SPARROW or SPARROW-SNN. Really, as stated in [95], we can observe two levels of entropy: a macro level in which organization takes place, balanced by the micro in that we have an increase of entropy. For the sake of clarity, in the flocking algorithm, micro-level is represented by red and white agents' positions, signaling respectively interesting and desert zones, and the macro level is computed considering all the agents positions. So, we expect to observe an increase of micro entropy by the birth of new red and white agents and, on the contrary, a decrease in macro entropy indicating organization in the coordination model of the agents. In fact, at the beginning, the agents move and spread out randomly. Afterward, the red agents act as catalyzers towards the most interesting zones, organization increases and entropy should decrease. Note that in the case of ants, attraction is produced by the effect of pheromone, while for our flock, it is caused by the attractive power of the red birds.

We introduce a more formal description of the entropy-based model. In information theory, entropy can be defined as:

$$S = -\sum_i p_i log p_i \tag{3.9}$$

Now, to adapt this formula to our aims, a location-based entropy is introduced. Consider an agent moving in a space of data divided in a grid $N \times M = K$, where all the cells have the same dimensions. So, if N and M are quite large, each random agent will have the same probability to be in one of the K cells of the grid. The entropy can be measured experimentally running the flocking algorithm for T tries and counting how many times an agent falls in the same cell i for each time-step. Dividing this number by T we obtain the probability p_i that the agent be in this cell.

Then, the locational entropy will be:

$$S = -\frac{\sum_{i=1}^{k} p_i log p_i}{log k} \tag{3.10}$$

In the case of a random distribution of the agents, every state has probability $\frac{1}{k}$, so the overall entropy will be $\frac{logk}{logk} = 1$; this explains the factor of normalization log k in the formula.

This equation can be generalized for P agents, summing over all the agents and averaging dividing by P. Equation 3.10 represents the macro-entropy; if we consider only red and white points, it represents the micro one.

We run our algorithm (averaged over 100 tries) for 2000 time-steps using 50 agents and computed the micro and macro locational entropy for the North-East dataset. The results are showed in figure 3.18 and 3.19. As expected, we can observe an increase in micro entropy and a decrease in macro entropy due to the organization introduced in the coordination model of the agents by the attraction towards red agents and the repulsion of white agents. On the contrary, in random and standard flock model, the curve of macro entropy is almost constant, confirming the absence of organization.

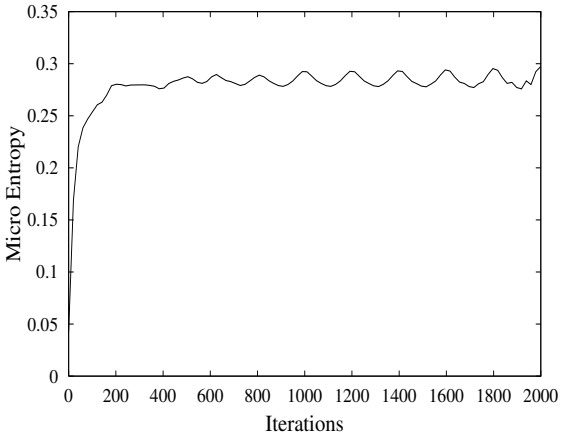


Fig. 3.18. Micro Entropy (red and white agents) for the North-East dataset using Sparrow-SNN.

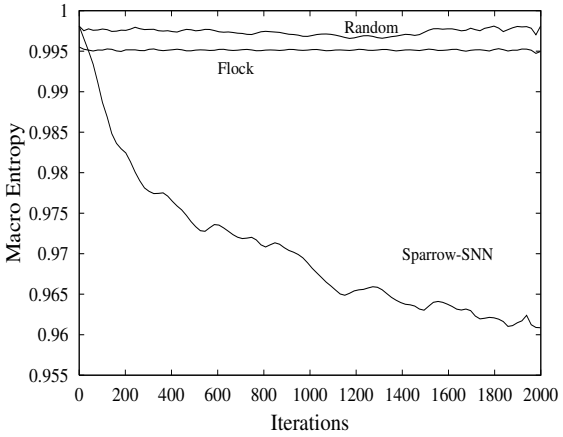


Fig. 3.19. Macro Entropy (all the agents) for the North-East dataset using SPARROW-SNN, random and standard flock.

The same behavior was observed for the algorithm Sparrow and for the other datasets.

3.5.2 Autocatalytic property

[96] defines autocatalytic property for agent systems as follows: "A set of agents has autocatalytic potential if in some regions of their joint state space, their interaction causes system entropy to decrease (and thus leads to increased organization). In that region of state space, they are autocatalytic". That behavior is not in contrast with the second law of thermodynamics, as the macro entropy is balanced by the reduction in the micro one and in the catalytic zones. In our modified flock, we have some zones of attraction due to the presence of red birds and some zones of repulsion due to the white birds.

We conducted simulations in order to verify the property of autocatalysm of our system and to better understand the behavior of our algorithm. We used our algorithm with the same parameters described in the previous subsection considering the CURE dataset, showed in figure 3.20. We have used this dataset, as it has a cluster distribution quite regular and that permits ourselves to better study the catalytical properties; in fact, the dataset contains 100000 points distributed in three circles and two ellipsoids and connected by a chain of outliers and random noise scattered in the entire space.

In figures 3.21 and 3.22 it is shown respectively the entropy in the cluster zones and outsides the clusters.

Entropy decreases both in cluster zones and outsides the clusters zones, as the flock visits more often cluster zones and keeps away from the other zones (this behavior also causes a decrease in the entropy). However these curves are not sufficient to verify the goodness of the algorithm as organization alone is

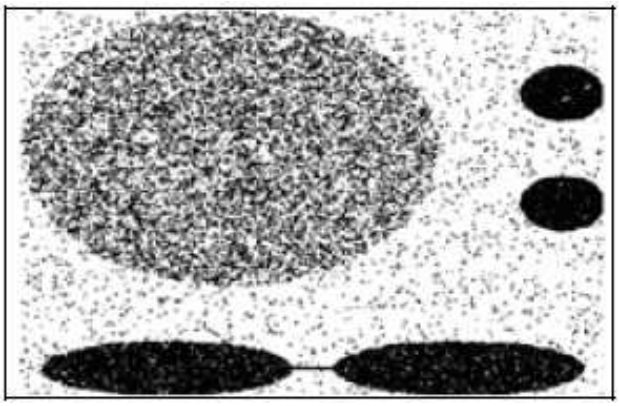


Fig. 3.20. CURE dataset.

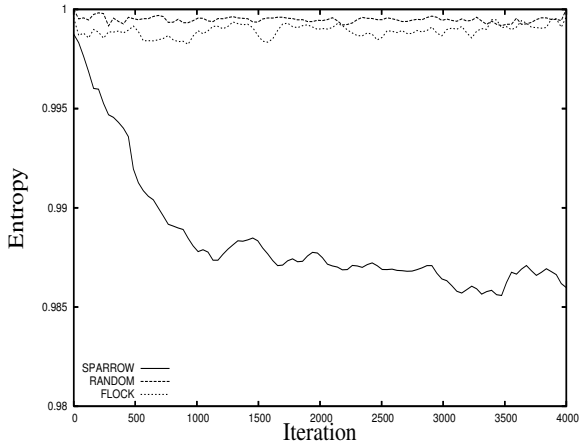


Fig. 3.21. Macro Entropy in the cluster zones for the Cure dataset using SPARROW-SNN, random and standard flock.

not sufficient to solve problems, but it must bring the search in the appropriate zones. In fact, the main idea behind our algorithm is to let the flocks dynamics explore the space and, when the birds reach a desirable region (zone dense of clusters), an autocatalytic force should be applied to the system (red birds) to keep the search in these zones.

Thus, we analyzed the average percentage of birds present in the two different zones (figure 3.23). In cluster zones we have about the 80% of the entire flock (while the space occupied from the clusters is about 65%) and this confirms that, in the interesting zones of the clusters, not only there is organization but there also a larger presence of searching agents.

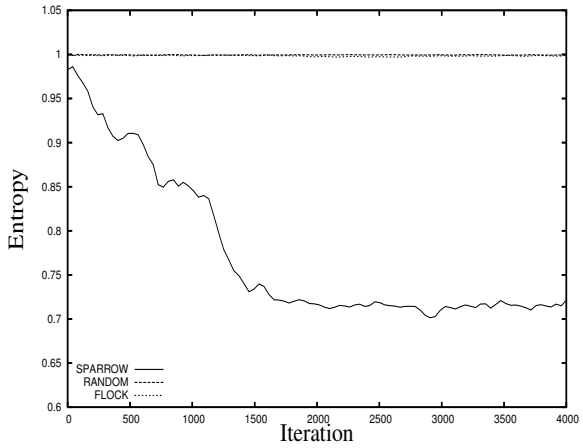


Fig. 3.22. Macro Entropy outside the cluster zones for the Cure dataset using SPARROW-SNN, random and standard flock.

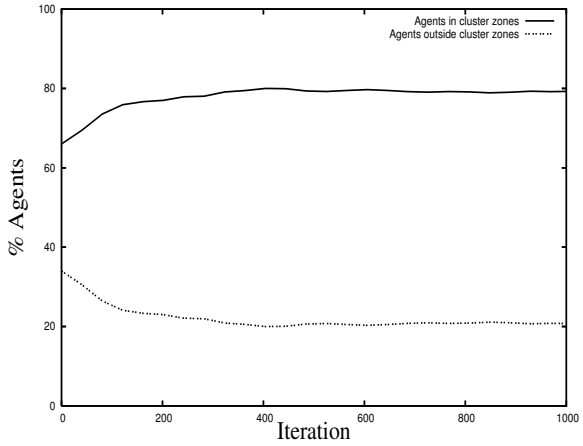


Fig. 3.23. Percentage of agents exploring cluster and non cluster zones for the Cure dataset using SPARROW-SNN.

3.6 Related works

Clustering problem has been faced in the recent past using algorithms based on the Swarm Intelligence paradigm. Termites, swarms of bees and especially ant colonies have polarized the attention of researchers but, to the best of our knowledge, the interesting property of flock of birds have not sufficiently exploited in the clustering task.

First pioneering work on clustering using artificial ants was conducted by Deneuborg in [24]. Ants-like agents clusterize together objects, moving through a two-dimensional grid and picking and dropping them with a prob-

ability based on an estimation of the density of objects of the same type in the neighborhood.

Lumer and Faieta in [79] extended this work including a distance function between data objects for the purpose of exploratory data analysis and applied these techniques to a database of customer profiles, projecting the space of attributes into a two-dimensional space. Moreover, they introduced ants with different moving speeds (fast ants group clusters on large scales, while slow ants work at smaller scales by placing data with more accuracy), a short term memory (ants can remembers the last m dropped data, avoiding to create equivalent clusters) and behavioral switches (ants destroy clusters if they have not performed any deposit or pick up actions for a given number of time steps in order to escape local non-optimal spatial configurations).

The ACluster algorithm overcomes the drawback associated with these three last characteristics and simplifies the model of Lumer and Faieta, introducing the concept of bio-inspired spatial transition probabilities that increase pheromone in locations where objects are more diffuse and adding two different threshold responses, one associated to the density of data and the other to their similarity. The algorithm is used for the task of image retrieval [101], for the clustering of textual documents [100] and extended in [99] for applying it to continuous data stream.

In [46], Handl and Meyer added adaptive scaling (when few pickup and drop action occurred over a specified time period the ants' sensitivity was reduced, in the opposite case was increased), ant jumps (more efficient exploration of the space), stagnation control (after a number of unsuccessful dropping attempts an ant drops its load regardless of the neighborhood's similarity) and eager ants (ant was put immediately on a data point) to the classical model. The algorithms yield significant improvements in terms of quality and speed and was employed in a document retrieval system.

Monmarché et al. in [89] proposed an algorithm that combines in an intelligent way clustering ants and k-means. Moreover many data can be put on the same cell and each cell with a non-zero number of items corresponds to a cluster. At the same way, each ant can carry more data until a maximum capacity. Initially, ants cluster data objects, then k-means was applied to refine the clusters, and this procedure is repeated for heap of objects. The evaluation on some synthetic and simple real world datasets gave good results in comparison with classical clustering algorithms.

Kanade and Hall in [59] developed an ant algorithm that, in a first stage, moves the cluster centers (and not the data points) in the feature space. As a second stage, the found centers are evaluated using the Fuzzy C Means or Hard C Means algorithm. These two stages are repeated until a fixed number of epochs. Results on some datasets are superior to the behavior of FCM and HCM initialized randomly.

In [122] an adaptive ant clustering algorithm (A2CA) was introduced. Among the novelty in comparison with the other algorithms, it is interesting the concept of progressive vision that permits to cope with cluster with differ-

ent dimension, making ants augment their vision when encounter a big cluster. The algorithm was also evaluated on a real world bioinformatics dataset in which the classical ant clustering algorithms fails, while A2CA worked quite well.

First idea of employing a flock of birds to perform the task of clustering can be found in [94]. The authors used a flock of colored birds to drive a geographical analysis machine (GAM); in practice, a flock of circles of different sizes explore the search space in order to find interesting geographical patterns. The system was validated on the city of Baltimore for discovering crime patterns. Note that the problem coped in this paper is not really clustering, as different kind of clusters are not discovered neither merged together, but only the problem of discovering interesting patterns was faced.

Kazemian et al. [63] introduced a data clustering strategy based on Flower Pollination by Artificial Bees (FPAB) in which the flowers represents the data and the gardens the clusters and pollen is used to move the flowers with lowest growth (probably not belonging to that cluster) from a garden to another. Then, the Fuzzy C Means algorithm is applied to refine the clusters and the new centers found are used as new flowers and the algorithm is repeated until an appropriate number of clusters is reached. FPAB is applied to some simple datasets and was compared with an ant algorithm.

Particle Swarm Optimization was also appled to the clustering task, but the point of view is quite different from the swarm based one. In fact, PSO was applied to discover clusters using an approach in that each particle represents a cluster centroid, as in [120], for example. This limits the applicability of the algorithm and presents all the problem of the centroid based approach.

4

P2P clustering algorithm

4.1 Introduction

In this chapter, I present P-SPARROW a novel algorithm which uses the concepts of a flock of birds that move together in a complex manner using simple local rules, to cluster distributed spatial data in P2P systems. P-SPARROW discovers the objects to be clustered at local sites. Each local site situates its own objects in a local 2D cellular space. P-SPARROW clusterizes data independently on the different local sites by a smart exploratory strategy based on a colored flock of birds combined with a density-based clustering method. On each cellular space, a set of agents, having different features (color and speed), discover local models of the clusters, represented by *core* objects, i.e. data points in which the cardinality of the neighborhood exceeds a fixed threshold. At intervals each node transfers the core points to the neighboring nodes. On each node, all the objects that are in the neighborhood of these core points are considered belonging to the same cluster. Furthermore, as the clusters are discovered, they are merged using an iterative distributed labeling strategy to generate global labels with which identify the global clusters. P-SPARROW has a number of nice properties. It has the advantages of being easily implementable on distributed systems as P2P networks and it is robust compared to the failure of individual agents and also of entire nodes. It can also be applied to perform efficiently *approximate clustering* since the points that are, to each iteration, visited and analyzed by the flock of agents represent a significant (in *ergodic* sense) subset of the entire dataset. The subset reduces the execution time since reduces the space of solutions that a clustering algorithm has to search keeping the accuracy loss as small as possible. P-SPARROW has no centralized coordinator. Each node acts independently from each other and intermediate results may be overturned as new data arrives. P-SPARROW behaves as an *anytime* algorithm in which the quality of results improves as computational time increases. Each node maintains an assumption of the correct result and updates it whenever new core points are discovered. During the execution of the algorithm, if the system remains static, then the solu-

tion will quickly converge toward the correct solution. However, in a dynamic system, in which nodes dynamically join or depart and the data changes over time, the changes are quickly and locally adjusted to, and the solution continues to converge. This property is particularly interesting if continuous data are analyzed. Furthermore, each node communicates only with its immediate neighbors. Locality implies that the algorithm is scalable to very large networks. Another outcome of the algorithm's locality is that the communication load it produces is small and decreases with the time. We have implemented P-SPARROW in a P2P network using the Jxta Protocol [1] to investigate the interaction of the parameters that characterize the algorithm.

4.2 P-Sparrow

P-SPARROW, a multi-agent distributed clustering algorithm implemented in a P2P network, combines the stochastic search of an adaptive flocking with a density-based clustering method and an iterative self-labeling strategy to generate global labels with which identify the clusters of all peers. Since P-SPARROW utilizes the principles of the conventional density-based algorithms, some of them are described first, then the algorithm is explained and the overall distributed architecture is illustrated.

4.2.1 Density-based clustering

Density-based clustering methods try to find clusters on the basis of the density of points in regions. Dense regions that are reachable from each other are merged to formed clusters. DBSCAN [32] is one the most popular density based methods and it is based on the idea that all the points of a data set can be regrouped into two classes: *clusters* and *noise*. Clusters are defined as a set of dense connected regions with a given radius (*Eps*) and containing at least a minimum number (*MinPts*) of points. The two parameters, Eps and MinPts, must be specified by the user and allow to control the density of the cluster that must be retrieved. The algorithm defines two different kinds of points in a cluster: *core points* and *non-core points*. A core point is a point with at least MinPts number of points in an Eps-neighborhood of the point. The non-core points in turn are either *border points* if they are not core points but they are *density-reachable* from another core point or *noise points* if they are not core points and are not density-reachable from other points. To find the clusters in a data set, DBSCAN starts from an arbitrary point and retrieves all points that are density-reachable from that point. A point p is density reachable from a point q, if the two points are connected by a chain of points such that each point has a minimal number of data points, including the next point in the chain, within a fixed radius. If the point is a core point, then the procedure yields a cluster. If the point is on the border, then DBSCAN goes on to the next point in the database and the point is assigned to the noise.

