Universidad Autónoma de Madrid
Escuela Politécnica Superior

Work to obtain the Master's Degree in
Ingeniería Informática y Telecomunicaciones.
by Universidad Autónoma de Madrid

Master thesis supervisor:

Eloy Anguiano

A Complexity Science Approach to Swarm Intelligence

Juan I. Cano Núñez

This work was presented at date September 9, 2010

Evaluators:

Eloy Anguiano Rey (chairman)

David Camacho Fernández

Francisco de Borja Rodríguez Ortiz

© 2010 by UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, nº 1
Madrid, 28049
Spain

Juan I. Cano Núñez
A Complexity Science Approach to Swarm Intelligence

Juan I. Cano Núñez
inaki.cano@gmail.es

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres.

To those who supported me,
family, friends and colleagues.
They gave me everything

We know that there is no help for us but from one another,
that no hand will save us if we do not reach out our hand.
And the hand that you reach out is empty, as mine is.
You have nothing. You possess nothing. You own nothing.
You are free. All you have is what you are, and what you give.
– Shevek (The Dispossessed)

Final Master's Thesis evaluators:

Eloy Anguiano Rey
(Chairman)

David Camacho Fernández

Francisco de Borja Rodríguez Ortiz

Contents

Acknowlegments

As with every long work, although this wasn't so long, a lot of people is involved in its development. I think it is impossible to mention everyone that one way or another made this possible, sometimes small gestures get big results. Before I forget someone, I would like to thank everyone, friends and family, for their patience and support.

The first acknowledgement is for my parents, Juan e Hilda, as they made me what I am and supported me in every step I took.

In the University I made good friends and colleagues, from them I would like to thank my "big sister" Carmen Navarrete who never really understood what I was doing but kept reading every article I wrote and gave me very useful advise about them and about the life of a researcher. I can't but to thank Luis Sánchez, German Retamosa, Pablo Llopis, Paul Goldbaum, Luis Santos and other friends for their support, sometimes by just playing some videogames, other times just listening for hours why complex systems is the way to go.

As in every thesis, there's a director involved. I would like to thank Eloy Anguiano for his effort, critique and guidance, dealing with a stubborn, idealist and rebelious student is not an easy task. Also important for this dissertation were Francisco de Borja Rodríguez, who apart from helping me with the articles helped me to keep my morale up, David Camacho, who helped me with my first article and taught me about agents and CSP, and Carlos Aguirre, who gave me my first article to read and originated my interest for complex systems.

Finally, I would like to thank Atienza Saldaña, a soon to be anthropologist and future researcher, for being always by my side, helping, caring and reminding me that "this too shall pass". Let her become a greater researcher than I will ever be.

ABSTRACT

This dissertation presents the work done in the past year towards the creation of a methodology or framework for the design of multiagent systems. The main goal is to use complexity science to shed some light in the design process of multiagent systems and swarms. The results presented here are from the initial phase of the project, the methodology is yet to be developed. In chapter 1, the four fields that this work cover are presented: Multiagent Systems, Swarm Intelligence, Constraint Satisfaction Problems and Complex Systems. After that, in chapter 2, a first approach to multiagent systems is presented with a system built to solve a university scheduling problem. As one of the goals is to build very light systems, chapter 3 presents a first step in the simplification using a swarm of agents based on memory and message discrimanation to solve a CSP benchmark problem, the N–Queen problem. Chapter 4 proposes a possible analysis using a complexity science approach to both systems. Finally, chapter 5 concludes and suggests future directions for the research.

Resumen

Este documento presenta el trabajo realizado el pasado año en la creación de una metodología o marco de trabajo para el diseño de sistemas multiagente. El objetivo principal es usar la ciencia de la complejidad para arrojar algo de luz sobre el proceso de diseño de sistemas multiagente y enjambres de agentes. Los resultados presentados aquí proceden de la etapa inicial del proyecto, todavía se ha de desarrollar una metodología. En el capítulo 1 se presentan los cuatro campos que abarca este trabajo: Sistemas Multiagente, Inteligencia Enjambre, Problemas de Satisfacción de Restricciones y Sistemas Complejos. Tras esto, en el capítulo 2, se presenta una primera aproximación a los sistemas multiagente con un sistema construido para resolver un problema de planificación en la universidad. Cómo una de las metas del trabajo es construir agentes muy ligeros, el capítulo 3 presenta el primer paso hacia esta simplificación usando un enjambre de agentes basados en memoria y discriminación de mensajes para resolver un problema de satisfacción de restricciones usado de referencia, el problema de las N Reinas. El capítulo 4 propone un posible análisis usando una aproximación de sistemas complejos en los dos sistemas. Finalmente, el capítulo 5 concluye y sugiere futuras direcciones para la investigación

1 Introduction

The main objective of this dissertation is to establish the foundations for a future doctoral thesis on the use of complexity science in the design and evaluation of multiagent systems. This thesis is still on an early state and the current dissertation only presents the key ideas to be developed for the doctoral dissertation and a swarm system that is based on these ideas. Before introducing the system, its characteristics and some initial results, we will describe some fundamental ideas and a previous work on negotiating agents to understand what is being done and why.