DBSCAN builds clusters in sequence (that is, one at a time), in the order in which they are encountered during space traversal. The retrieval of the density of a cluster is performed by successive spatial queries. Such queries are supported efficiently by spatial access methods such as R*-trees. DBSCAN is not suitable for finding *approximate* clusters in very large datasets. DBSCAN starts to create and expand a cluster from a randomly picked point. It works very thoroughly and completely accurately on this cluster until all points in the cluster have been found. Then another point outside the cluster is randomly selected and the procedure is repeated. This method is not suited to stopping early with an approximate identification of clusters.

Recently, DBDC a distributed version of DBSCAN algorithm has been presented in [47]. DBDC uses DBSCAN to make local clustering and determinate a local model after the local clustering is finished. All information which is comprised within the local model, i.e. the representatives and their corresponding e-ranges, is sent to a global server site, where a global clustering representation is produced from local representations. Based on this small number of representatives, the global clustering can be done very efficiently. After having created a global clustering, the complete global model is sent back to all client sites. The client sites relabel all objects located on their site independently from each other. DBDC does not scale-up well since, in a P2P scenario, no centralized coordinator and limited communications are required.

4.2.2 Distributed clustering

As in DBSCAN, P-SPARROW finds cluster performing region-queries on core points but it replaces the exhaustive search of the core points with a stochastic multi-agent search that discovers in parallel the points. P-SPARROW is constituted of two phases: a local phase for the **discovery** of the core points on each peer and a **merge** phase that concerns a global relaxation process in which nodes exchange cluster labels with nearest neighbors until a fixed point (i.e. all nodes detect no change in the labels) is reached.

All the data are partitioned among the peers, proportionally to the computing power and to the cpu-load of the peer itself. Each peer implements the flocking algorithm, described in figure 4.1, using a fixed number of agents that initially occupy a randomly generated position in the space. Each agent moves testing the neighborhood of each object (data point) it visits in order to verify if the point can be identified as a *core point*. Then, P-SPARROW uses a flocking algorithm with an exploring behavior in which individual members (agents) search some goals, whose positions are not known *a priori*, in parallel and signal the presence or the lack of significant patterns into the data to other flock members, by changing color. The entire flock then moves towards the agents (*attractors*) that have discovered interesting regions to help them, avoiding the uninteresting areas that are instead marked as obstacles. The color is assigned to the agents by a function associated to the data analyzed during the exploration, according to the DBSCAN density-based rules and

```
for i=1 ... MaxIterations
        foreach agent (yellow, green)
                if (not visited (current_point))
                        density = compute_local_density();
                        mycolor= color_agent(density);
                endif
        end foreach
        foreach agent (yellow, green)
                dir= compute_dir();
        end foreach
        foreach agent (all)
                switch (mycolor){
                        case yellow, green: move(dir, speed(mycolor)); break;
                        case white: stop ();generate_new_agent();break;
                        case red: stop (); merge(); if (new_red()) clone_agent(); break; }
        end foreach
        if ((bag_out.dimension()> threshold)or(i%IterMigr==0)) send_bag();
        if (bag_in_full()) notify_changes();
end for
```

Fig. 4.1. The pseudo-code of P-SPARROW executed on every peer.

with the same parameters: *MinPts*, the minimum number of points to form a cluster and *Eps*, the radius of the circle containing these points. In practice, the agent computes the local density (*density*) in a circular neighborhood (with a radius determined by its limited sight, i.e. *Eps*) and then it chooses the color (and the speed) in accordance to some simple rules. So *red*, reveals a high density of interesting patterns in the data ($density > MinPts$ and the agent takes speed=0), *green*, a medium one ($\frac{MinPts}{4} < density \leq MinPts$ and speed=1), *yellow*, a low one($0 < density \leq \frac{MinPts}{4}$ and speed=2), and white, indicates a total absence of patterns ($density = 0$ and speed=0). The color is used as a communication mechanism among flock members to indicate them the roadmap to follow. The main idea behind our approach is to take advantage of the colored agent in order to explore more accurately the most interesting regions (signaled by the red agents) and avoid the ones without clusters (signaled by the white agents). Red and white agents stop moving in order to signal these regions to the others, while green and yellow ones fly to find clusters. Green agents will move more slowly than yellow agents in order to explore more carefully zones with an higher density of points. The variable speed introduces an adaptive behavior in the algorithm. In fact, agents adapt their movement and change their behavior (speed) on the basis of their previous experience represented from the red and white agents. Anyway, each flying agent computes its heading by taking the weighted average of alignment, separation and cohesion (as illustrated in previous chapter). Green and yellow agents compute their movement observing the positions of all the agents that

are at most at some fixed distance (*dist_max*) from them and applying the rules of Reynolds' [103] with the following modifications: *alignment* and *cohesion* do not consider yellow agents, since they move in a not very attractive zone; *cohesion* is the resultant of the heading towards the average position of the green flockmates (centroid), of the attraction towards red agents, and of the repulsion by white agents; a *separation* distance is maintained from all the agents, whatever their color is.

New red agents executes the merge procedure; in practise a temporary label will be given to these agents and to all the points of their neighborhood, if they are not already labeled. Otherwise the minimum of all the labels will be assigned to all the core points in this neighborhood, in order to make them belong to the same cluster. In this way, on each peer the set of red agents (core points) determinates the local model of clustering. Neighboring peers must be informed about the new core points or about the new labels in order to merge all the points belonging to the same cluster. To this end, red agents create clone agents and put them in an apposite bag and, when a fixed number of clone agents is achieved (i.e. a bag of agents has reached the desired dimension) or a certain number of iterations have been performed, each peer will send the bag containing the cloned red agents to the neighboring peers. Consequently, the agents received from the other peers will be put in another bag that will be used in the next iteration (or when it become full) for the merge phase. In practise, the new agents continuously update the labels as multiple clusters take shape concurrently. This continues until nothing changes, by which time all the clusters will have been labeled with the minimum initial label of all the sites containing the data. All the points having the same label form a cluster.

4.2.3 The software architecture

The software architecture of P-Sparrow is described in figure 4.2.

The **flock platform** manages the cellular space in which the agents move. Furthermore, it supplies the main procedures concerning the agents (move, remote move, create new agent, clone agents, etc..) using the underlying levels. Agents of different colors will be scheduled by means of the **agents scheduler**.

The **resource manager** (RM) execute efficiently range queries (i.e. compute density) in the dataset, accessing the repository, in order to choose the new color of the agents. The RM is also responsible of putting new agents received by the neighboring peers in the appropriate zone in order to start a new phase of merge. The arrival of a new bag of agents is signaled by the **notifier manager** that supplies also information about new events such as the fall of a peer, the convergence of the algorithm, etc... The **network manager** handles the send and the receive of the bags of agents on the basis of the topology of the system (see subsection 4.3.2 for more details about the topology used), using JXTA sockets.

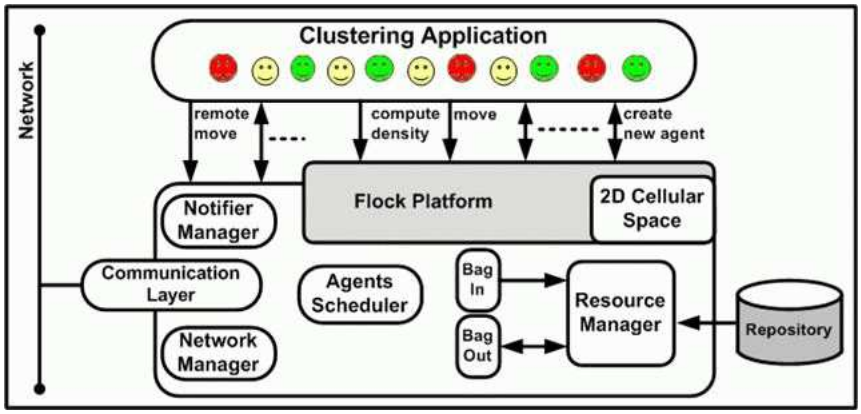


Fig. 4.2. The software architecture of P-Sparrow.

4.3 Experimental Results

In this section, we want to analyze the goodness of our algorithm in the task of performing approximate clustering and we want to verify some interesting properties of our distributed system (i.e. accuracy, scalability, etc..). In our experiments, we used two spatial datasets: one synthetic (called DS1-CURE) and one real (SEQUOIA). DS1-CURE (used in [43]) contains 100000 points in three circles and two ellipsoids connected by a chain of outliers and random noise scattered in the entire space; SEQUOIA was composed by 62556 names of landmarks (and their coordinates), and was extracted from the US Geological Survey's Geographic Name Information System. The three main clusters in this dataset represent respectively the areas of S. Francisco, Sacramento and Los Angeles.

4.3.1 Accuracy and Scalability

We run our algorithm using 100 agents working until they explore the 1%, 2%, 5% and 10% of the entire data set, using 16, 32 and 80 peers. All the experiments were averaged over 30 tries. Our algorithm uses the same parameters as DBSCAN. Therefore, if we visited all the points of the dataset, we would obtain the same results as DBSCAN. Then, in our experiments we consider as 100% the cluster points found by DBSCAN (note DBSCAN visit all the points). We want to verify how we come close to this percentage visiting only a portion of the entire dataset and that must be effective for different number of peers involved in the computation. Note that the dominant operation in the computation is the execution of the range queries, effectuated each time a point is visited, while the time of the other operations is negligible. So, the fact of reducing the % of visited points considerably reduces the execution time.

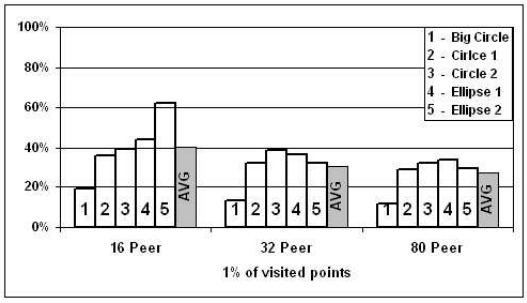


Fig. 4.3. Number of points for cluster for Cure dataset (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 1% of total points, using 16, 32 and 80 peers.

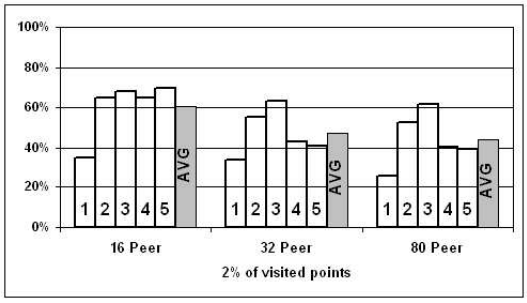


Fig. 4.4. Number of points for cluster for Cure dataset (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 2% of total points, using 16, 32 and 80 peers.

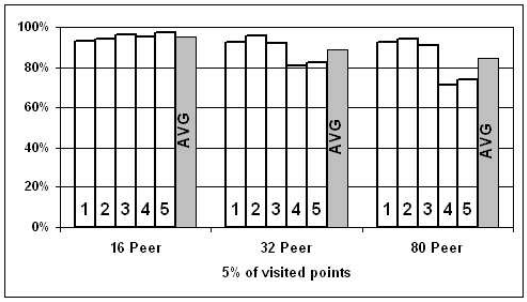


Fig. 4.5. Number of points for cluster for Cure dataset (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 5% of total points, using 16, 32 and 80 peers.

For a large number of peers, the density of points for cluster for peer necessarily decreases; so we have to choose a different value of the parameter

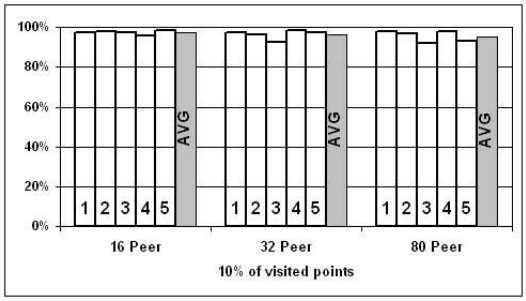


Fig. 4.6. Number of points for cluster for Cure (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 10% of total points, using 16, 32 and 80 peers.

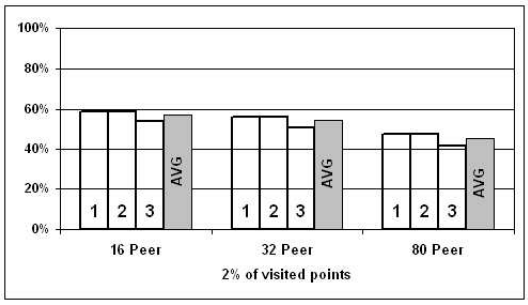


Fig. 4.7. Number of points for cluster for Sequoia dataset (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 1% of total points, using 16, 32 and 80 peers.

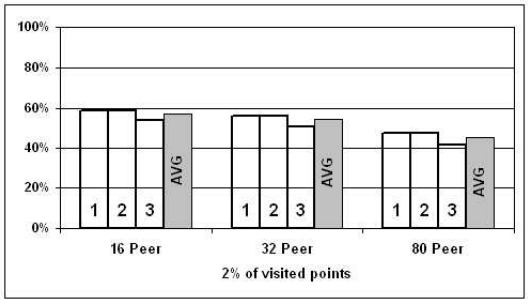


Fig. 4.8. Number of points for cluster for Sequoia dataset (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 2% of total points, using 16, 32 and 80 peers.

MinPts to keep into account this aspect. In practice, we choose a value of MinPts inversely proportional to the number of peers (i.e. if we fix MinPts

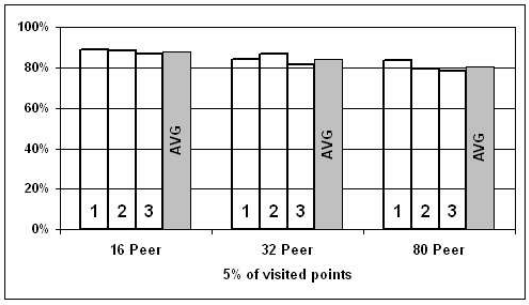


Fig. 4.9. Number of points for cluster for Sequoia dataset (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 5% of total points, using 16, 32 and 80 peers.

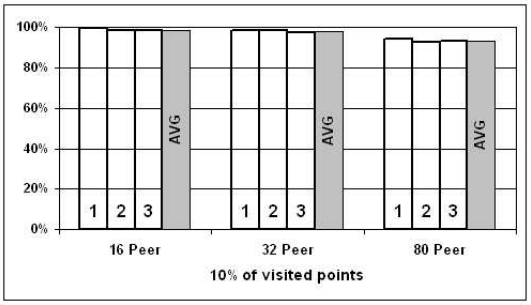


Fig. 4.10. Number of points for cluster for Sequoia (percentage in comparison to the total number of points for cluster) when P-SPARROW analyzes 10% of total points, using 16, 32 and 80 peers.

as 8 on 16 peers, we have to fix as 4 on 32 peers and so on). Form figure 4.3 to figure 4.10, we show the experimental results concerning the accuracy and scalability of the algorithm by varying the number of peers for the SEQUOIA and DS1-Cure dataset.

For instance, on 80 peers, visiting only the 5% of points, on average, we obtain an accuracy of about 80% for Sequoia and about 85% for Cure and visiting the 10% of data we reach 93% and 95% of accuracy for the same datasets.

Furthermore, the scalability (i.e. the effect on the accuracy of increasing the number of peers and so reducing the number of data points for peer) is quite good for both the datasets. In fact, if look at the Cure dataset, for the 5% case, we obtained a reduction from 95% for 16 peers to 84% for 80 peers while for the 10% case, we have a little reduction from 97% to 95%. For Sequoia, for the same cases, we have respectively a reduction from 88% to 81% and from 99% to 94%. Visiting only 1% of the dataset we have low

percentage of point found, however they are sufficient to have an approximate idea of shape of the clusters.

4.3.2 The impact of Small World topology

During the first experimentations, peers were arranged using a logical ring topology. However, using this topology the merge phase can waste many iterations before to merge all the clusters. This is due mainly to the main characteristics of the ring topology, i.e. we have a high average hop count between two nodes. Furthermore, the fall of a node (event not infrequent in p2p networks) causes a dramatic increase in the hop count. On the other hand, if we used a completely connected topology, the network will be congested by the large number of messages exchanged. An interesting alternative is using the small world topology, introduced in [125] to describe the social phenomenon that two arbitrary persons in the world are linked by a short chain of acquaintances. We can characterize network topologies using two parameters: characteristic path length CPL (i.e. the length of the shortest path between each pair of nodes averaged over all possible pairs) and clustering coefficient CC (i.e. the ratio between the number of edges in the neighborhood of a node and the total number of possible edges averaged over all nodes). Small world topology, showed in figure 4.11 b have a high CC but a low CPL and that is a really useful property in p2p networks, as we need a few hops to reach a node, but the disconnection of a node does not change the behavior/performance of the system.

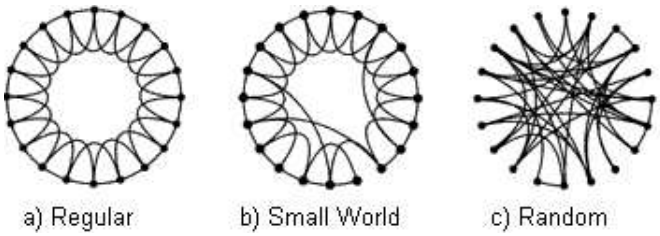


Fig. 4.11. (a) Regular Lattice $\beta = 0$ (b) Small World $\beta = 0.1$ (c) Random $\beta = 0.8$

Small world network can be built starting from a regular one dimensional lattice (an extension of ring topology, with each node having k neighbors instead of 1, see figure 4.11 a, with $k \ll N$ and $\frac{k}{2}$ edges to the left and $\frac{k}{2}$ to the right node respectively). We start from a node of the lattice and we rewire each its edge with a probability β, i.e. if the right probability is extracted, we deleted the edge and reconnect the node with another node randomly chosen among all the nodes. If the edge was already present in the network the older remains unchanged. The procedure go on for the next right neighbor node until all the graph is visited and it is repeated for the second link of each

node. The topology of the network depends on the parameter β, if it $\rightarrow 0$ we will have a regular lattice topology, if it varies from 0.01 to 0.1 we will have a small world topology and higher values will bring to a random topology (figure 4.11 c).

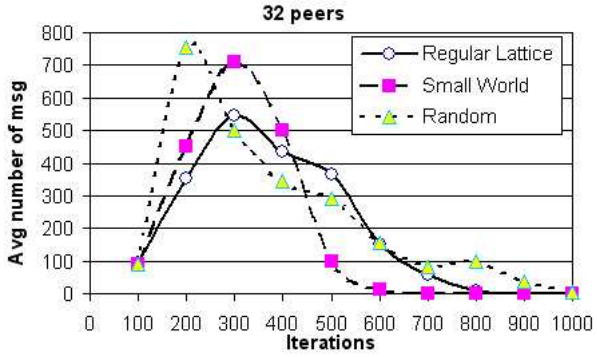


Fig. 4.12. Avg number of messages exchanged for the three different topologies (regular lattice, small world and random) using 32 peers for Sequoia.

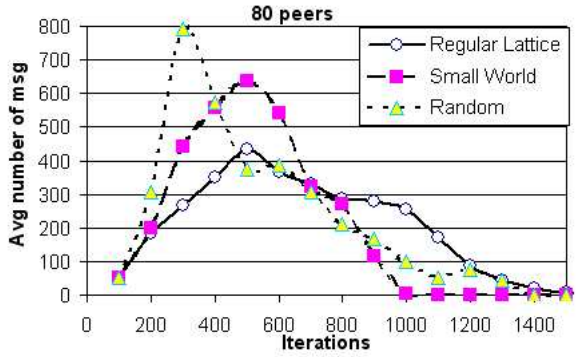


Fig. 4.13. Avg number of messages exchanged for the three different topologies (regular lattice, small world and random) using 80 peers for Sequoia.

We run P-Sparrow with the same parameters reported in the previous subsection using the three different topologies ($\beta = 0$, $\beta = 0.1$ and $\beta = 0.8$). In figure 4.12 and 4.13, we reported the average number of messages exchanged for peer (peers exchange core points each 100 generation) for the three different topologies for the Sequoia dataset. Note that when the number of messages approach to 0 means that the algorithm does not discover new solutions (core points) and then it converged. The SW topology reach a higher peak in com-

parison with the regular (but less than the random), but then converges more quickly, probably because of the effect of the long-range links that accelerates the diffusion of the core points and then the process of clustering. In the case of the random topology, the low clustering coefficient disperses many core points and this slow down the convergence.The same behavior was observed for different rate of exchanging messages (50 and 200) and for the DS1-Cure dataset, but the figures are not reported for lack of space.

5

An Ant-Inspired Protocol for Mapping Resources in Grid

5.1 Introduction

This chapter examines an approach based on ant systems to replicate and map Grid services information on Grid hosts according to the semantic classification of such services. A Grid information system should rely upon two basic features:

- the replication and dissemination of information about Grid services and resources;
- the distribution of such information among Grid hosts.

The ARMAP protocol(Ant-based Replication and MApping Protocol) aims to disseminate information in a controlled way, in order to maximize the benefit of the replication mechanism and facilitate discovery operations. Replicas are spatially mapped on the Grid so that resource descriptors belonging to the same class are placed in nearby Grid hosts. The mapping of resource descriptors is managed through a multi agent approach, inspired by the model that was introduced in [24]to emulate the behavior of ants which cluster and map items within their environment. This section proposes a variant of that model, in which items (in our case the resource descriptors) are both replicated and mapped. A number of agents traverse the Grid via the underlying P2P interconnections and copy or move resource descriptors from one host to another, by means of appropriate pick and drop random functions. In particular, each agent is tailored to pick resource descriptors of a given class from a region in which that class of resources is scarcely present, and drop them in a region where those resources are already being accumulated. A spatial entropy function is defined to evaluate the effectiveness of the ARMAP protocol in the logical reorganization of resources. Each agent can operate in either the copy modality or the move modality. If the copy modality is used, the agent, when executing a pick operation, leaves the resource descriptors on the current host, generates a replica of them, and carries the replicas until it will drop them in another host. Conversely, with the move modality, as an agent picks the

resource descriptors, it removes them from the current host, thus preventing an excessive proliferation of replicas. In a first phase, the copy modality is used to generate an adequate number of resource descriptor replicas on the network. However, the copy modality cannot be maintained for a long time, since eventually every host would have a huge number of resource descriptors of all classes, thus weakening the efficacy of resource mapping. Therefore, each generated agent must switch from the copy to the move modality, and a mechanism must be defined to determine the correct time at which this modality switch must be performed. The analysis of the entropy trend allowed for the definition of a decentralized self-organizing mechanism through which each agent can tune its modality by itself, by analyzing its activeness, i.e. the frequency of pick and drop operations it performs. This decentralized mechanism, inspired on ants' pheromone [8], guarantees that the ARMAP protocol is fully scalable and fault-tolerant, since agents can tune their behavior without having a global knowledge of the state of the system.

5.2 ARMAP protocol

ARMAP aims to disseminate Grid resource descriptors and spatially map them on the Grid according to their semantic classification, in order to gather a consistent number of resource descriptors of the same class in a restricted region of the Grid. It is assumed that the resources have been previously classified into a number of classes `Nc`, according to their semantics and functionalities (see [17] and [84]). ARMAP exploits the random movements and operations of a number of mobile agents that travel the Grid using P2P interconnections. This approach is inspired by biological systems, in particular by ant systems [8][18][24], in which swarm intelligence emerges from the collective behavior of very simple mobile agents (ants), and a complex overall objective is achieved. In ARMAP, each mobile agent can pick a number of resource descriptors on a Grid host, carry such descriptors while moving form host to host, and deposit them on another Grid host. Initially, it is assumed that each agent is "class-specific", i.e. it manages the resource descriptors of only one class. This assumption will be released later. Section 5.3 describes the basic features of the ARMAP protocol (agent movements and pick and drop operations) and the approach used to tackle the dynamic nature of the system. Section 5.4 introduces the entropy function used to evaluate the effectiveness of ARMAP, and explains the advantage of switching the ARMAP modality from copy to move. Furthermore, section 5.4 discusses a decentralized approach, based on ants' pheromone, that is used by each agent to evaluate the correct time at which it must operate the modality switch. Section 5.4.1 discusses the role of the ARMAP protocol in the design of a Grid information system.

5.3 Basic operations

5.3.1 Agent Movement

Each agent travels over the Grid through P2P interconnections among Grid hosts. The ARMAP protocol has been analyzed in a P2P network in which peers are arranged in a mesh topology, as in the SWARM simulator [86], and each peer is connected to at most 8 neighbor peers, including horizontal, vertical and diagonal neighbors. The mesh topology was chosen to achieve a more intuitive and immediate graphical representation of the system evolution, which helps to understand the involved dynamics. However, it is also assumed that peers frequently leave and re-join the network, as discussed later, which assures that at a specific time a peer is actually connected to a random number of active peers (at most 8), so relaxing the rigid mesh assumption. At random times, each agent makes a random number of hops along the P2P network (the maximum number of hops `Hmax` is a protocol parameter), executes the agent's algorithm specified by the ARMAP protocol, and possibly performs a pick or drop operation.

5.3.2 Pick operation

Once an agent specialized in a class Ci gets to a Grid host, if it is currently unloaded, it must decide whether or not to pick the resource descriptors of class Ci that are managed by that host. In order to achieve the replication and mapping functionalities, a pick random function `Ppick` is defined with the intention that the probability of picking the resource descriptors of a given class decreases as the local region of the Grid accumulates such descriptors. This way resource mapping is further facilitated. The `Ppick` random function, defined in formula 5.1, is the product of two factors, which take into account, respectively, the relative accumulation of resource descriptors of a given class (with respect to the other classes), and their absolute accumulation (with respect to the initial number of resource descriptors of that class).

$$Ppick = \left(\frac{k1}{k1 + fr}\right)^2 + \left(\frac{fa^2}{k2 + fa^2}\right)^2 \tag{5.1}$$

In particular, the fr fraction is computed as the number of resource descriptors of class `Ci` accumulated in the peers located in the visibility region divided by the overall number of resource descriptors that are accumulated in the same region. The visibility region includes all the peers that are reachable from the current peer with a given number of hops (i.e. within the visibility radius). Here it is assumed that the visibility radius is equal to one, so that the visibility region is composed of at most 9 hosts, the current one included. It is assumed that each host is informed, through a soft state mechanism, about the resource descriptors that are maintained by the hosts located within the

visibility region. The fa fraction is computed as the number of resource descriptors of class Ci that are owned by the hosts located in the visibility region out of the number of resource descriptors that are presently maintained by such hosts, including the descriptors deposited by the agents. The inverse of fa gives an estimation of the extent to which the hosts under interest have accumulated resource descriptors of class Ci so far. k1 and k2 are threshold constants which are both set to 0.1. If the ARMAP protocol works in the copy modality, when an agent picks the resource descriptors of class Ci, it leaves a copy of them in the current host; conversely, if the move modality is assumed, such resource descriptors are removed from the current host. In the latter case, the current host will only maintain the descriptors of class Ci that it owns, but loses all the information about the descriptors of class Ci which have been deposited by the agents.

5.3.3 Drop operation

Whenever an agent specialized in a class Ci gets to a new Grid host, it must decide whether or not to drop the resource descriptors of class Ci, in the case that it is carrying any of them. As opposed to the pick operation, the dropping probability is directly proportional to the relative and absolute accumulation of resource descriptors of class Ci in the visibility region. The `Pdrop` function is shown below.

$$Ppick = \left(\frac{fr}{k3 + fr}\right)^2 + \left(\frac{k4}{k4 + fa^2}\right)^2 \tag{5.2}$$

In formula 5.2, the threshold constants `k3` and `k4` are set to 0.3 and 0.1, respectively. A high-level description of the ARMAP algorithm executed by each agent is given in figure 5.1: the role of the protocol modality (copy or move) is highlighted. So far, only class-specific agents were considered: with this assumption, each peer casually selects the class of resources in which the generated agent will be specialized. However, to improve performance, it is also possible to generate generic agents, which are able to pick and drop resource descriptors belonging to all the resource classes or a subset of them. In such a case, the algorithm shown in 5.1 is slightly modified: the agent computes the pick and drop random functions separately for each class it can manage. This way an agent may pick the resource descriptors of class `Ci` from a Grid host, and drop the descriptors of another class `Cj` into the same host. The performance increase obtained with generic agents will be shown in section 5.7.

5.3.4 Dynamic Grid

In a dynamic Grid, peers can, more or less frequently, go down and reconnect again. As a consequence of this dynamic nature, two different and opposite issues must be tackled. The first one is related to the management of

new resources provided by new or reconnected hosts: once all the agents have switched to the move modality, it becomes impossible to replicate and disseminate information about the new resources; hence agents cannot live forever, and must be gradually be replaced by new agents that set off in the copy modality. At the same time, the system must deal with obsolete resources, i.e. with resources, provided by peers that has left the system, that are no more exploitable. To tackle these two issues, it is necessary to manage the lifecycle and the gradual turnover of agents, and control the overall number of agents that travel the Grid. The proposed solution is to correlate the lifecycle of agents to the lifecycle of peers. An assumption is made that the average amount of time for which a peer remains connected to the Grid is known and equal to `PlifeTime`. When joining the Grid, each peer generates a number of agents given by a discrete Gamma stochastic function, with average Ngen, and sets the life time of these new agents to `PlifeTime`. This setting assures that (i) the relation between the number of peers and the number of agents is maintained over the time (more specifically, the overall number of agents is approximately equal to the number of active peers times `Ngen`) and (ii) a proper turnover of agents is achieved, which allows for the dissemination of new resource descriptors, since new agents start with the copy modality. Furthermore, a peer, when leaving the Grid, looses all the information about the resource descriptor replicas that it has collected during its connection time. This solves the problem of managing obsolete resources: indeed the descriptors related to an obsolete resource are gradually eliminated by the Grid, as the peers that are storing such replicas leave the system and the agents that are carrying them expire. Finally, it is worth highlighting that the described approach implicitly manages any unexpected peer fault, because this occurrence is handled in exactly the same way as a peer disconnection is. Indeed, the two events are indistinguishable, since (i) a peer does not have to perform any procedure before leaving the system, and (ii) in both cases the resource descriptors that the peer has accumulated so far are discarded.

5.4 Spatial entropy and pheromone mechanism

A spatial entropy function is defined to evaluate the effectiveness of the ARMAP protocol, as shown in formula 5.4. For each peer p, the entropy `Ep` gives an estimation of the extent to which the visibility region centered in p has accumulated resource descriptors belonging to one class. This function, inspired by the Shannon entropy formula, is constructed upon the value of fr(i), defined as the fraction of resource descriptors of class Ci within the visibility region. The `Ep` formula is normalized, so that possible values are comprised between 0 (when all the resource descriptors in the visibility region belong to just one class) and 1 (when the resource descriptors are equally distributed among the different classes). The overall spatial entropy `E` of the

```
// Na = number of agents: each one is specialized in a class of resources
// Hmax = max number of P2P hops that an agent can perform between two
// successive operations
// mod =ARMAP modality (copy or move)
For each agent a (specialized in class Ci) do forever{
    Compute integer number h between 1 and Hmax;
    a makes h P2P hops;
    if (a is unladen) {
            compute Ppick;
            draw random real number r between 0 and 1;
            if (r¡=Ppick) then {
                pick resource descriptors of class Ci from current host;
                if (mod == move)
                    remove resource descriptors of class Ci from current host;
            }
    }
    else {
        compute Pdrop;
        draw random real number r between 0 and 1;
        if (r¡=Pdrop) then
            drop resource descriptors of class Ci into current host;
    }
}
```

Fig. 5.1. High-level description of the ARMAP algorithm

network is defined as the average of the entropy values `Ep` computed at all Grid hosts.

$$Ep = \frac{\sum_{i=1\ldots Nc} fr(i) * \lg \frac{1}{fr(i)}}{\lg Nc} \tag{5.3}$$

$$E = \frac{\sum_{p \epsilon Grid} Ep}{Np} \tag{5.4}$$

Simulation runs were executed to evaluate the correct time at which the modality switch (from copy to move) should be performed in order to minimize the spatial entropy. Results are given in section 5.6. The assumption here is that each agent knows the value of the overall entropy at every instant of time. However, in the real world an agent maintains only a local view of the environment, and cannot determine its behavior on the basis of global system parameters such as the overall system entropy. Therefore, a method is introduced with the purpose of enabling a single agent to perform the modality switch only on the basis of local information. Such a method is based on the observation that an increase in the overall entropy value corresponds to a significant decrease in the activity of agents, i.e. in the frequency of pick

and drop operations that are performed by agents. Indeed a low agent activity is a clue that a high degree of resource reorganization has already been achieved. Accordingly, a single agent can evaluate its own activeness by using a pheromone mechanism [97]. In particular, at given time intervals, i.e. every 2000 seconds, each agent counts up the number of successful and unsuccessful pick and drop operations (a pick or drop operation attempt is considered successful when the operation actually takes place - i.e. when the random number extracted is lower than the value of the operation probability function). At the end of each time interval, the agent makes a deposit into its pheromone base, by adding a pheromone amount equal to the fraction of unsuccessful operations with respect to the total number of operation attempts. An evaporation mechanism is used to give a higher weigh to the recent behavior of the agent. In more details, at the end of the i_{th} time interval, the pheromone level Φ_i is computed with formula 5.5.

$$\Phi = Ev * \Phi_{i-1} + \phi_i \tag{5.5}$$

The evaporation rate `Ev` is set to 0.9, and i is the fraction of unsuccessful operations performed in the last time interval. As soon as the pheromone level exceeds a given threshold `Tf`, the agent realizes that the frequency of pick and drop operations has remarkably reduced, and switches its protocol modality from copy to move. The value of `Tf` is set by observing the global system behavior, as explained in section 5.6. The choice of updating the pheromone level every time interval, instead of every single operation, has been made to fuse multiple observations into a single variable, so giving a higher strength to agents' decisions.

5.4.1 Design of P2P information systems in Grids

I believe that the ARMAP protocol can be a significant step towards the efficient design and implementation of a P2P-based information system in a Grid environment. However, to better understand its role, it is necessary to discuss how ARMAP can be related to the overall information system design process, which could be composed of the following three components/steps:

- classification of Grid resources;
- replication and mapping of resource descriptors with the ARMAP protocol;
- construction of a discovery service.

The first component allows users to identify the features and functionalities of the resources they need (i.e. a particular resource class). Classification of resources can be performed with different techniques, as discussed in section 5.9. The second component, ARMAP, is the subject of this chapter. The third component, the discovery service, assumes that the Grid resources have been logically reorganized through the ARMAP protocol, or at least that ARMAP is working while discovery requests are being forwarded. The use of ARMAP

permits to handle discovery requests by combining the flexible and scalable features of a blind approach with the efficiency and fastness of an informed approach. A discovery protocol that takes full advantage of the ARMAP work is briefly described in the following and is analyzed in following chapter. Query messages first travel the Grid network with a blind mechanism, according to the random walk technique [17]; however, the search procedure is turned into an informed one as soon as a query message approaches a region which has gathered resource descriptors belonging to the requested class. A number of peers, i.e. the peers that collect a large number of resource descriptors belonging to a specific class, are elected as representative peers, and assume the role of attractors for query messages. This way, a query can be routed towards the nearest representative peer of the class under consideration, and there it will easily find a large number of useful results.

5.5 Protocol Analysis

In this section we introduce and discuss the parameters and performance indices used to evaluate the ARMAP protocol (in section 5.5.1), than we report and discuss simulation results which demonstrate the protocol effectiveness in a Grid environment. In particular, the trend of the overall system entropy (Section 5.6) confirms the advantage of using the ARMAP protocol with two modalities, copy and move. In Section 5.6, it is also shown how agents can autonomously choose the protocol modality with a pheromone mechanism. Section 5.7 analyzes the performance of ARMAP achieved by varying the number, type and mobility of agents. Finally, section 5.8 investigates the impact of the number of resource classes and the Grid size on ARMAP behavior.