1.1 Multiagent Systems

Since its inception in the early ages of computers, artificial intelligence (AI) has been a quest for the design in software of an intelligent entity. As this goal seemed very ambitious and some walls were encountered along the way, AI has been redefining itself from the goal of creating a single entity that had all the traits we expect from an intelligent being [1] to try to recreate one of those traits in isolation [2, 3], from trying to imitate human intelligence to improve results of problem solving software with "intelligence" [4].

Nowadays, AI researchers are working on multiagent systems (MAS). The definition of agent is a very discussed topic as every researcher in the field has his own definition [5–7], but for this paper we will understand an agent as an autonomous entity that has some capabilities for interaction (usually expressed as inputs and outputs), some intentions or goals and the ability to move towards that goal. Multiagent systems, then, are defined as systems comprised of multiple interacting agents. This definitions of agent and multiagent system are very relaxed, a precise definition is impossible in practice. Also, both definitions can refer to software or natural and social systems. A market, an ant colony and an ecosystem are examples of multiagent systems. This is the reason why many models of natural and social systems are built using this approximation, reducing its components to simple, well-defined agent representations.

When referring to software agents or systems, we should differentiate between Agent Oriented Programming (AOP) and Agent Based Models (ABM) [8], depending on what kind of software system approach we are using. AOP is considered the next paradigm after Object Oriented Programming (OOP), where the key elements of the systems are considered agents. This approach has more to do with Software Engineering than with AI as it determines some guidelines when designing a software system with multiple interacting elements and it's not focused on the capabilities (i.e. reasoning, learning, etc) of the agents. ABM is also a methodology, but one oriented to the design of models and simulations of systems. Each agent represents an element of the system studied, for example in a social system each agent would be a person or in an ecosystem each agent would represent some species, and each agent is programmed with some behavior or a set of capabilities. Once the agents are modeled, the simulation is run with a group of them and this way the experimenter can extract some conclusions about the system. The artificial intelligence approach to multiagent systems stands

in a middle ground. AI researchers focus on the design of agents that can perform adequatly in a multiagent environment [7] or in the design of a whole system with some desired properties based on the cooperation and/or competition between agents [5].

A multiagent system can, for example, be applied to problem solving. In this case, the role of the researcher is the design of a MAS where the agents give a solution to a problem. This approach was a response to the increasing availability of clusters and grids and to the observation that natural and social systems also uses a distributed approach to problem solving. Some examples of distributed problem solving in the natural and social world can be seen in markets, where different sellers adjust prices and make pacts to increase the benefits or cut loses [9], or in the ants foraging and defense behavior [10].

When designing agents, we can recognize two types named *heavy agents* and *light agents*. Heavy agents are agents that have some abilities as reasoning, learning, negotiation or complicated communication mechanisms. The use of heavy agents can be understood as motivated by the idea that cooperation or competition are achieved through a reasoning process. This branch of multiagent systems heavily relies on ideas from game theory, mathematical logic and psychology [7]. On the other hand, light agents are simple reactive agents with none of the previous mentioned abilites and with a simple communication mechanism if any. This kind of agents can be seen in natural systems, neurons, ants and cells are examples, and this is the inspiration behind the use of this agents.

In the next chapters we will discuss some systems built following both approaches, heavy (although not so heavy) and light agents, but before introducing them there are three concepts that need to be explained: Constraint Satisfaction Problems, Swarm Intelligence and Complex Systems.

1.2 Constraint Satisfaction Problems

A constraint satisfaction problem is a problem where we have a set of variables with different possible values and we need to assign to each variable a value that doesn't interfere with the other variable's values. This intereferences are given as a set of constraints and a solution to the problem is a set of assignments where each constraint is satisfied [11].

An easy to understand example of this kind of problem is the map coloring problem. In this problem we have map with different countries or regions and the goal is to color each region with a color different from the colors of its neighbors. Here the variables are the regions, the domains for each variable are the colors available and the constraints are for each variable to be different from the neighboring variables. We can represent this as a graph where each node is a region and each link represents a contraint (fig. 1.1).

In the figure, X_1, X_2 and X_3 are the variables and C_1, C_2 and C_3 are the constraints. For this to be a complete definition of a CSP we need to define a domain for each variable. The domains don't need to be the same, we can have problems where the domains are completely different and even problems where the variables don't represent the same thing. To illustrate this we will consider the domains to be

$$\begin{aligned} D(X_1) &= D(X_2) = \{Red, Green, Blue\} \\ D(X_3) &= \{Red, Green\} \end{aligned}$$

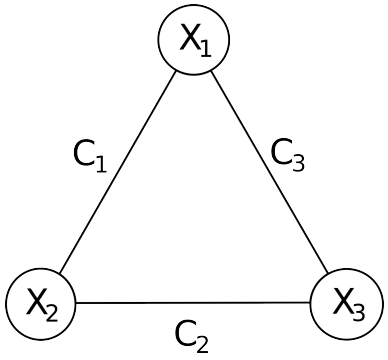

Figure 1.1: The map coloring problem with three regions represented as a graph. Each node represents a region and the links represents a constraint.

and the contraints

$$\begin{aligned} C_1 &= < X_1 \neq X_2 > \\ C_2 &= < X_2 \neq X_3 > \\ C_3 &= < X_1 \neq X_3 > \end{aligned}$$

With this we have completely defined a CSP, we have a set of variables X = {X_1, X_2, X_3}, a set of domains D = {$D(X_1)$, $D(X_2)$, $D(X_3)$} and a set of constraints C = {C_1, C_2, C_3}.

The first attempts to solve CSP used *Backtracking* algorithms: values are assigned to each variable until a variable has no available value, then we go back one step and try to assign a different value to the previous variable. At each step, the domains of the unassigned variables are updated to contain only values that can satisfy the constraints. Following the alphabetical order for the variables and the order afore written for the domains (Red then Green and then Blue), applying this algorithm to our toy problem problem we have the following trace:

(1) Assign Red to X_1 and update $D(X_2)$ and $D(X_3)$
$X_1 = Red\ ;\ D(X_2) = \{Green, Blue\},\ D(X_3) = \{Green\}$

(2) Assign Green to X_2 and update $D(X_3)$
$X_1 = Red,\ X_2 = Green\ ;\ D(X_3) = \{\}$

(3) Rollback, $D(X_3)$ is empty
$X_1 = Red\ ;\ D(X_2) = \{Green, Blue\},\ D(X_3) = \{Green\}$

(4) Assign Blue to X_2 and update $D(X_3)$
$X_1 = Red,\ X_2 = Blue\ ;\ D(X_3) = \{Green\}$

(5) Assign Green to X_3 and we have a solution
$X_1 = Red,\ X_2 = Blue,\ X_3 = Green$

This algorithms are very expensive computationally and they were improved using heuristics before better algorithms were developed [12]. For example, ordering the variables by the number of constraints they have or the size of their domains can improve the results. This family of algorithms are costful not only because the size of the search, but also because checking the constraints, sorting the variables (in the case of using

heuristics) and some other side operations can be very expensive computationally. Some improvements of the backtracking algorithm and heuristics are the backmarking algorithm [13], backjumping [14], the use of look-ahead [15] and constraint learning [16].

Using concepts of local consistency some algorithms have been developed that can solve the problem, simplify it or verify its satisfiability. This family of algorithms are commonly known as *Constraint Propagation* algorithms. Local consistency implies that constraints taken in a local context are satisfied. The most typical local consistency conditions are node consistency (the unary constraints are satisfied by all values in the domain), arc consistency (taking pairs of variables, their domains only contain values that satisfy a binary constraint between them) and path consistency (similar to arc consistency but considering three variables).

Constraint propagation techniques are commonly used to clean up the domains of a CSP before applying another algorithm to solve the problem. The implementation of these algorithms are usually based on cascading effects: after one domain is altered, all the related variables are also updated until no domain is changed. These techniques are very useful for large problems where the time taken in this domain clean up is less than the time taken to solve the system without any change, and sometimes this preparation step doesn't improve significantly the overall time to solve the problem. On the other hand, using a constraint propagation algorithm can simplify a lot the problem and even solve it or mark it as unsolvable.

Finally, local search can be used to solve this kind of problems, but they don't guarantee to find a solution even if there is one [17]. There are also distributed techniques used to solve this problems and they will be discussed in the next chapter when we discuss the negotiating agents' approach to a CSP.

1.3 Swarm Intelligence

Swarm Intelligence is the term used to describe systems where an optimal state is achieved through the interaction of a group of simple agents. The term originated in the late 1980s to refer to cellular robotics and was coined by Gerardo Beni and Jing Wang [18]. Although this was the initial use of the term, swarm intelligence has being used in the context of modeling and simulation of software and natural systems. Among the most famous examples of swarm intelligence we remark *Ant Colony Optimization* (ACO) [19], *Particle Swarm Optimization* (PSO) [20] and *Bees algorithm* [21] in the domain of optimization and some models of birds flocking [22] and schools of fish [23] in the domain of simulation. As we are interested in swarm intelligence for problem solving, we will focus on the first three examples.