5.5.1 Simulation Parameters and Performance Indices

Simulation runs were performed by exploiting the software architecture and the visual facilities offered by the SWARM environment [86]. **SWARM** is a software package for multi-agent simulation of complex systems, developed at the Santa Fe Institute. Table 5.1 and Table 5.2 report, respectively, the simulation parameters and the performance indices used in our analysis. The number of peers `Np` (or Grid size) was varied from 225 (a 15x15 grid) to 10000 (a 100x100 grid). The number of resources (Grid services) owned and published by a single peer is determined through a gamma stochastic function having an average value equal to 15 (see [53]). Grid resources are classified in a number of classes `Nc` varying from 3 to 9; the class to which each resource belongs is selected by the simulator with an uniform random function. When an agent moves to a destination peer, it performs the algorithm shown in 5.1, possibly picks and/or drops a number of resource descriptors, and finally moves to another peer. Each peer generates a random number of agents with average equal to `Pgen`: by modulating this parameter, the overall number of

agents Na was varied from Np/4 to 2Np. Both class-specific and generic agents were considered in the simulation analysis. The average connection time of a peer (PlifeTime) is set to 100,000 sec. Each time a peer joins the Grid, it generates its current connection time by using a stochastic Gamma function with average PlifeTime. The average time between two successive agent movements (i.e. between two successive evaluations of the pick and drop functions) is set to 60 sec. To move towards a remote host, an agent exploits P2P interconnections among Grid hosts. The number of hops performed within a single agent movement is a random function: the maximum number of hops, Hmax, is varied from 1 to D/2, where D is the square root of Np. Finally, the visibility radius Rv, defined in section 5.3, is set to 1. Among the performance indices, the overall entropy E, defined in section 5.4, is the most important one, since it is used to estimate the effectiveness of the ARMAP protocol in the logical reorganization of resources. The Nrep index is defined as the mean number of replicas per resource that are available (i.e. that are maintained by active peers) on the Grid. Fop is the frequency of successful operations (pick and drop) that are performed by each agent; this index gives an estimation of agents' activeness and system stability, since successful operations are less frequent when a low level of entropy has been achieved. Finally, the traffic load L is defined as the number of hops per second that are performed by all the active agents.

Table 5.1. Simulation parameters

Parameter	**Symbol**	**Value**
Grid size (number of peer)	**Np**	225 to 10000
Mean number of resources published by a peer	*not used*	15
Number of classes of resources	**Nc**	3 to 9
Number of agents (class-specific or generic)	**Na**	**Np**/4 to 2**Np**
Mean life time of a peer	**Plifetime**	100,000s
Mean amount of time between two successive movements of an agent	*not used*	60 s
Maximum number of hops	**Hmax**	1 to **D**/2
Visibility radius	**Rv**	1

5.6 Protocol modality

The results shown in this section and in section 5.7 are relative to simulations performed for a network with 2,500 (50x50) hosts that provide resources belonging to 5 different classes, unless otherwise stated. In particular, results shown in this section are obtained by setting the number of agents Na to half

Table 5.2. Performance indices

Symbol	Definition
E	Mean Spatial Entropy - Defined in section 5.4
Nrep	Mean number of replicas per resource(included the original copy) which are available in the Grid
Fop	Mean number of successful operations - pick or drop - that are performed by a single agent per unit time (operations/s)
L	Mean number of hops that are performed by all the agents of the Grid per unit time (hops/s)

the number of peers **Np**(`Pgen` is set to 0.5), and the maximum number of hops `Hmax` to 3. All agents are generic, therefore they can pick and drop resource descriptors belonging to every class. A comparison with class-specific agents is shown in section 5.7. Graphs of performance measures, reported versus time, illustrate the gradual effect of the ARMAP protocol in the reorganization and mapping of resource descriptors. Figure 5.2 shows that the exclusive use of the copy modality is not effective: the overall system entropy decreases very fast in a first phase, but increases again when the agents create an excessive number of replicas (figure 5.3). Indeed, if agents continue to create new replicas, eventually all peers will possess a huge number of resource descriptors of all classes, thus completely undoing the mapping work. The curve labeled as "copy/move" in figure 5.2 is obtained by switching the protocol modality, from copy to move, when it is observed that the entropy function (calculated every 2000 seconds) increases two times in succession. The effect of this switch is very interesting: the system entropy not only stops increasing but decreases to much lower values. This behavior is the effect of the action of the agents, which do not generate further replicas, as shown in figure 5.3, but continue their work in creating low-entropy regions specialized in particular classes of resources. In these tests a global parameter of the system - the value of the overall system entropy - is assumed to be known by all the agents. In the following, this approach will be referred to as the global one. Figure 5.2 and 5.4 show that, with the copy modality, at approximately the same time as the system entropy begins to increase, the frequency of operations performed by a single agent stops increasing and begins to decrease. This experimental result can be exploited to define a local approach that allows agents to perform the modality switch on their own. This is achieved through the pheromone mechanism explained in section 5.4. However, it is necessary to set a proper value of the pheromone threshold **Tf**, that is used by agents to realize when the modality switch must be performed. This value was set with the following method: by running simulations with the global approach, we calculated the mean pheromone value of a generic agent at the time at which the system entropy

begins to increase. This value was used as the pheromone threshold in the local approach. Figure 5.5 and 5.6 compare the curves of system entropy and average number of replicas obtained with the global and local approaches. It is seen that the local approach is effective, since it approximates the global one very strictly. Figure 5.7 gives a graphical description of the mapping process: the 50x50 square grid represents the simulated network, and each cell represents a peer. To facilitate the comprehension of the process, the number of resource classes is set to 3. The local approach is assumed. Each cell is marked with a circle, a square or a cross: such symbols correspond to class **C1**, **C2** and **C3**, respectively. The presence of a circle means that in the corresponding node the number of resource descriptors of class C1 exceeds the numbers of descriptors belonging to the other 2 classes; the presence of a square or a cross has an analogous meaning. Furthermore, the thickness of the symbol represents how much the peer is specialized in one class, i.e. it is proportional to the difference between the number of descriptors belonging to the most numerous class and the number of descriptors belonging to the second most numerous class. To depict the gradual accumulation of resource descriptors, four snapshots of the network are shown. Such snapshots are taken at time 0 (when the ARMAP protocol is started), and after 25,000, 50,000 and 100,000 seconds, respectively. The sequence of snapshots confirms the effectiveness of the accumulation process, which is very fast in the first phase, thus allowing for a notable performance enhancement of discovery operations after a short amount of time.

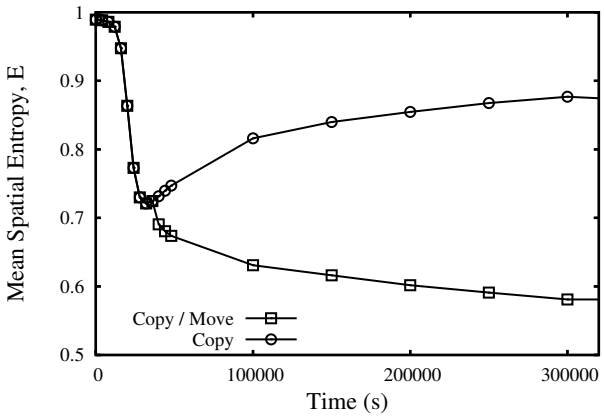


Fig. 5.2. Mean spatial entropy vs. time; comparison between ARMAP used with the copy modality and ARMAP with the copy/move modality switch.

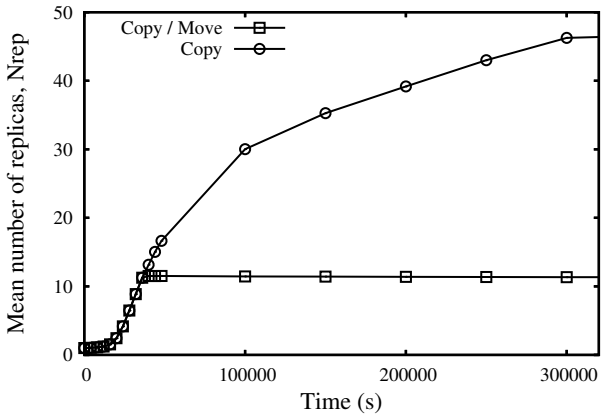


Fig. 5.3. Mean number of replicas vs. time; comparison between ARMAP used with the copy modality and ARMAP with the copy/move modality switch.

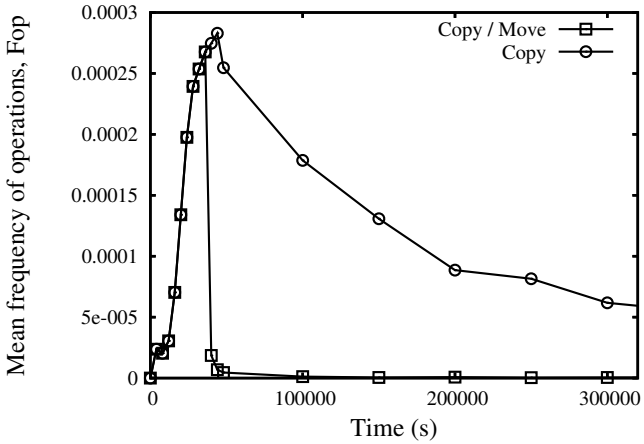


Fig. 5.4. Mean frequency of operations vs. time; comparison between ARMAP used with the copy modality and ARMAP with the copy/move modality switch.

5.7 Number and mobility of agents

In section 5.6 the number of agents `Na` was set to `Np/2`, and the parameter `Hmax` was set to 3. The aim of this section is to analyze how the performance of ARMAP is affected by the number, type and mobility of agents. A first consideration is that the value of the pheromone threshold `Tf` depends on the number of agents per peer, i.e. on the ratio `Na/Np`. Table 6.2 shows, for different values of the ratio Na/Np, the corresponding instants of time at which the switch modality should be performed (calculated with the global approach), and the mean pheromone levels reached by a generic agent at those instants.

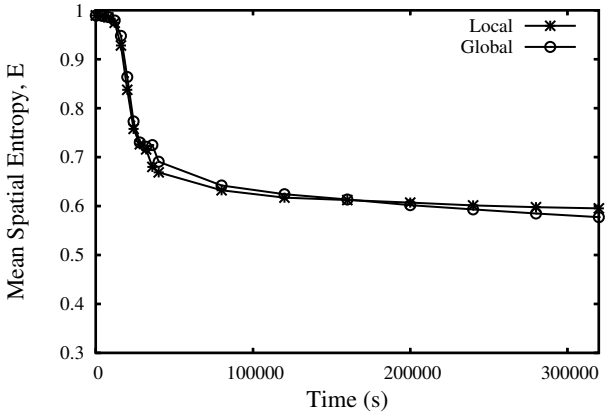


Fig. 5.5. Mean spatial entropy vs. time; comparison between global and local approaches.

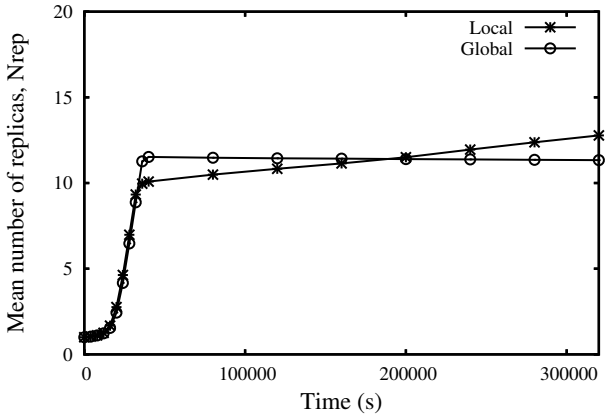


Fig. 5.6. Mean number of replicas vs. time; comparison between global and local approaches.

It can be observed that as the number of agents increases, the time interval after which such agents should stop creating new replicas becomes shorter, since the overall activity of agents is higher. As a consequence, the pheromone threshold `Tf` decreases as well: a single agent will reach the threshold after a shorter interval of time. Figure 5.8 reports the trend of the overall system entropy obtained with different numbers of agents; the local approach is used and the pheromone thresholds are set to the corresponding values shown in Table 6.2. An increase in the number of agents makes the system entropy decrease faster and reach lower values. However, a higher activity of agents also causes an increase in the traffic load (Figure 5.9). A correct setting of the ratio Na/Np should take into account the trend of these performance

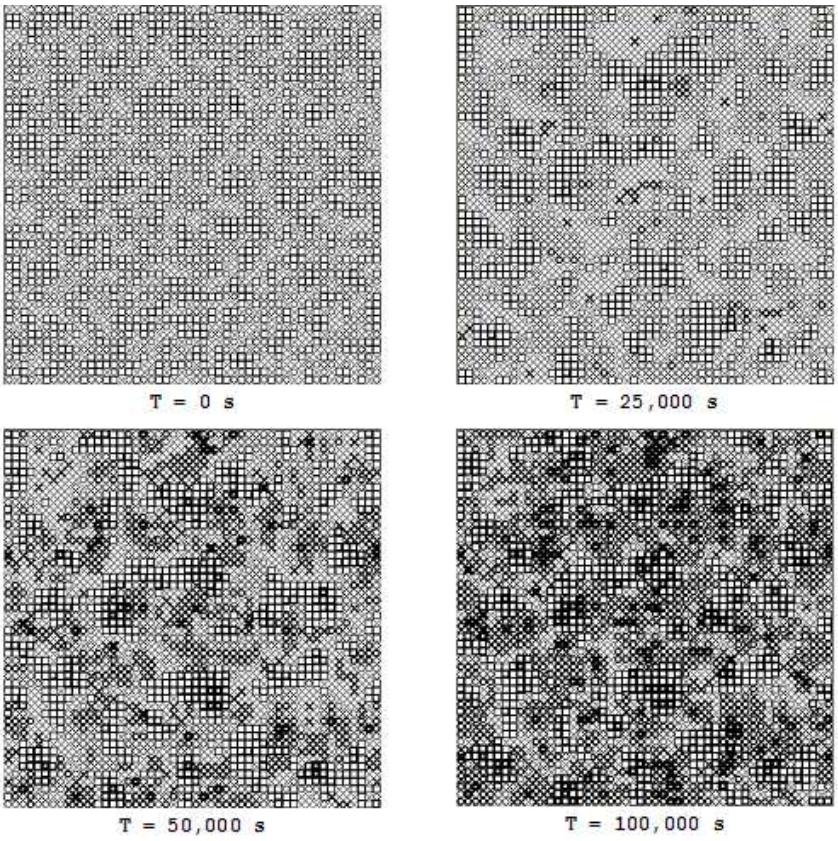


Fig. 5.7. Gradual mapping of resources in a network with 2,500 peers and 3 resource classes. Each peer contains a symbol (circle, square or cross) that corresponds to the class of resources which is the most numerous in such a peer. The symbol thickness represents the level of specialization of the peer.

indices and in general should depend on system features and requirements, for example on the system capacity of sustaining a high traffic load. Figure 5.10 compares the trend of the overall system entropy obtained with generic and specialized (class-specific) agents: in both cases the number of agents is set to Np/2. The performance increase achieved with the use of generic agents is confirmed, since they permit to obtain much lower values of the system entropy. Finally, the effect of the parameter `Hmax` is analyzed in Figures 5.11 - 5.13. Here, the number of (generic) agents Na is set to Np/2. An increase in `Hmax` speeds up the mapping of resources (Figure 5.11), since an agent can move resource descriptors between more distant peers, and causes an increase in the mean number of replicas (Figure 5.12) and in the traffic load

(Figure 5.15). Again, a correct setting of `Hmax` should take into consideration the network requirements. We chose to set `Hmax` to 3, because this is the minimum value that permits to move an agent between two visibility regions that are completely disjoint, given that the visibility radius is set to 1.

Table 5.3. Modality switch time and pheromone level at switch time with different numbers of agents.

Number of agents Na	**Modality switch time (s)**	**Pheromone level at switch time**
Np/4	64,000	8.33
Np/2	42,000	7.73
Np	22,000	6.17
2Np	16,000	5.20

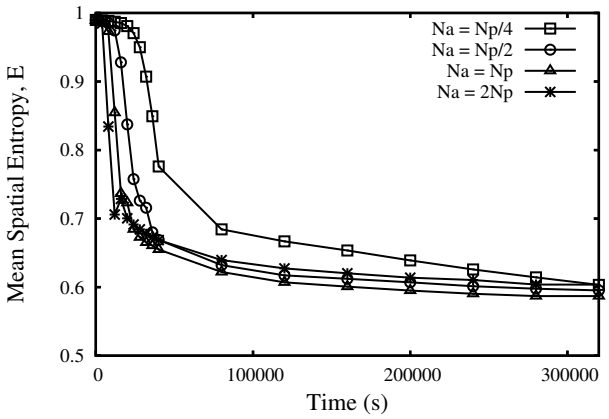


Fig. 5.8. Mean spatial entropy vs. time, with different numbers of agents.

5.8 Number of classes and network size

So far, it was assumed that the resources are pre-classified in exactly 5 classes, and that the network is composed of 2,500 (50x50) peers. The aim of this section is to investigate how the number of classes Nc and the Grid size `Np` affect the behavior of the ARMAP protocol. The results shown in this section are obtained with `Na=Np/2` and `Hmax`=3. The results shown in Table 5.4 were obtained by adopting the global approach with different values of Nc, and

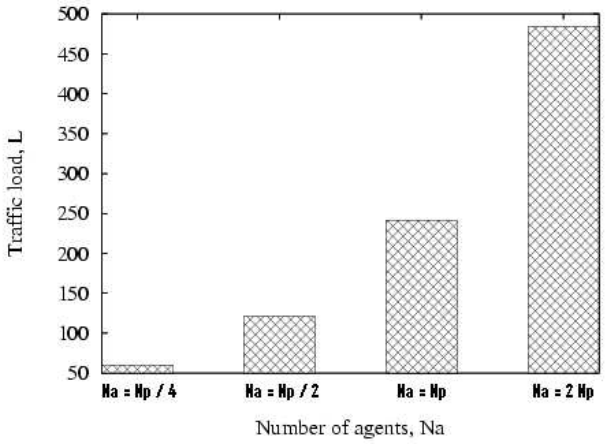


Fig. 5.9. Mean traffic load (hops/s), with different numbers of agents.

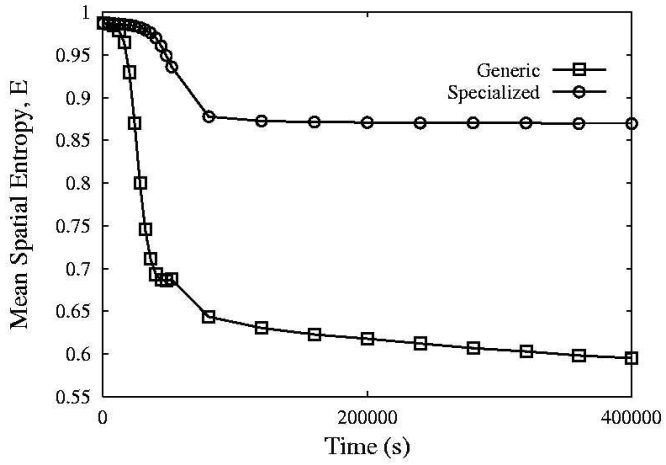


Fig. 5.10. Mean spatial entropy vs. time. Comparison between generic and specialized agents. The number of agents Na is set to Np/2.

Np equal to 2,500. As the number of classes increases, the optimum modality switch time increases. In fact, when the number of classes is higher, the agents have to replicate a larger number of resource descriptors to achieve a significant reduction of the system entropy. As a consequence, the pheromone threshold must be higher, as also appears from Table 5.4. Figure 5.14 shows the trend of the overall entropy achieved with the local approach and the

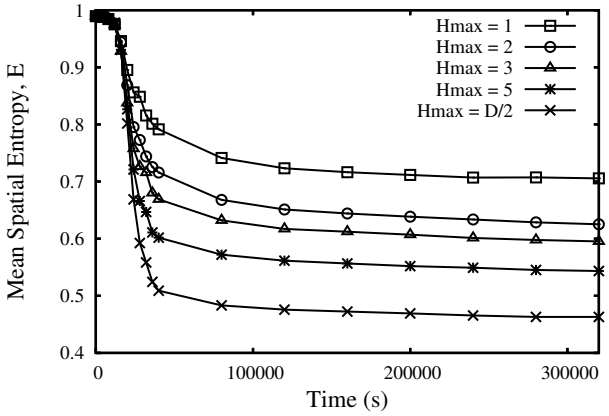


Fig. 5.11. Mean spatial entropy vs. time, with different values of Hmax.

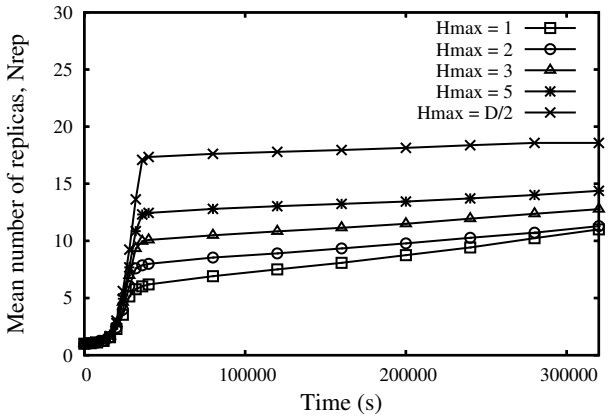


Fig. 5.12. Mean number of replicas vs. time, with different values of Hmax.

pheromone threshold values reported in Table 5.4. It is interesting to note that initially the entropy function decreases more rapidly when the number of classes is small; the reason is that in the first phase it is easier to separate resource descriptors belonging to 3 classes rather than 7 or 9. However, in the long term the agents' work permits to reach entropy values that are smaller and smaller as the number of classes increases. Indeed, the entropy curves cross and completely invert their order within a time interval time comprised between 25,000 and 50,000 seconds from the start of the simulation. A set of simulation runs was performed to evaluate the effect of the network size on performance results. Figures 5.15 and 5.16 show, respectively, the overall entropy and the mean number of replicas for different values of the number of peers Np. The number of classes is set to 5. It can be seen that the effect of

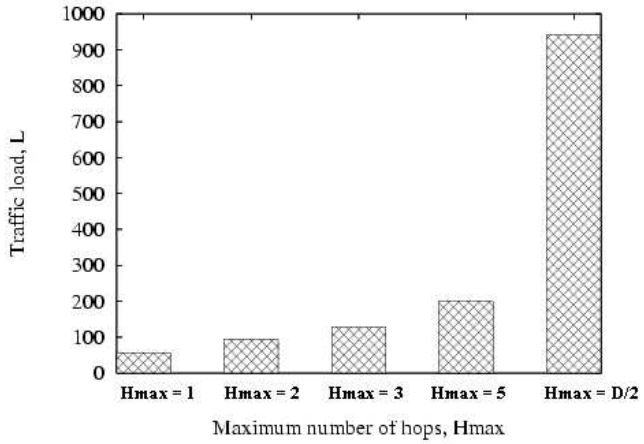


Fig. 5.13. Mean traffic load (hops/s), with different values of Hmax.

the Grid size is almost irrelevant: we can deduce that a single peer does not need to have any knowledge about the size of the network to regulate its behavior. Other simulation results, not reported here, show that also the mean number of resources published by each peer has little effect on the behavior of the ARMAP protocol. As a conclusion, an effective tuning of ARMAP can be obtained by setting the pheromone threshold to a proper value, as shown in Tables 5.3 and 5.4. This value essentially depends on two parameters: the number of agents per peer, i.e. the ratio Na/Np, and the number of classes Nc. The former parameter is known because it is equal to `Pgen`, the average number of agents that a peer generates when joining the Grid. The latter parameter is also known if it is assumed that the resources are categorized in a predetermined number of classes.

Table 5.4. Modality switch time and pheromone level at switch time with different numbers of classes of resources.

Number of classes	**Modality switch time (s)**	**Pheromone level at switch time**
3	40,000	7.69
5	42,000	7.73
7	48,000	7.97
9	56,000	8.18

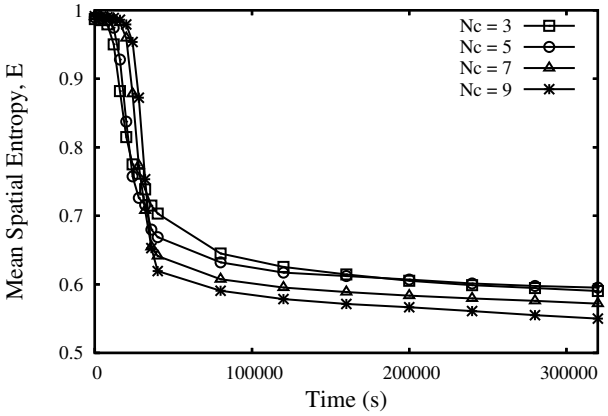


Fig. 5.14. . Mean spatial entropy vs. time, with different values of the number of classes Nc.

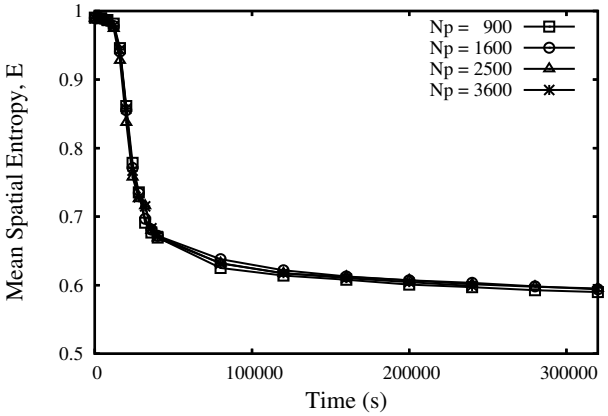


Fig. 5.15. Mean spatial entropy vs. time, with different values of the network size Np.

5.9 Related Work

Since Grid hosts provide a large set of distributed and heterogeneous resources, an efficient Grid information service is a pillar component of a Grid. Current Grid information services offer centralized or hierarchical information services, but this kind of approach is going to be replaced by a decentralized one, supported by P2P interconnection among Grid hosts. In the last years, a number of P2P techniques and protocols have been proposed to deploy Grid information services: for example, super-peer networks [84][128] achieve a balance between the inherent efficiency of centralized search, and the autonomy, load balancing and fault-tolerant features offered by distributed search. P2P search

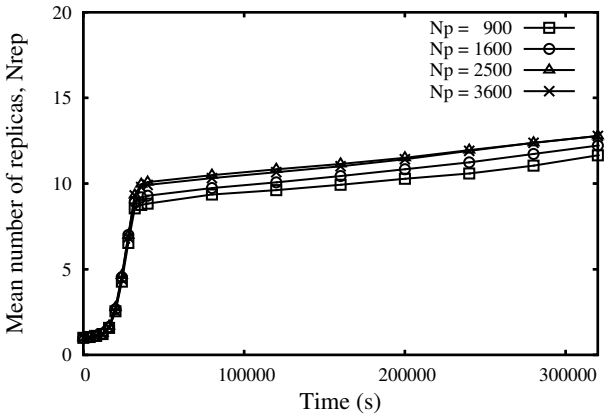


Fig. 5.16. Mean number of replicas vs. time, with different values of the network size Np.

methods can be categorized as structured or unstructured. The structured approach assumes that hosts and resources are made available on the network with a global overlay planning. In Grids, users do not usually search for single resources (e.g. MP3 or MPEG files), but for software or hardware resources that match an extensible set of features. Accordingly, while structured protocols, based on highly structured overlays and Distributed Hash Tables (e.g. Chord [111]), are usually very efficient in file sharing P2P networks, unstructured or hybrid protocols seem to be preferable in largely heterogeneous Grids. Unstructured search methods can be further classified as blind or informed [119]. In a blind search (e.g. using flooding or random walks [80]), nodes hold no information that relates to resource locations, while in informed methods (e.g. routing indices [17] and adaptive probabilistic search [118]), there exists a centralized or distributed information service that drives the search for the requested resources. As discussed in section 5.4.1, the approach presented in this section aims to combine the flexible and scalable features of a blind approach with the efficiency and rapidity of an informed approach. The ARMAP protocol introduced in this chapter is based on the features of Multi-Agent Systems (MAS), and in particular of ant-based algorithms. A MAS can be defined as a loosely coupled network of problem solvers (agents) that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver [114]. Research in MASs is concerned with the study, behavior, and construction of a collection of autonomous agents that interact with each other and the environment. Ant-based algorithms are a class of agent systems which aim to solve very complex problems by imitating the behavior of some species of ants [8]. The Anthill system [5] is a framework that supports the design, implementation and evaluation of P2P applications based on multi-agent and evolutionary programming. In Anthill, societies of

adaptive agents travel through the network, interacting with nodes and cooperating with other agents in order to solve complex problems. Reference [18] introduces an approach based on ant behavior and genetic algorithms to search resources on a P2P network. A sub-optimal route of query messages is learnt by using positive and negative pheromone feedbacks and a genetic method that combines and improves the routes discovered by different ants. Whereas in [18] the approach is tailored to improve search routes with a given distribution of resources in the network, this work proposes to logically reorganize and replicate the resources in order to decrease the intrinsic complexity of discovery operations. Instead of directly using ant-based algorithms to search resources, the ARMAP protocol exploits an ant-based replication and mapping algorithm to replicate and reorganize resource descriptors according to their categorization. Our protocol is a variant of the basic sorting algorithm proposed in [24]. However, the latter assumes that only one item can be placed in a cell of a toroidal grid, and items can only be moved form one cell to another, without the possibility of performing any replication. Conversely, the ARMAP protocol assumes that a cell (i.e. a Grid host) can store several items (i.e. resource descriptors) and agents can create many replicas of the same item. The problem of driving the behavior of a single agent, which should autonomously be able to take actions without having an overall view of the system, is discussed in [97]. There, a decentralized scheme, inspired by insect pheromone, is used to limit the activity of a single agent when it is no more concurring to accomplish the system goal. A similar approach is used to drive the behavior of an agent, in particular to evaluate the correct time at which it should switch from the copy to the move modality. Information dissemination is a fundamental and frequently occurring problem in large, dynamic, distributed systems, since it consents to lower query response times and increase system reliability. Reference [52] proposes to disseminate information selectively to groups of users with common interests, so that data is sent only to where it is wanted. In this work, instead of classifying users, it is proposed to exploit a given classification of resources: resource descriptors are replicated and disseminated with the purpose of creating low-entropy regions of the network that are specialized in a specific class of resources. The so obtained information system allows for the definition and usage of a semi-informed search method, as explained in section 5.4.1. The ARMAP protocol assumes that a classification of resources has already been performed. This assumption is common in similar works: in [17], performance of a discovery technique is evaluated by assuming that resources have been previously classified in 4 disjoint classes. Classification can be done by characterizing the resources with a set of parameters that can have discrete or continuous values. Classes can be determined with the use of Hilbert curves that represent the different parameters on one dimension [3]; alternatively, an n-dimension distance metric can be used to determine the similarity among resources [69].

6

Discovering Categorized Resources in Grids

6.1 Introduction

To discover Grid resources having specific characteristics, i.e. belonging to a given class, we showed, in this section, the ARDIP protocol (Ant-based Resource DIscovery Protocol). The objective of ARDIP is to drive a query message towards a region of the Grid in which the needed class of resources is being accumulated. Because ARDIP fully exploits the replication and spatial mapping of resources achieved by ARMAP, the two protocols should be used together: as ARMAP agents perform the logical reorganization of resources and build the Grid information system, the number of useful resources that can be discovered by a query message increases and at the same time the query response time decreases, as shown in 6.5.2. The ARDIP protocol is based upon three modules:

- a module for the identification of representative peers which work as attractors for query messages;
- a module which defines the semi-informed search algorithm;
- a stigmergy mechanism that allow query messages to take advantage of the positive outcome of previous search requests.

These modules are described in the following.

6.2 Identification of representative peers

As resources of a given class are accumulated in a Grid region, the peer that, within this region, collects the maximum number of resources belonging to that class is elected as a representative peer for that class. The objective of a search operation is to let a query message get to a representative peer in an amount of time as low as possible, since such a peer, as well as its neighbors, certainly manages a large number of useful resources. Each peer periodically verifies if it can elect itself as a representative peer for a class of resources.

The ARDIP protocol assumes that a peer is a representative peer of a given class if the following condition is satisfied:

- the peer maintains a number of logical resources (i.e. of metadata documents) of that class that exceeds `Fg` times the mean number of physical resources, belonging to the same class, which are offered by a generic peer; hereafter, `Fg` is referred to as gathering factor.

Moreover, to limit the number of representative peers in the same region, each representative peer periodically checks if other representative peers are present in its neighborhood, specifically within the comparison radius `Rc`: two neighbor representative peers must compare the number of resources that they maintain, and the "loser" will be downgraded to a simple peer.

6.3 Semi-informed search

When a client or user needs to discover resources belonging to a given class, a number of asynchronous query messages are issued by ARDIP. The semi-informed search algorithm includes a blind search phase and an informed search phase. For the blind search phase, the random walk paradigm [80] is used: the query messages travel the Grid through the P2P interconnections by following a random path. The network load is limited by means of a caching mechanism that avoids the formation of cycles and the use of a time-to-live parameter TTL: the TTL value is equivalent to the maximum number of hops that can be performed by a query message before being discarded. The blind search procedure is switched to an informed one as soon as one of the issued query messages approaches a representative peer of the class of interest, i.e. when such a message is delivered to a peer which knows the existence of a representative peer and knows a route to it (see the description of the stigmergy module below). During the informed search phase, the query is driven towards the representative peer, and the TTL parameter is ignored so that the query cannot be discarded until it actually reaches the representative peer. Therefore, the semi-informed walk of the query message ends in one of the two following cases: (i) when the TTL is decremented to 0 during the blind phase; (ii) when the query reaches a representative peer. In both cases a queryHit message is created, and all the resources of the class of interest, which are found in the current peer, are put in this message. The queryHit follows the same path back to the requesting peer and, along the way, collects all the resources of the class of interest that are managed by the peers through which it passes.