Ant Colony Optimization is a family of algorithms that can be applied to any problem that can be transformed into finding an optimal path in a graph. These algorithms are based on the foraging behavior of ant colonies. Briefly, ants start in a node and follow links to other nodes, marking the links they visit with pheromones. These pheromones alter the probability of choosing a path, giving the links in the shortest paths a higher chance to be selected. There are several versions of the original algorithm, given by Marco Dorigo in his PhD thesis in 1992, each of them specify some rules for the ants, the pheromone trails or the way they are distributed in the graph. The convergence of these algorithms, their ability to find a global optimum in finite time) cannot be proved for all the variations and all the situations, and the time needed to find one of these solutions cannot be easily estimated if at all, but this is a general problem of algorithms based on swarm intelligence.

In Particle Swarm Intelligence a set of agents, called particles, move around a problem space selecting the points that improve they're fitness function. This kind of local search allows the optimization of a function without calculating the gradient (the direction of the greatest increase) which is usually very costful. The population of particles fly over the space keeping track of the best position each particle has found so far and a global best position. Each particle move and steer following the direction of the best point it has found and the best position found by the swarm. As with the previous method, and also the Bees algorithm discussed next, the convergence to the best solution cannot be guaranteed in all situations, although the three methods prove to make very acceptable approximations in case the optimum is not found.

If the ant colony optimization was based on the foraging behavior of ants, the Bees Algorithm is based on the behavior of foraging bees. The algorithm starts with a group of scout bees randomly distributed across the problem space. The fitness of each bee is evaluated and the best ones are chosen for a neighborhood search where some more bees are assigned to search the space around the spots found by the scout bees. From these neighborhood searchs the best bees are chosen and the remaining bees start searching again in other places and the process is repeated.

A formal proof of the convergence for the Bees Algorithm has not yet been found but in the case of ACO and PSO it's been proved that under certain circumstances the global optima is found [24]. The variations of these algorithms have not been so thoroughly studied so there's no proof of convergence, even to a local optima, for many of these variations.

Broadly speaking, we can say that swarm intelligence are multiagent systems where the agents are extremely light and the resulting behavior can be seen as an act of intelligence. There are other examples of SI algorithm, for example based on social insects' labor division [25] and the artificial inmune system [26]. There's usually a biological inspiration associated with the algorithms that fall under this label, but there are some techniques that are inspired by social phenomena (e.g. PSO), physical phenomena (e.g. Gravitational Search Algorithm (GSA) [27]) and others.

1.4 Complex Systems and Self–organization

There's been a lot of buzz around the word complex. *Complex Systems* or *Complexity Science* are commonly mentioned in articles ranging from physics to sociology and economics. Although there are a lot of articles dealing with complexity there is not a general agreement about what systems are complex or even what makes something complex. There is a general feeling that something is complex if it cannot be easily understandable. The word comes from latin complexus: com- ("together") and plectere ("to weave, braid"). If we refer to its etimology, something complex should have at least two elements and have some intricate relationship between its elements. Thus, a complex system should be a system with different elements and a mesh of relationships among them.

With this idea in mind there's been two well-defined postures, one where complex systems are just a type of systems [28] and another where complexity is taken as a basic property of the universe [29,30]. If complexity is a basic property of the universe the classical thinking in science is broken as four key points of its method aren't valid anymore [31]:

1.– *Reductionism*: a system cannot be decomposed to its constituent elements as the relationship between them would be broken.

2.– *Determinism*: relations between elements of the system can create feedback loops making the system sensitive to initial conditions, a feature of chaos.
3.– *Correspondence theory of knowledge*: simply by observation we cannot gather complete knowledge of the system, we can just build models that represents some aspects of the system.
4.– *Dualism*: as we can only observe parts of the system we reach a point where a dualism (i.e. something belonging to two exclusive categories) is necessary to explain phenomena (matter-mind, wave-particle)

Even considering complex systems as a kind of system, the previous points can be applied to their study confirming that a new method is needed. Following this view, that complex systems are just a type of system, we can enumerate some properties commonly agreed:

1.– Two or more elements interrelated
2.– Behavior in the edge of chaos (between order and disorder)
3.– Elements not completely dependent nor independent
4.– Emergent properties: local interactions develop a global behavior that cannot be explained by reduction

To this properties we can add self–organization. As everything that has anything to do with complex systems, there's no general consensus about how to describe or measure self–organization. A common property used to characterize self-organization is the creation of order, understood as the negative of entropy, without the intervention of an external agent. The process of self–organization is considered a collective effort of its elements, which act in parallel and in a distributed manner creating a robust and resistant organization [32].