6.4 Stigmergy mechanism

As showed in chapter 2, Grass coined the term "stigmergy" in the 1950's [40] to describe a broad class of multi-agent coordination mechanisms that rely

on information exchange through a shared environment. For example, in ant colonies, each ant adjusts its activity according to the state of the environment, which in turn can be modified by the activity of other ants. An ant that finds a food source leaves a pheromone along its way back to the nest, and such a pheromone will signal to other ants the presence of the food source. The term is formed from the Greek words stigma "sign" and ergon "action", and captures the notion that an agent's actions leave signs in the environment, signs that it and other agents sense and that determine their subsequent actions. Two varieties of stigmergy are distinguished [11]. Such a distinction is whether the signs consist of special markers that agents deposit in the environment ("marker-based" or "sign-based" strigmergy) or whether agents base their actions on the current state of the environment ("sematectonic" or "cue-based" strigmergy). The approach proposed in this chapter uses both types of stigmergy. Indeed sematectonic stigmergy is used by ARMAP agents to choose their operation modality (copy or move), as mentioned in previous chapter. Conversely marker-based stigmergy is exploited by the ARDIP protocol: when a query accidentally gets to a representative peer for the first time, the returning queryHit will deposit an amount of pheromone in the peers that it encounters as it retreats from the representative peer. In this work, it is assumed the pheromone is deposited only in the first two peers of the queryHit path. A pheromone base is maintained for each resource class, and information is given about the right direction (i.e. the next neighbor peer) to get to the representative peer of the desired class. Accordingly, when a query gets to a peer along its blind search, it checks the amount of pheromone which has been deposited in that peer for the resource class of interest; if the pheromone exceeds a threshold `Tf`, it means that a representative peer is close, so the search becomes informed and the query is driven towards the representative peer. Since the set of representative peers can change as the ARMAP clustering process proceeds, an evaporation mechanism is defined to assure that the pheromone deposited on a peer does not drive queryHits to ex-representative peers. The pheromone level at each peer is computed every time interval of 5 minutes, which is equal to the mean time interval between the issue of two successive query messages from a peer (see Table 6.1 in Section 6.5.1). The amount of pheromone Φ, after the i_{th} time interval, is given by formula 6.1.

$$\Phi = Ev * \Phi_{i-1} + \phi_i \tag{6.1}$$

The evaporation rate `Ev` is set to 0.9; i is equal to 1 if a pheromone deposit has been made in the last time interval by at least one agent, otherwise it is equal to 0. The pheromone level can assume values comprised between 0 and 10: the superior limit can be obtained by equalizing Φ_i to Φ_{i-1} and setting Φ_i to 1 in formula 6.1. The threshold `Tf` is set to 2. With these parameters, it is assured that the threshold is exceeded as soon as a few number of queryHits deposit their pheromone in different time intervals, while the algorithm is more conservative when it has to recognize that a representative peer has

been "downgraded": indeed, up to 15 time intervals are necessary to let such a level assume a value lower than the threshold. It can also happen, though quite rarely, that a peer collects pheromone deposited by two (or more) different queryHits which are related to the same resource class but come from two (or more) different representative peers. To which neighbor should a query for that resource class be forwarded. In such cases the peer maintains a different pheromone quantity for each neighbor. The query is forwarded to the neighbor peer associated to the higher amount of pheromone, provided that it succeeds the threshold. It corresponds to sending the query to the "oldest" representative peer, which intuitively is most likely the representative peer that has collected the largest number of resources so far.

6.5 Protocol Analysis

6.5.1 Description of the environment

A wide set of simulation runs were performed by exploiting a simulation environment, the core component of which is a Java event-based object-oriented simulator, that has already been used for other research issues [84]. In the simulator, both peers and agents are associated to objects which communicate among them through events. Such events are ordered on the basis of their expiration time, i.e. the time at which they have to be delivered to destination objects. When an event is received by an object, the latter (e.g. a peer) reacts according to its automaton, and may send other events to other objects (e.g. an ARDIP query message to neighbor peers). Furthermore, the simulator was extended in order to integrate and exploit a set of libraries offered by the SWARM environment [86]: graphical libraries which allowed for monitoring the behavior of the ARMAP and ARDIP protocols, and libraries for setting the network topology. For each simulation session, 10 simulation runs were performed and, by tuning the length of each run, all performance values were computed with a pre-determined statistical relevance, i.e. with at least 0.95 probability that the statistical error was below 5%. Table 6.1 reports the main parameters used in the simulation analysis, categorized in network parameters, ARMAP parameters and ARDIP parameters. The number of peers `Np` (or Grid size) is set to 2500, and each peer is connected to a maximum of 8 neighbor peers, as discussed before. Grid resources are classified into `Nc` different classes, with `Nc` set to 5 in this work. The number of physical Grid resources owned and published by a single peer is determined through a Gamma stochastic function having an average value equal to 15 (see [53]). Such resources are uniformly distributed among the `Nc` classes.

ARMAP parameters are defined in the second section of Table 6.1; their impact was analyzed in previous chapter. Each peer, whenever connecting to the Grid, spawns a random number of ARMAP agents; this number is generated by a Gamma random function the average of which is equal to

Table 6.1. Network, ARMAP and ARDIP parameters

Grid Parameter	**Symbol**	**Value**
Grid size (number of peer)	**Np**	2500
Maximum number of neighbor peers	*not used*	8
Mean number of resources published by a peer	*not used*	15
Number of resource classes	**Nc**	5
ARMAP Parameter	**Symbol**	**Value**
Number of ARMAP agents	**Na**	Np/2
Mean life time of a peer	**Plifetime**	100,000 s
Mean time interval between two successive movements of an ARMAP agent	*not used*	60 s
Maximum number of hops for each ARMAP agent's movement	**Hmax**	3
Visibility radius	**Rv**	1
ARDIP Parameter	**Symbol**	**Value**
Mean query generation frequency	*not used*	1/300 (1/s)
Number of query messages generated when issuing a query	**Nqm**	2-8
Time to live	**TTL**	3-7
Gathering factor for the identification of representative peers	**Fg**	24-48
Comparison radius for the identification of representative peers	**Rc**	2
Mean message processing time	*not used*	50 ms
Mean link delay	*not used*	50 ms

`Pgen`. In this work, `Pgen` was set to 0.5, so that the number of agents Na is approximately equal to half the number of peers `Np`. The average connection time of a peer (`PlifeTime`) is set to 100,000 sec. At random times, specifically every 60 seconds on average, each agent moves in the P2P network (each movement is composed of a number of hops not greater than `Hmax`, which is set to 3), and possibly performs a pick or drop operation, as described in the previous chapter. The visibility radius `Rv`, defined in Section 5.3, is set to 1. ARDIP parameters are defined in the last section of Table 6.1. When a peer issues a query for a class of resources (queries are uniformly distributed among the Nc classes), a number `Nqm` of asynchronous query messages, with `Nqm` set to values ranging from 2 to 8 in different simulation sessions, are forwarded to neighbor peers. They follow different directions and move in parallel through the Grid according to the semi-informed mechanism described in section 6.3. The `TTL` parameter was varied from 3 to 7. Table 6.1 also specifies the values of the gathering factor `Fg` and of the comparison radius `Rc`, used to elect the representative peers. Finally, the mean amount of time for processing a query

or queryHit message and the mean link delay between two neighbor peer were both assumed to be 50 ms on average, with Gamma statistical distributions.

6.5.2 Performance

The performance of the ARDIP protocol was analyzed by evaluating the performance indices defined in Table 6.2. In the following such indices are plotted w.r.t. time, to evaluate the expected outcome of a discovery request at different stages of the ARMAP process. It is worth highlighting that results obtained at time 0 are achieved without exploiting the ARDIP informed phase, because no representative peers have been selected yet. Therefore it is possible to compare the ARDIP protocol with the classical random walk technique, since in the ARDIP blind phase the random walk paradigm is adopted. The Nrep index is the overall number of representative peers (of all classes) that are selected by ARDIP with the mechanism described in Section 6.2. Figure 6.1 depicts the trend of this index evaluated as the ARMAP mapping process proceeds. Curves obtained with different values of the gathering factor `Fg` are compared: since `Fg` is a threshold used to elect the representative peers, obviously its increase causes a decrease in the number of representative peers. The gathering factor is hereafter set to 36. With this value, `Nrep` converges to a value slightly higher than 200, corresponding to about 40 representative peers (out of 2500 peers) per class. A snapshot of the state of the network was taken 300,000 seconds after the start of the ARMAP process (this value was chosen because at that time the system has reached a stable condition, as can be observed in previous figures) and is shown in figure 6.2. For two of the five resource classes, this figure confirms that representative peers are located in the core of the respective accumulation regions, and that nearby peers are able to drive query messages to a representative peer. The presence of a significant number of representative peers of the same class guarantees a high probability of reaching at least one of those in a limited amount of time.

The index `Fsq` is defined as the fraction of queries for which at least one of the `Nqm` issued query messages enters the informed phase and gets to a representative peer. In the following, a query message that reaches a representative peer, as well as the related query request, are referred to as striking message and striking query, respectively. Since the ARDIP discovery protocol aims at driving query messages towards representative peers, the index `Fsq` is essential to evaluate the extent to which the replication and organization of resources performed by ARMAP helps to increases the effectiveness of ARDIP. Figures 6.3 and 6.4 confirm the valuable effect caused by the combined use of ARMAP and ARDIP protocols. In fact, after a very small amount of time, the work of ARMAP produces a significant increase in `Fsq`. Results in figure 6.3 are obtained by setting the parameter `Nqm` to 4 and varying the TTL parameter from 3 to 7, whereas in figure 6.4 the TTL was set to 5 and `Nqm` was varied from 2 to 8.

Table 6.2. Performance indices for the ARDIP protocol

Performance Index	Definition
Nrep	Mean number of representative peers of all classes that are selected by ARDIP to attract query messages
Fsq	Fraction of query requests that are successfully driven towards a representative peer
Nres, **Nres(stk)**, **Nres(no-stk)**	Mean number of resources that a node discovers after its query (computed for all the queries and separately for striking and non-striking queries).
Tavg, **Tavg(stk)**, **Tavg(no-stk)**	Mean amount of time that elapses between the generation of a query and the reception of a corresponding queryHit (computed for all the queries and separately for striking and non-striking queries).
Tfst, **Tfst(stk)**, **Tfst(no-stk)**	Mean amount of time that elapses between the generation of a query and the reception of the first corresponding queryHit (computed for all the queries and separately for striking and non - striking queries).
Lq, **Lq(rep)**	Mean frequency of query messages (messages/sec) received by a peer (calculated for all the peers and for the representative peers only)

Figure 6.3 shows that `Fsq` increases with the value of TTL, since a search request can extend the blind search phase and has more chances to get to a representative peer. An analogous effect is achieved by increasing the `Nqm` parameter, since representative peers are searched by a larger number of query messages in parallel. The most important performance measure is obviously `Nres`, the mean number of results that are discovered by a query - to be precise by all the query messages generated by a single query request - issued to search resources belonging to a specific class. Indeed, it is generally argued that the satisfaction of the query depends on the number of discovered resources returned to the user that issued the query: for example, in [127] a resource discovery operation is considered satisfactory only if the number of results exceeds a given threshold. Figure 6.5 shows the number of results that are collected on average by striking queries, by non-striking queries, and by all the queries, for two different values of TTL and `Nqm` set to 4. Many interesting conclusions can be drawn by observing this figure. The most obvious one is that the number of results increases with the TTL value, since a larger fraction of the network can be explored; however response times and traffic increase as well, as shown later.

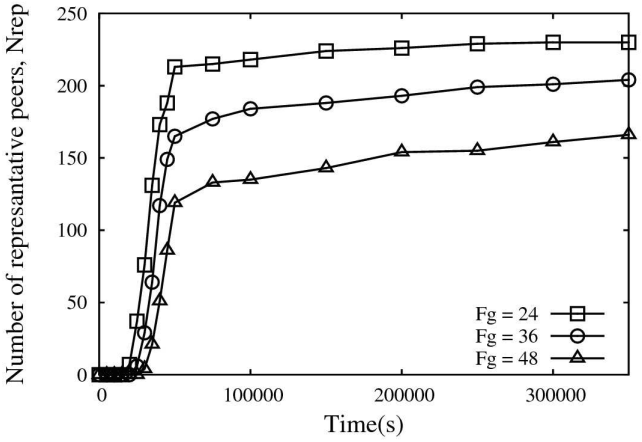


Fig. 6.1. Selection of representative peers as the ARMAP mapping process proceeds. The number of representative peers is reported for different values of the gathering factor Fg.

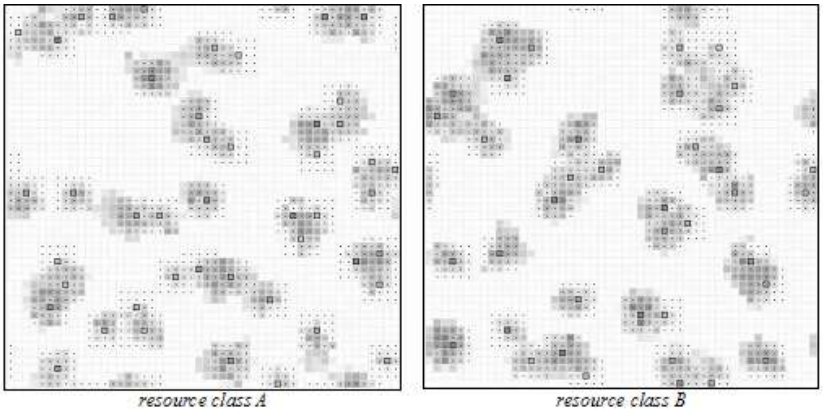


Fig. 6.2. Representative peers, belonging to two classes (out of five), selected by ARDIP 300,000 seconds after the start of the ARMAP process, with a gathering factor Fg set to 36. In the two figures, the squares are the representative peers respectively belonging to classes A and B, the dots are the peers that are able to drive the query messages to a nearby representative peer, and the number of resources of the respective classes is represented with a grey scale.

More important, the mean number of results increases as resources are being organized by ARMAP, meaning that specialized regions are able to offer a significant number of resources. Moreover, it is noted that striking queries

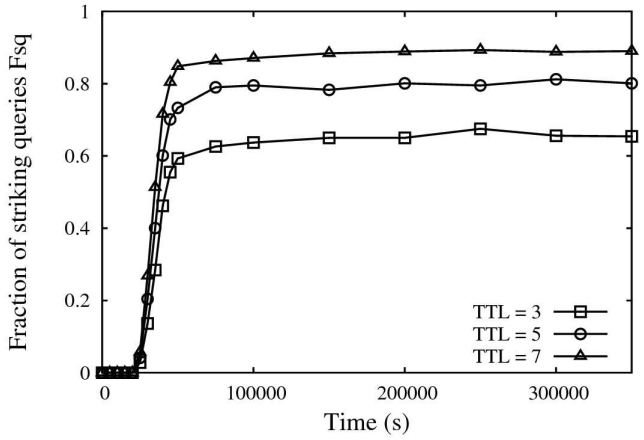


Fig. 6.3. Use of representative peers as the ARMAP process proceeds: fraction of search requests that are successfully driven to a representative peer, for different values of TTL and Nqm set to 4.

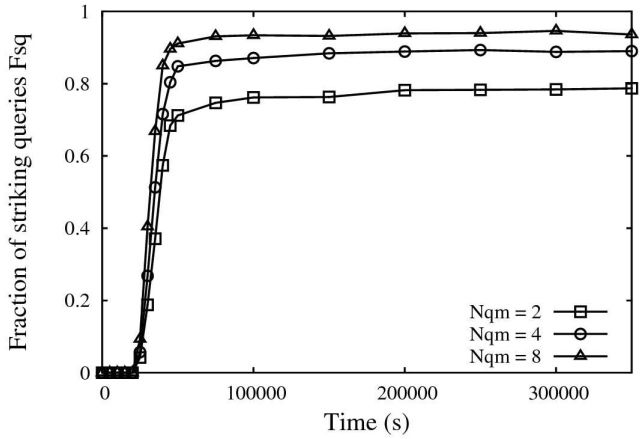


Fig. 6.4. Use of representative peers as the ARMAP process proceeds: fraction of search requests that are successfully driven to a representative peer, for different values of Nqm and TTL set to 7.

can discover considerably more results than non-striking queries, and such a difference increases with time, mostly because the progressive clustering of resources reduces the number of results that can be collected by non-striking queries (which however tend to be very rare, as figure 6.4 shows). Similar

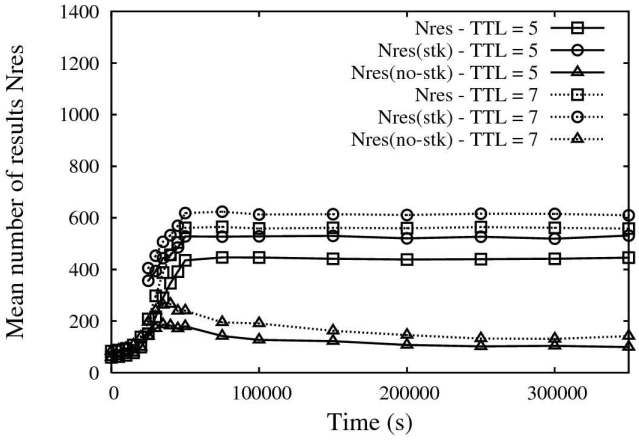


Fig. 6.5. Mean number of results achieved as the mapping process proceeds. Performance values, calculated for different values of TTL and Nqm set to 4, are reported for all the queries, and separately for striking and non striking queries.

considerations raise from the observation of figure 6.6, in which the TTL value is set to 7 and the number of query messages is varied from 2 to 8. The number of results increases with the `Nqm` parameter, but the traffic load increases as well, as will be shown in the following. The use of the ARMAP/ARDIP protocols not only increases the number of results, but also allows users to discover them in a shorter amount of time.

Figure 6.7 and 6.8 show the average response time experienced by varying the TTL value and the parameter `Nqm`, respectively. From figure 6.7, it appears that a higher TTL causes an increase in the response time, due to the fact that the queryHit messages have to travel a longer path. Whereas the response time of non-striking queries is almost constant over time, since all the query messages exploit the entire value of the TTL parameter, the reorganization of resources produces a significant decreases of average response times for striking queries and, since the relative weight of striking queries increases, for generic queries as well. In fact, when a query message gets to a representative peer, the discovery operation is stopped even if the TTL value is still greater than 0, and a queryHit is immediately issued. Therefore the presence of a striking message decreases the average response time. This can be clearly observed in figure 6.8: the average response time of a striking query decreases with the value of `Nqm` because the relative impact of the response time experienced by the striking message is higher for a lower number of query messages. Figures 16 and 17 depict the values of the response times corresponding to the first received queryHit, obtained by varying the TTL value and the parameter

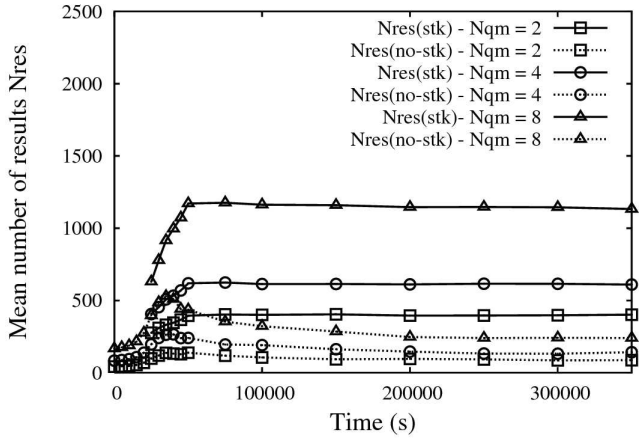


Fig. 6.6. Mean number of results achieved as the mapping process proceeds. Performance values, calculated for different values of Nqm and TTL set to 7, are reported separately for striking and non striking queries.

`Nqm`, respectively. These response times are significantly lower than average response times (figures 6.7 and 6.8), due to two phenomena.

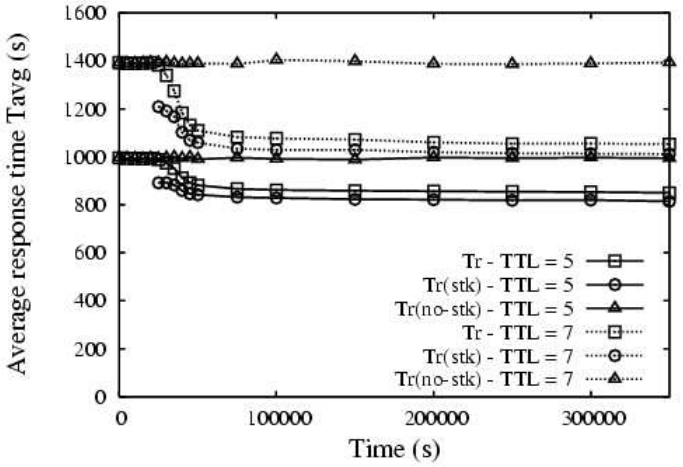


Fig. 6.7. Average response time achieved as the mapping process proceeds. Performance values, calculated for different values of TTL and Nqm set to 4, are reported for all the queries and separately for striking and non- striking queries.

The first one is related to the statistical nature of message processing time (the minimum processing time is lower than the average), and has impact on all the queries; the second one, which has impact only on striking queries, comes from the fact that the first queryHit most likely corresponds to a striking query message, which generally performs a lower number of hops. It is also interesting to note, from figure 6.10, that the time interval necessary to obtain the first result decreases as the `Nqm` parameter increases, which is different to what happens on average response times shown in figure 6.8. The reason is that a larger number of query messages allows for a more massive exploration of the Grid and hence a faster discovery of representative peers. The trend of the query traffic load, defined in the last row of Table 6.2, is depicted in figures 6.11 and 6.12.

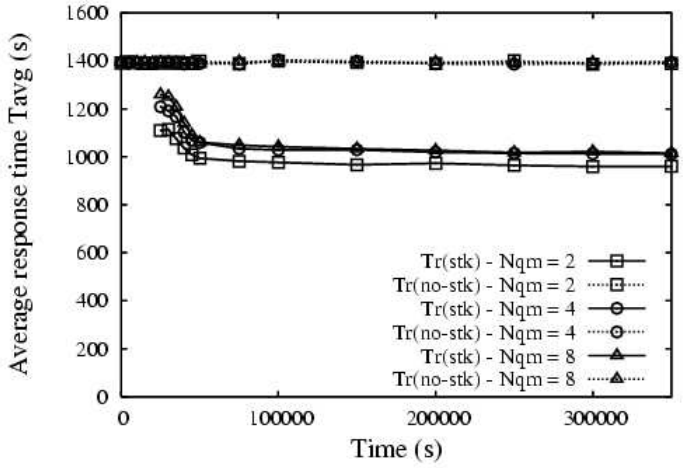


Fig. 6.8. Average response time achieved as the mapping process proceeds. Performance values, calculated for different values of Nqm and TTL set to 7, are reported separately for striking and non striking queries.

Average response time achieved as the mapping process proceeds. Performance values, calculated for different values of TTL and `Nqm` set to 4, are reported for all the queries and separately for striking and non - striking queries. As expected, it increases with the TTL value and with the number of query messages `Nqm`. Since the increase of these two parameters also allows for achieving a larger number of results, it is necessary to reach a trade-off that takes into account the expected number of results and the processing load that a Grid node can undergo. However, a very interesting consideration is that the logical reorganization of resources, and the use of the ARDIP protocol, allows for decreasing the query traffic load experienced by a single peer. Indeed, when

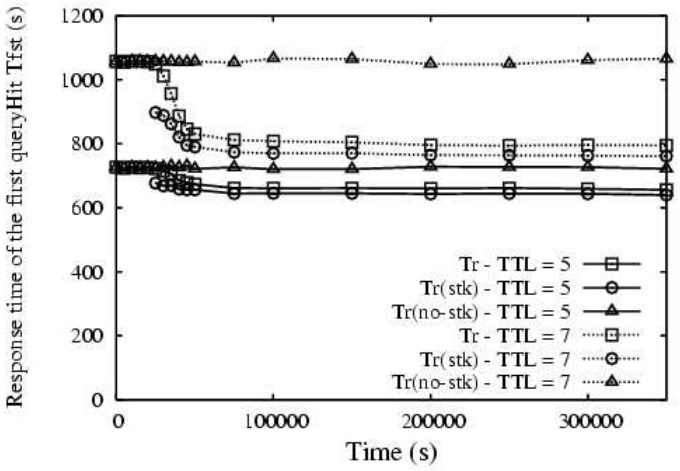


Fig. 6.9. Response time of the first queryHit achieved as the mapping process proceeds. Performance values, calculated for different values of TTL and Nqm set to 4, are reported for all the queries, and separately for striking and non striking queries.

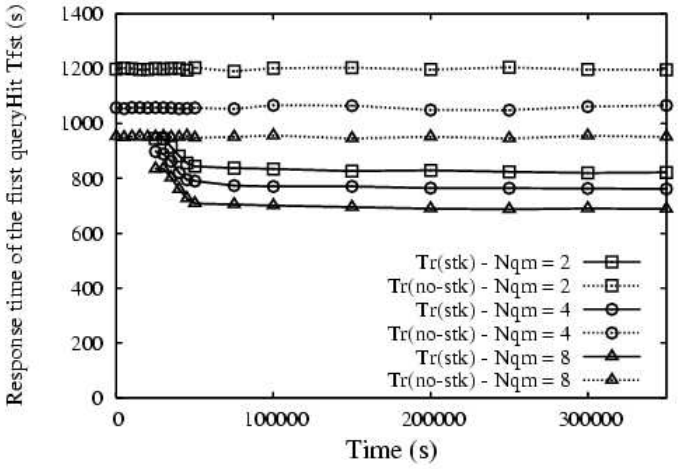


Fig. 6.10. Response time of the first queryHit achieved as the mapping process proceeds. Performance values, calculated for different values of Nqm and TTL set to 7, are reported separately for striking and non striking queries.

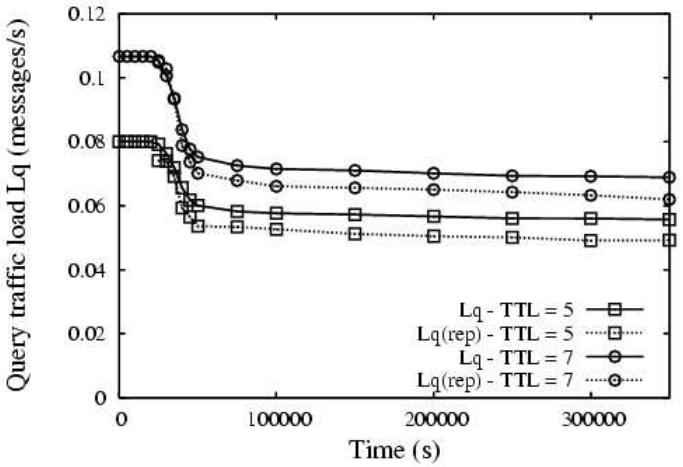


Fig. 6.11. Query traffic load experienced by generic peers and representative peers as the mapping process proceeds. Performance values are calculated for different values of TTL and Nqm set to 4.

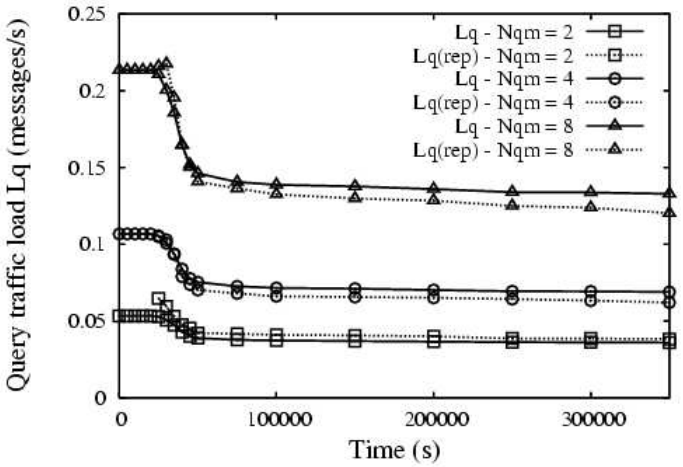


Fig. 6.12. Query traffic load experienced by generic peers and representative peers as the mapping process proceeds. Performance values are calculated for different values of Nqm and TTL set to 7.

a query messages is successfully driven towards a representative peers, on average it has to perform a lower number of hops than that experienced with a

blind search. A further benefit coming from the use of ARMAP/ARDIP protocols is that representative peers experience an even lower query load than a generic peer. In fact, when a query is issued by a peer which is adjacent to a representative one, the route to the representative peer is most likely known in advance, thanks to the pheromone mechanism described in section 6.4, and the blind search phase can be entirely skipped. In such a case, only one query message is generated by the requesting peer instead of `Nqm` (in fact all the `Nqm` queries would follow the same informed path that leads to the adjacent representative peer and would collect the same results), thus reducing the query traffic load at the nearby representative peer.

7

Conclusions & Future Works

Distributed systems have been a very active research field for a number of decades. Many current systems and infrastructure, such as the Grid Computing, Peer-to-Peer systems, and ad hoc wireless and sensor networks have the characteristic of being decentralized, pervasive, and composed of a large number of autonomous entities. Often systems deployed on such infrastructure need to run in highly dynamic environments, where content, network topologies and work loads are continuously changing. Adaptation thus becomes a key feature of a system's behavior. In addition, such systems involve a social dimension, for example, the entities within such systems can engage in interactions, discover suitable other participants, negotiate, and perform transactions. In certain cases, the complexity of the system is such that no centralized or hierarchical control is possible. In other cases, it is the unforeseeable context, in which the system evolves or moves, which makes any direct supervision difficult. These characteristics are similar to those which one finds in self-organizing systems we see in nature, such as physical, biological and social systems. Indeed, natural self-organizing systems have the characteristic to function without central control, and through contextual local interactions. Each component within such a system carries out a simple task, but as a whole such systems are able to carry out much more complex tasks. Such behavior emerges in a coherent way through the local interactions of the various components. These systems are particularly robust, because they adapt to the environmental changes, and are able to ensure their own maintenance or repair. There is currently an increasing appreciation that modern applications and systems can gain (in robustness, and simplicity) if they are developed by following the principles of self-organization which one finds in nature. In first section of this thesis, I describe P-SPARROW, a algorithm for distributed clustering of data in peer-to-peer environments combining a smart exploratory strategy based on a flock of birds with a density-based strategy to discover clusters of arbitrary shape and size in spatial data. The algorithm has been implemented in a peer-to-peer system and evaluated using a synthetic and a real word dataset. Experimental results show that P-SPARROW can be

efficiently applied as a data reduction strategy to perform approximate clustering. Moreover, the algorithm scales well when the number of peer increases. Finally, the use of a small world topology helps the algorithm to merge clusters more quickly with a slightly higher number of messages exchanged and permits a better fault tolerance because of high clustering coefficient characteristic of this topology. Furthermore, we have described an adaptive flocking algorithm and its application to the problem of clustering spatial data. The approach is based on the use of swarm intelligence techniques. Two novel algorithms that combine density-based and shared nearest neighbor clustering strategy with a flocking algorithm have been presented. The algorithms have been implemented in SWARM and evaluated using synthetic and real word datasets. Measures of accuracy of the results show that the flocking algorithm can be efficiently applied as a data reduction strategy to perform approximate clustering. Moreover, the algorithm presents a good scalable behavior. Finally, an entropy-based model has been applied to study the behavior of the algorithm. It has confirmed that the adaptive search strategy of our flocking algorithm is more efficient than the random-walk search strategy and the standard flocking algorithm and this can be explained because, in the interesting zones of the clusters, not only there is more organization but also a larger presence of searching agents. Last section, introduces an approach based on multi agent systems for building an efficient information system in Grids. A number of self-organizing agents travel the network by exploiting P2P interconnections; agents replicate and gather information related to resources having similar characteristics in restricted regions of the Grid. Such a logical reorganization of resources is exploited by a semi-informed resource discovery protocol, namely the ARDIP protocol, which is tailored to route a query message towards a "representative peer" that collects a large number of resources having the desired characteristics. Simulation analysis showed that, as the reorganization of resources proceeds, the ARDIP protocol allows users to discover more and more resources in a shorter amount of time, and at the same time to decrease the traffic load experienced by Grid hosts. It must be remarked that performance results are related to a particular choice of parameter values: for example resources are categorized in 5 different classes and most results are computed for a Grid of 2,500 hosts. However, performance evaluation w.r.t several parameters, e.g. Grid size, number of agents, and TTL, suggests that an imperfect choice of parameter values cannot spoil the reorganization and discovery process, but can only make such process faster or slower, and final achievements (for example the decrease in overall entropy and the increase in the number of results) seem to be preserved in any case. Furthermore the self-organizing and decentralized nature of the involved algorithms, along with the analysis of performance results obtained with variable Grid sizes, suggest that the proposed approach is scalable and can be adopted in a Grid framework regardless of the Grid size. Current work focuses on the implementation of ARDIP for the discovery of WSRF-compliant Web services. Web services can be categorized according to their syntactic and semantic fea-

tures (e.g. WSDL specification of input and output parameters, pre and post conditions, ontology concepts) and QoS information (e.g. information about service availability, reliability, execution time, price).

References

1. The jxta project. http://www.jxta.org.
2. Adler, M., Davis, A., Weihmayer, R., and Worrest, R. *Conflict Resolution Strategies for Nonhierarchical Distributed Agents*, vol. 2. 1989.
3. Andrzejak, A., and Xu, Z. Scalable, efficient range queries for grid information services. In *Proceedings of the Second IEEE International Conference on Peer-to-Peer Computing* (Linkping University, Sweden, 2002).
4. Babaoglu, O., Canright, G., Deutsch, A., Caro, G. A. D., Ducatelle, F., Gambardella, L. M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., and Urnes, T. Design patterns from biology for distributed computing. *ACM Transactions on Autonomous and Adaptive Systems 1* (2006), 26–66.
5. Babaoglu, O., Meling, H., and Montresor, A. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Proceedings of the 22th International Conference on Distributed Computing Systems ICDCS '02* (Vienna, Austria, 2002).
6. Barbuceanu, M., and Fox, M. Cool: A language for describing coordination in multi-agent systems. In *Proc. First Intl Conf. Multi-Agent Systems* (Menlo Park, Calif., USA, 1995), AAAI Press, pp. 17–24.
7. Beni, G., and Wang, J. Swarm intelligence. In *In Proc. of the Seventh Annual Meeting of the Robotics Society of Japan* (Tokyo, Japan, 1989), RSJ Press, pp. 425–428.
8. Bonabeau, E., Dorigo, M., and Theraulaz, G. Swarm intelligence: From natural to artificial systems. *J. Artificial Societies and Social Simulation 4*, 1 (2001).
9. Bond, A., and Gasser, L. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, 1988.
10. Brooks, R. A. Intelligence without representation. *Artificial Intelligence 47*, 1-3 (1991), 36–46.
11. Camazine, . S., Deneubourg, J. L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. Self-organization in biological systems. Princeton University Press.
12. Cammarata, S., McArthur, D., and Steeb, R. Strategies of cooperation in distributed problem solving. In *In Proceedings of the Eighth International*

Joint Conference on Artificial Intelligence (Menlo Park, Calif., USA, 1983), pp. 767–770.

13. Carver, N., and Lesser, V. A new framework for sensor interpretation: Planning to resolve sources of uncertainty. In *Proc. Natl Conf. Artificial Intelligence* (1991), pp. 724–731.
14. Cohen, P. R., and Levesque, H. J. *Intention Is Choice with Commitment*, vol. 42. 1990.
15. Conry, S. E., Meyer, R. A., and Lesser, V. R. *Multistage Negotiation in Distributed Planning*. Morgan Kaufmann, San Francisco, Calif., USA, 1988.
16. Corkill, D. D., and Lesser, V. R. Use of metalevel control for coordination in a distributed problem solving network. In *In Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (Menlo Park, Calif., USA, 1983).
17. Crespo, A., and Garcia-Molina, H. Routing indices for peer-to-peer system. In *ICDCS '02: Proceedings of the 22th International Conference on Distributed Computing System* (Vienna, Austria, 2002), Springer-Verlag, pp. 23–33.
18. Dasgupta, P. Intelligent agent enabled p2p search using ant algorithms. In *Proceedings of the 8th International Conferences on Artificial Intelligence* (Las Vegas, NV, 2004), pp. 751–757.
19. Datta, S., Bhaduri, K., Giannella, C., Wolff, R., and Kargupta, H. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing (to appear)*.
20. Davis, R., and Smith, R. G. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence 20*, 1 (1983), 63–100.
21. Decker, K., and Lesser, V. Designing a family of coordination algorithms. In *In ProProceedceedings of the First International Conference on Multiagent Systems* (Menlo Park, Calif., 1985), AAAI Press, pp. 73–80.
22. Decker, K., Pannu, A., Sycara, K., and Williamson, M. Designing behaviors for information agents. In *In Proceedings of the First International Conference on Autonomous Agents* (New York, 1997), Association of Computing Machinery, pp. 404–412.
23. Decker, K., Sycara, K., and Williamson, M. Matchmaking and brokering. In *Proceedings of the Second International Conference on Multiagent Systems* (Menlo Park, Calif., 1996), AAAI Press, p. 432.
24. Deneubourg, J. L., Goss, S., Franks, N., Sendova-Franks, Detrain, A. C., and Chretien, L. The dynamics of collective sorting robot-like ants and ant-like robots. In *From Animals to Animats: Proc. of the 1st Int. Conf. on Simulation of Adaptive Behaviour* (1990), MIT Press/Bradford Books.
25. Dent, L., Boticario, J., McDermott, J., D., T. M., and Zabowski. A personal learning apprentice. In *In Proceedings of the Tenth National Conference on Artificial Intelligence* (Menlo Park, Calif., USA, 1992), AAAI Press, pp. 96–103.
26. Drogul, A., and Ferber, J. From tom thumb to the dockers: Some experiments with foraging robots. In *In From Animals to Animats: Second Conference on Simulation of Adaptive Behavior* (Cambridge, Mass., 1992), MIT Press.
27. Durfee, E. H. *A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network*. Department of Computer and Information Science, University of Massachusetts, 1987.