To measure self–organization in physics and related areas we can use the thermodynamical concept of entropy, as stated by its second law: an isolated system tends to evolve to a state of maximum entropy, its thermodynamical equilibrium. In other areas where we can define probabilities over the state space we can use the statistical entropy, defined by:

$$H(P) = -\sum_{s \in S} P(s) \log P(s)$$

where H is the entropy, $P(s)$ is the probability of the system of being in state s from the set of all its possible states S. Using this definition of entropy in the second law of thermodynamics, we can say that a system tends to its most probable state as the maximum statistical entropy is achieved with higher values of $P(s)$

Some argue that self–organization is just a point of view and that any system can thought of as self–organizing [33]. The role of the observer can be emphasized with an example by Stafford Beer (as quoted by Carlos Gershenson in [34]): *When ice cream is taken from a freezer, and put at room temperature, we can say that the ice cream disorganizes, since it loses its purpose of having an icy consistency. But from a physical point of view, it becomes more ordered by achieving equilibrium with the room, as it had done with the freezer.*

It is clear that multiagent systems and by extension swarms are complex systems. Depending on the way we understand complex systems (type of system vs. worldview), we can argue that some MAS are not complex, but the MAS we are going to deal with fall under this label as they are composed by several agents and different relations between them. As stated before, a new approach is needed to understand these systems and what we are going to develop, starting with this dissertation, is a way to design simple swarms to solve some optimization problems based on theories and methods from complexity science.

2 Negotiating Agents

The first step in this research was the use of very light agents to solve a Resource Allocation Problem [35, 36], from the family of Constraint Satisfaction Problems. In a resource allocation problem we have a set of resources that need to be assigned to a series of agents, as for example the allocation of processing time to the users of a mainframe [37] or the assignment of runways to planes in an airport [38].

In general, in a resource allocation problem we have a set of resources that are limited and a set of agents that need these resources in some specific way. The nature and characteristics of these resources are very important when deciding a solution as they define part of the constraints to take into account. The rest of the constraints are defined by the agents or are imposed externally. A solution of the problem is found when we can make a feasible allocation (every agent has a resource assigned) or we find the optimal allocation. In the latter case we must decide on a way to measure the optimality of a solution. Also, there are cases where there is no possible solution.

This kind of problems have been treated in many different ways [39, 40]. One approach recently developed is MultiAgent Resource Allocation (MARA) [41], which uses multiagent systems to solve the allocation problem. This approach comes very natural as instead of programming an abstract algorithm we design a model of the problem, create some behavior for the agents and let system evolve to a solution. The key of this method is to capture the relevant aspects of the problem and define some utility function for the agents.

In the first section of this chapter there is a precise description of the problem, starting with an overview of the problem and then dealing with the relevant details to build the solution. In the second section we explain the design of the system built and why some decisions were made based on the description of the problem. Finally, we will show some results and conclusions, and how this system could be improved.

2.1 Description of the Problem

The problem to be solved was chosen for its familiarity and difficulty. Every year, in our university, the students enroll in some courses, usually five per term. The majority of this courses have two parts, one theoretical and one practical. The theoretical part of the courses are usually not a problem when allocating students, there are enough teachers and classrooms for the lectures, but for the practical part of the courses the laboratory space is limited. Even if there were huge computer labs, the relation between student and teacher is crucial for this part.

The practical sessions take place once a week. At the beginning of each term, the coordinator of the practical sessions for each course defines the number of groups, who is going to teach each of them and when the practical session for each group takes place. The group size is limited by space available in the lab. Students can join any of these groups, but once they choose a group they are required, unless there's a good

reason, to attend to that group until the end of the term. This creates some conflict and competition among the students as they have time restrictions and preferences over the teachers (see fig. 2.1), and the group size limit makes that not every student is satisfied with her assigned groups.

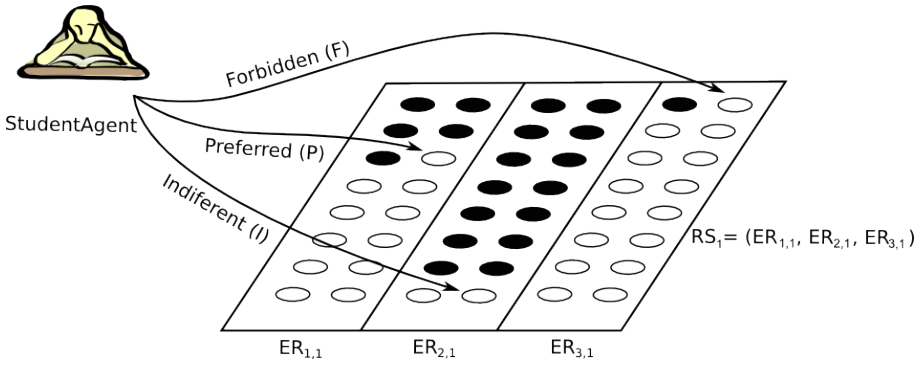

Figure 2.1: Student preferences in a course with three groups. The whole square represent the course, the vertical divisions the groups. White dots are available places, black dots are occupied seats. The student in this figure preferes the first group, cannot assist to sessions in the third group and shows herself indifferent about the second group.