28. Durfee, E. H. *Coordination of Distributed Problem Solvers*. Kluwer Academic, Boston, 1988.
29. Durfee, E. H., and Lesser, V. Negotiating task decomposition and allocation using partial global planning. *Distributed Artificial Intelligence 2*.
30. Durfee, E. H., Lesser, V. R., and Corkill, D. D. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers C-36* (1987), 1275–1291.
31. Ertoz, L., M.Steinbach, and Kumar, V. A new shared nearest neighbor clustering algorithm and its applications. In *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining* (2002).
32. Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining* (1996), pp. 226–231.
33. Ferber, J. *Reactive Distributed Artificial Intelligence: Principles and Applications*. Wiley, New York, 1996.
34. Finin, T., Fritzson, R., McKay, D., and McEntire, R. Kqml as an agent communication language. In *In Proceedings of the Third International Conference on Information and Knowledge Management* (New York, 1994), Association of Computing Machinery, pp. 456–463.
35. Folino, G., and Spezzano, G. An adaptive flocking algorithm for spatial clustering. In *PPSN* (2002), pp. 924–933.
36. Forestiero, A., Mastroianni, C., and Spezzano, G. A multi agent approach for the construction of a peer-to-peer information system in grids. In *Proc. of 2005 International Conference on Self-Organization and Adaptation of Multi-agent and Grid Systems* (, Glasgow, UK, 2005).
37. Garrido, L., and Sycara, K. Multiagent meeting scheduling: Preliminary experimental results. In *In Proceedings of the Second International Conference on Multiagent Systems* (Menlo Park, Calif., USA, 1996), AAAI Press, pp. 95–102.
38. Gasser, L. *Social Conceptions of Knowledge and Action*, vol. 47. 1991.
39. Genesereth, M. R., and Ketchpel, S. P. Software agents. *Communications of the ACM 37(7)* (1994), 48–53.
40. Grass, P. *La Reconstruction du nid et les Coordinations Inter-Individuelles chez Beellicositermes Natalensis et Cubitermes sp. La Thorie de la Stigmergie : Essai d'interprtation du Comportement des Termites Constructeurs in Insect. Soc. 6*. Morgan Kaufmann, 1959.
41. Grosz, B., and Kraus, S. *Collaborative Plans for Complex Group Actions*, vol. 82. 1996.
42. Grosz, B., and Sinder, C. Plans for discourse. *In Intentions in Communication* (1990), 417–444.
43. Guha, S., Rastogi, R., and Shim, K. CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data* (June 1998), pp. 73–84.
44. Han, J., and Kamber, M. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, September 2000.
45. Han, J., Kamber, M., and a. K. H. Tung. *Spatial Clustering Methods in Data Mining: A Survey, Geographic Data Mining and Knowledge Discovery*. Morgan Kaufmann, 2001.

46. Handl, J., and Meyer, B. Improved ant-based clustering and sorting. In *PPSN VII: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature* (London, UK, 2002), Springer-Verlag, pp. 913–923.
47. Hans-Peter, E. J. Towards effective and efficient distributed clustering, 2003.
48. Hewitt, C. *Offices Are Open Systems.* ACM Transactions of Office Automation Systems 4(3), 1986.
49. Hu, J., and Wellman, M. P. Self-fulfilling bias in multiagent learning. In *Proceedings of the Second International Conference on Multiagent Systems* (Menlo Park, Calif., 1996), AAAI Press, pp. 118–125.
50. Huberman, B., and Clearwater, S. A multi-agent system for controlling building environments. In *Proc. First Intl Conf. MultiAgent Systems* (Menlo Park, Calif., 1995), AAAI Press, pp. 171–176.
51. Huberman, B. A., and Hogg, T. Behavior of computational ecologies. *The Ecology of Computation.*
52. Iamnitchi, A., and Foster, I. Interest-aware information dissemination in small-world communities. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing* (Research Triangle Park, NC, 2005).
53. Iamnitchi, A., Foster, I., Weglarz, J., Nabrzyski, J., Schopf, J., and Stroinski, M. .: A peer-to-peer approach to resource location in grid environments. *Grid Resource Management* (2003).
54. Jarvis, R. A., and Patrick, E. A. Clustering using a similarity measure based on shared nearest neighbors. *IEEE Transactions on Computers C-22*, 11 (1973).
55. Jennings, N. *Coordination Techniques for Distributed Artificial Intelligence.* Foundations of Distributed Artificial Intelligence, 1996.
56. Jennings, N., Sycara, K., and Wooldridge, M. *A Roadmap for Agent Research and Development.* Autonomous Agents and Multiagent Systems 1(1), 1998.
57. Jennings, N. R., Corera, J. M., and Laresgoiti, I. Developing industrial multiagent systems. In *In Proceedings of the First International Conference on Multiagent Systems* (Menlo Park, Calif., USA, 1995), AAAI Press, pp. 423–430.
58. Jha, S., Chalasani, P., Shehory, O., and Sycara, K. A formal treatment of distributed matchmaking. In *In Proceedings of the Second International Conference on Autonomous Agents* (New York, 1998), Association of Computing Machinery.
59. Kanade, P., and Hall, L. Fuzzy ant clustering by centroid positioning. In *IEEE International Conference on Fuzzy Systems* (Budapest, 2004), IEEE Computer Society, pp. 371–376.
60. Karsai, I. Decentralized control of construction behavior in paper wasps: An overview of the stigmergy approach. In *Artificial Life 5* (1999), pp. 97–116.
61. Karypis, G., Han, E.-H., and Kumar, V. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer 32*, 8 (1999), 68–75.
62. Kaufman, L., and Rousseeuw, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley, 1990.
63. Kazemian, M., Ramezani, Y., Lucas, C., and Moshiri, B. Distributed data clustering based on flowers pollination by artificial bees. In *WSTST 2005 - 4th IEEE International Conference on Soft Computing as Transdisciplinary Science and Technology* (Muroran, Japan, May 2005), Springer-Verlag Engineering Series.

64. Kennedy, J., and Eberhart, R. C. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks* (Perth, Australia, 1995), IEEE Service Center.
65. Kephart, J. O., Hogg, T., and Huberman, B. A. Dynamics of computational ecosystems: Implications for dai. *Artificial Intelligence 2* (1989), 79–96.
66. Kinny, D., Ljungberg, M., Rao, A., Sonenberg, E., Tidhard, G., and Werner, E. *Planned Team Activity.* Springer-Verlag, 1992.
67. Kollios, G., Gunopulos, D., Koudas, N., and Berchtold, S. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions on Knowledge and Data Engineering 15*, 5 (2003), 1170–1187.
68. Kornfeld, W. A., and Hewitt, C. E. The scientific community metaphor. *IEEE Transactions on Systems, Man, and Cybernetics 11*, 1 (1981), 24–33.
69. Kronfol, A. *FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine.* PhD thesis at Princeton Univerity, 2002.
70. Kuntz, P., and Snyers, D. Emergent colonization and graph partitioning. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3* (Cambridge, MA, USA, 1994), MIT Press, pp. 494–500.
71. Kuntz, P., and Snyers, D. Emergent colonization and graph partitioning. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3* (Cambridge, MA, USA, 1994), MIT Press, pp. 494–500.
72. Lander, S., and Lesser, V. Sharing meta-information to guide cooperative search among heterogeneous reusable agents. *IEEE Trans. Knowledge and Data Eng. 9*, 2 (1997), 193–208.
73. Lander, S., Lesser, V. R., and Connell, M. E. Conflict-resolution strategies for cooperating expert agents. In *In Proceedings of the International Working Conference on Cooperating Knowledge-Based Systems* (New York, 1991), Springer-Verlag, pp. 183–200.
74. Lesser, V. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering 11*, 1 (1999).
75. Lesser, V. R. A retrospective view of fa/c distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics 21*, 6 (1991), 1347–1363.
76. Lesser, V. R., Durfee, E. H., and Corkill, D. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering 1*, 1 (1989), 63–83.
77. Lewis, C. M., and Sycara, K. *Reaching Informed Agreement in Multispecialist Cooperation*, vol. 2. 1993.
78. Ljunberg, M., and Lucas, A. The oasis air-traffic management system. In *In Proceedings of the Second Pacific Rim International Conference on AI* (Seoul, Korea, 1992).
79. Lumer, E. D., and Faieta, B. Diversity and adaptation in populations of clustering ants. In *From Animals to Animats: Proc. of the Third Int. Conf. on Simulation of Adaptive Behaviour* (1994), MIT Press.
80. Lv, C., Cao, P., Cohen, E., Li, K., and Shenker, S. Search and replication in unstructured peer-to-peer networks. *ACM*.
81. Macgill, J. Using flocks to drive a geographical analysis engine. In *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial*

Life (2000), M. A. B. J. S. M. N. H. P. S. Rasmussen, Ed., The MIT Press, Cambridge, Massachusetts.
82. Maes, P. *Designing Autonomous Agents*. MIT Press, Cambridge, Mass., 1990.
83. Mason, C., and Johnson, R. *DATMS: A Framework for Distributed Assumption-Based Reasoning*, vol. 2. Morgan Kaufmann, San Francisco, Calif., USA, 1992.
84. Mastroianni, C., Talia, D., and Verta, O. A super-peer model for resource discovery services in large scale grids. *Future Generation Computer System 21*, 8 (2005), 1235–1456.
85. Michener, C. D. The social behavior of the bees: a comparative study. Harvard University Press.
86. Minar, N., Burkhart, R., Langton, C., and Askenazi, M. The swarm simulation system, a toolkit for building multi-agent simulations, 1996.
87. Minsky, M. *The Society of Mind*. Simon and Schuster, New York, 1986.
88. Mllen, T., and Wellman, M. P. *Some Issues in the Design of Market-Oriented Agents*. Springer-Verlag, New York, 1996.
89. Monmarché, N., Slimane, M., and Venturini, G. On improving clustering in numerical databases with artificial ants. In *ECAL '99: Proceedings of the 5th European Conference on Advances in Artificial Life* (London, UK, 1999), Springer-Verlag, pp. 626–635.
90. Neiman, D., Hildum, D., Lesser, V., and Sandholm, T. Exploiting meta-level information in a distributed scheduling system. In *Proc. of 12th Natl Conf. Artificial Intelligence* (1994).
91. Newell, A. *Unified Theories of Cognition*. 1990.
92. Oates, T., Prasad, M. N., and Lesser, V. *Cooperative Information Gathering: A Distributed Problem Solving Approach*, vol. 144. 1997.
93. O'Hare, G., and Jennings, N. *Foundations of Distributed Artificial Intelligence*. John Wiley, New York, 1996.
94. Openshaw, S., and Macgill, J. The use of flocks to drive a geographical analysis machine. In *GeoComputation 98, Proceedings of the 3rd International Conference on GeoComputation* (Bristol, United Kingdom, September 1998).
95. Parunak, H. V. D., and Brueckner, S. Entropy and self-organization in multi-agent systems. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents* (New York, NY, USA, 2001), ACM Press, pp. 124–130.
96. Parunak, H. V. D., and Brueckner, S. Engineering swarming systems. In *Methodologies and Software Engineering for Agent Systems* (2004), Kluwer, pp. 341–376.
97. Parunak, H. V. D., Bureckner, S. A., Matthews, R., and Sauter, J. Pheromone learning for self-organizing agents. *IEEE Transaction on System 35*, 3 (2005).
98. Parunak, V. *Manufacturing Experience with the Contract Net*, vol. 1. London, 1987.
99. Ramos, V., and Abraham, A. Swarms on continuous data. In *CEC03 - Congress on Evolutionary Computation* (Canberra, Australia, December 2003), IEEE Computer Society.
100. Ramos, V., and Merelo, J. J. Self-organized stigmergic document maps: Environment as mechanism for context learning. In *AEB 2002 - 1st Spanish Conference on Evolutionary and Bio-Inspired Algorithms* (Mrida, Spain, September 2002).

101. Ramos, V., Muge, F., and Pina, P. Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies. In *Soft Computing Systems - Design, Management and Applications* (2002), A. Abraham, J. Ruiz-del-Solar, and M. Köppen, Eds., Frontiers in Artificial Intelligence and Applications Vol. 87, IOS Press Amsterdam, Berlin, Oxford, Tokyo, Washington D.C., pp. 500–512.
102. Rao, A. S., and George, M. P. *Toward a Formal Theory of Deliberation and Its Role in the Formation of Intentions*. Victoria, Australia, 1991.
103. Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 25–34.
104. Rick, C., and Sidner, C. Collagen : When agents collaborate with people. In *In Proceedings of the First International Conference on Autonomous Agents* (Association of Computing Machinery, New York, 1997), pp. 284–291.
105. Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov. 2*, 2 (1998), 169–194.
106. Sandholm, T. An implementation of the contract net protocol based on marginal cost calculations. In *In Proceedings of the Eleventh National Conference on Artificial Intelligence* (Menlo Park,Calif., 1993), American Association for Artificial Intelligence, pp. 256–262.
107. Sandholm, T., and Lesser, V. Issues in automated negotiation and electronic commerce: Extending the contract net protocol. In *In Proceedings of the Second International Conference on Multiagent Systems* (Menlo Park,Calif., 1995), AAAI Press, pp. 328–335.
108. Seeley, T., Camazine, S., and Sneyd, J. Collective decision making in honey bees: how colonies choose among nectar sources. In *Behavioral Ecology and Sociobiology* (1991), pp. 277–290.
109. Shoham, Y. *Agent-Oriented Programming*, vol. 60. 1993.
110. Simon, H. *Models of Man: Social and RationalMathematical Essays on Rational Human Behavior in a Social Setting*. John Wiley, New York, 1957.
111. Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM* (San Diego, CA, USA, 2001).
112. Sugawara, T., and Murakami, K. A multi-agent diagnostic system for internetwork problems. In *ProceedingsINET* (1992).
113. Sycara, K. Using option pricing to value commitment flexibility in multiagent systems. *Technical report, CMU-CSTR* (1997), 97–169.
114. Sycara, K., Decker, K., Pannu, A., Williamson, M., and Zeng, D. Distributed intelligent agents. *IEEE Expert 11*, 6 (1996), 36–46.
115. Tambe, M. *Toward Flexible Teamwork*, vol. 7. 1997.
116. Thomas, J., and Sycara, K. Stability and heterogeneity in multiagent systems. In *In Proceedings of the Third International Conference on Multiagent Systems* (Menlo Park, Calif., USA, 1998), AAAI Press.
117. Travers, M. Animal construction kit. *Artificial Life*.
118. Tsoumakos, D., and Roussopoulos, N. Adaptive probabilistic search for peer-to-peer networks. In *In: Third International Conference on Peer-to-Peer Computing, P2P '03* (Linkoping, Sweden, 2003), pp. 102–110.

119. Tsoumakos, D., and Roussopoulos, N. Comparison of peer-to-peer search methods. In *Proc. of the Sixth International Workshop on the Web and Databases* (San Diego, CA, USA, 2003), pp. 61–66.
120. Van der Merwe, D. W., and Engelbrecht, A. P. Data clustering using particle swarm optimization. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003* (Canberra, 8-12 December 2003), R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, Eds., IEEE Press, pp. 215–220.
121. Veloso, M., Stone, P., Han, K., and Achim, S. Cmunited: A team of robotic soccer agents collaborating in an adversarial environment. In *In Proceedings of the FirstInternational Workshop on ROBOCUP* (San Francisco, Calif., 1997), Morgan Kaufmann.
122. Vizine, A. L., de Castro, L. N., Hruschka, E. R., and Gudwin, R. R. Towards improving clustering ants: An adaptive ant clustering algorithm. *Informatica (Slovenia) 29*, 2 (2005), 143–154.
123. Wang, W., Yang, J., and Muntz, R. R. Sting: A statistical information grid approach to spatial data mining. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece* (1997), Morgan Kaufmann, pp. 186–195.
124. Wang, X., and Hamilton, H. J. Dbrs: A density-based spatial clustering method with random sampling. In *PAKDD* (2003), pp. 563–575.
125. Watts, D. J., and Strogatz, S. H. Collective dynamics of 'small-world' networks. *Nature 393*, 6684 (June 1998), 440–442.
126. Wooldridge, M., and Jennings, N. *Intelligent Agents: Theory and Practice*, vol. 10. 1995.
127. Yang, B., and Garcia-Molina, H. Efficient search in peer-to-peer networks. In *, Proc. of ICDCS* (Wien, Austria, 2002).
128. Yang, B., and Garcia-Molina, H. Designing a super-peer network. In *19th Int'l Conf. on Data Engineering* (Los Alamitos, CA, USA, 2003), IEEE Computer Society Press.
129. Zeng, D., and Sycara, K. Benefits of learning in negotiation. In *In Proceedings of the National Conference on Artificial Intelligence* (Menlo Park, Calif., USA, 1997), AAAI Press, pp. 36–41.