This gets even more complicated as they have more than one course, usually five as mentioned earlier (see fig. 2.2). A student cannot join groups that overlap in time and this adds a dynamic restriction to the problem: once a student is assigned to a group, the time slot when the sessions take place are no longer available for other course's groups. This is a big problem from a CSP point of view, what once was a feasible solution or even an optimal solution can change after one assignment and become a terrible solution, and the other way around, what once was a terrible solution can become an optimal solution.

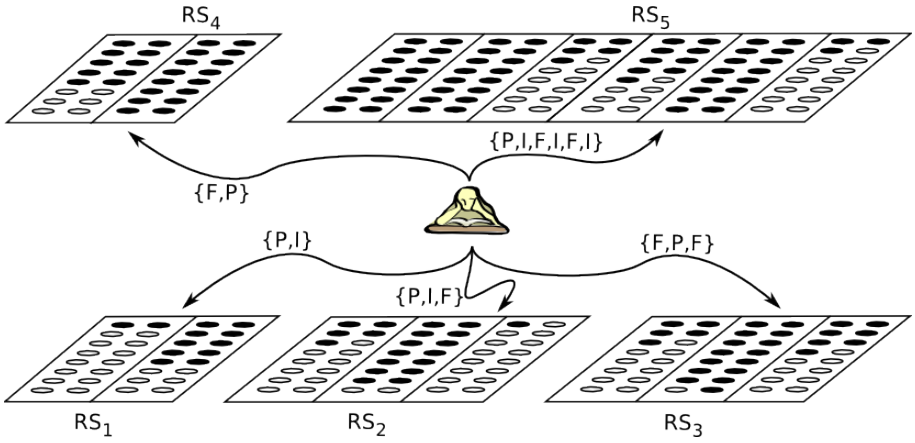

Figure 2.2: A student's global situation. Each RS (Resource Set) represents a course, and each course is divided as explained in fig. 2.1. The values between curly brackets represent the preferences of the student. For example, {P,I,F} means that the student prefers the first group (P), is indifferent about the second group (I) and cannot assist to the third group (F).

Once explained the problem in general terms, we can analyse it precisely to build a solution. In the following description we are going to use Resource Set (RS) for courses and Educational Resource (ER) for the groups. This way ER_1 will be the course number one and $RS_{1,1}$ will be the first group of the first course. To address correctly the problem, it will be described by a set of properties and characteristics (resource types, preference representation, social welfare, allocation procedures, complexity, and simulation) from [41], which can be used

to characterize MARA systems and applications.

- Resource Type
  - *Discrete vs Continuous:* The ER are discrete, they cannot be represented with real numbers nor it can be divided.
  - *Divisible or not:* The ER are not divisible
  - *Sharable or not:* Each ER can be assigned to more than one agent at the same time, but they have limits (seats/places).
  - *Static or not:* The ERs doesn't change during the negotiation, they're not consumable nor perishable.
  - *Single–unit vs Multi–unit:* Each ER is unique, it can't be confused with another ER.
  - *Resource vs Task:* The ER is assigned as a resource, not as a task.
- Preference Representation
  - *Preference structure:* Although our model distinguishes between three different choices (Preferred (P), Indifferent (I), Forbidden (F)) of a particular educational item, and we could construct an ordinal preference structure (where $P > I > F$), we use an evaluation function that translates the agent preference into an integer which is used later to obtain a quantitative value, so a cardinal preference structure is the structure used.
  - *Quantitative preferences:* A utility function is used to map the bundle of resources assigned to an agent into a quantitative value, which will be later maximized.
  - *Ordinal preferences:* Not applicable.
- Social Welfare
  Our approach is based on Collective Utility Function (CUF) because the aim is to maximize the average value of individual agent welfares [42]. In the egalitarian social welfare, the aim is to improve the agent with the lowest welfare and in the utilitarian social welfare the aim is to improve the sum of all welfares, whereas in our approach the global state of the whole agent society by means of the average welfare is the aim of optimisation. Among the different possibilities (pareto optimality, collective utility function (CUF), leximin ordering, generalisations, normalised utility or envy–freeness) a CUF has been selected that defines the utilitarian social welfare as the total sum of agent cardinal values divided by the total number of agents.
- Allocation Procedures
  - *Centralized vs Distributed:* Our approach is fully distributed, since the solution is reached by means of a local negotiation amongst agents and there is not a global perspective of the ERA problem. An aggregation of individual preferences is used and the agent preferences are used to assess the quality of the global resource allocation.
  - *Auction protocols:* No auction algorithm is considered.
  - *Negotiation protocols:* A simplified version of the Concurrent Contract–Net Protocol (CCNP) has been implemented, where each agent can act as a manager and a bidder in the simulation step [43].
  - *Convergence properties:* Our negotiation algorithm needs a multilateral deal, where any interested agent in a particular educational item can negotiate with the manager (in our approach the agent who is trying to obtain a specific allocation).
- Complexity
  Analysis of the models and assumptions, or the computational vs. communication complexity, used in our approach is not relevant for this dissertation.
- Simulation
  A Multi–Agent Simulation Toolkit (MASON) has been used to deploy and test our proposed solution [44] [45].

To summarize the restrictions of this problem, we have that each student must have one, and only one, group assigned for each course with practical sessions. The students have preferences over the groups and are limited by other courses they are assisting, so the solution should maximize the student's happiness and can't assign to a student groups that overlaps in time. To these preferences we add that the groups must be evenly distributed and, by our institution requirement, courses from higher levels must have preference over the rest.

2.2 The design of the system

The main objective of this work was to design a light system capable of solving this problem efficiently. Another objective was to use ideas and concepts that comes natural with the problem described. The multiagent approach let's the researcher use the problem terminology when constructing solution and the system built can be said to be socially inspired.

While building the solution, we created a model of how students interact between themselves when enrolling into groups. We observed that when signing into a group's list, students negotiate between them exchanging positions they have for the ones they want to have. This process is very local as they only contact friends and usually limit themselves to one subject or one group. In the model created, students negotiate with every other student that have something to offer. The exact negotiation process will be explained bellow.

Before explaining the negotiation, we need to understand how the students perceive their status and evaluate the proposals. We define the student's happiness or utility function as a function that increases with the number of groups assigned that they deem preferred. We have to maintain the groups balanced as this is good for the students as it is for the teachers, so the student happiness also varies with the occupation of the group in relation with the occupation of the other groups of this course. The happiness function would have the following form:

$$H(a_i) = \frac{\sum_{i=1}^{n}(f_1(RS_i(a)) + f_2(q, RS_i(a)))}{n}$$

In this equation we have $RS_i(a)$ that returns the group assigned to student a in the RS_i, q is the group's ocupation, f_1 maps the student preference to an integer value, f_2 represents how the student perceive the balance of the group and n is the number of subjects the student has.

As explained in the above section, each student can assign one of three preference categories to each subject: P for Preferred, I for Indifferent and F for Forbidden. The values of f_1 for each preference category is 5 for P, 3 for I and 1 for F. We chose this values so that $P > I + F$ and F is better than not assigning a group.

The f_2 function assigns a value between 1 and 5 to the group occupation. The maximum value is achieved when the group occupation is exactly the same as the mean occupation of the groups, that is the number of students in a course divided by the number of groups that course has. To enroll in an empty group is always better than a full group, so full groups receive the minimum value. The shape of the f_2 function can be seen in fig 2.3.

This is the general version of the happiness function, during the execution of the system there are some variations. The first variation is that, to enforce the assignment of groups for the last year's courses (as required by the institution), when calculating the mean happiness this groups are valued double, we multiply the values of f_1 and f_2 by two. The other variation is that, when negotiating the students can ask once and again for the same group (we will see this later). To avoid this situation, each time a student cannot get a group in a negotiation, the happiness this group gives her will be reduced by a 10%.

The negotiation algorithm employed is a simplification of the Contract-Net Protocol [43]. The negotiation only takes place when a student wants to enter a group that is already full. In that case, the student interested in the group, let's call her the initiator, sends a list of groups she was already assigned to the students in that group, whom we will call the receivers. The list of groups sent only includes the groups that, if changed for the

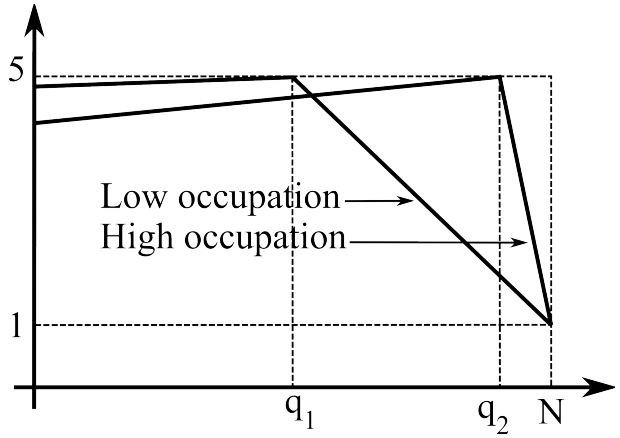

Figure 2.3: Shape of the function that assigns a value to the occupation of a group. q_1 and q_2 are mean occupation of the course, N is the total number of students in the course. The slope of the function varies, depending on the number of students a course has.

group the initiator is interested in, doesn't decrease her current level of happiness. The first receiver interested in an offered group, that is, his happiness is not reduced, swaps places with the initiator. A pseudocode of the algorithm can be seen below in code 2.1.

Code 2.1: Negotiation algorithm

```
1   offer ← AssignedGroups(Initiator)
2   Filter(offer)
3
4   for all Receiver in Group do
5       if Receiver is interested in ER_x ∈ offer then
6           Swap places
7           return true
8       end if
9   end for
10  return false
```

Before starting a negotiation, the student evaluates where would she be better. With the list of courses and groups she has assigned, she evaluates which group assignment can be improved. If changing one group for another can be done, that is, the new group has enough space, the student swaps there directly. If the new group is already full, the student starts a negotiation with the students enrolled in that group. If there's no one to negotiate with or there's no group we can offer because any change would decrease the happiness, a desist factor is applied to that group. The happiness that group contributes to the student happiness is reduced by a 10% and at some point the student will start asking for a different group, possibly in a different course.

To find a solution for the problem, we let this model evolve until an equilibrium state is achieved. This equilibrium state is the state where no student wants to swap places with another student. The students start with no groups assigned and have complete freedom to enroll any group that is not full. As we stated before, MARA let's the designer use terminology and ideas from the problem to be solved with little abstractions needed.

2.3 Experimental Setup and Results

Once the system was built it needed to be tested. This section will detail the datasets employed and how the system built responded. To deepen in the study, the system was compared with a traditional CSP approach, as described in [36].

2.3.1 Data Sets

Several data sets, with incremental constraint-based complexity, have been considered. They have been generated by using real statistical information from the Technical School at Universidad Autónoma de Madrid (UAM). For each course, the number of laboratories available, students registered for each course, capacity of labs, and time tables, has been considered. These data have been used to generate a probability distribution of students/course and the number/capacity of labs that students need to attend. A four year degree has been considered (it corresponds to the current degree in Computer Engineering at UAM). Table 2.1 shows the student enrollment distributions by course (only those courses with laboratories are considered), the number of courses for which students have enrolled and the distribution of students with this number of courses.

Table 2.1: Distribution of labs by course.

Year	No. Courses	Percentage of Students
1^{st}	2	100%
2^{nd}	2 - 3 - 4	20% - 50% - 30%
3^{rd}	4 - 5 - 6	15% - 70% - 15%
4^{th}	4 - 5 - 6	15% - 70% - 15%

Table 2.1 assumes that students from any year has at least one course from that year and no courses from higher years. This means that a third year student has at least one third year course and no fourht year courses. This also can also mean that a third year students can be enrolled in first and second year courses. To simplify the tests, we limited this so that a student can only have courses from one year and the immediatly below. The first and second year has a total number of 2 courses with practical sessions, we don't take into account theoretical-only courses, while third and fourth year have 5 courses with practical sessions.

As an example of how to read the table, the second row represents the second year students. This students can have 2, 3 or 4 courses, and some of them can be from the first year, up to 2 as there are only 2 courses in the first year. From the total number of second year students, a 20% have 2 courses, 50% have 3 courses and 30% have 4 courses.

As mentioned above, students can have courses from one year below the current year. In the second year, students with only 2 courses have both from second year, students with 3 have 1 course from first year and students with 4 have 2 from first year. In Table 2.2 the fraction of courses from another year are shown for the third and fourth years. The table shows the distribution of courses of different years for each student. By examining the real data, it can be noted that the number of students with at least one course from another year is higher than the number of students that have courses only of their own year. In addition, the number students with 6 courses of the same year are very low.

Based on the previous information, six data sets of 1000 students were generated. The synthetic restrictions for each student were also randomly generated by using some historic data related to previous years.

Table 2.2: Distribution of labs for 3^{rd} and 4^{th} year courses.

No. enrolled courses	All the same year (%) year back (%)	One course from one year back (%)	Two courses from one year back (%)	Three courses from one (%)
4	20	60	20	0
5	10	40	50	0
6	5	40	40	15

Table 2 summarizes the basic features for each data set, where Ds0 considers only one Preferred group per ER and StudentAgent (e.g. $\langle\langle P, I, I\rangle, \langle P, I, I\rangle\rangle$), whereas Ds5 considers that 30% of ERs are marked as Preferred and the rest (70%) are Forbidden. For example, $\langle\langle P, F, F\rangle, \langle P, F, F\rangle, \langle P, F, F\rangle\rangle$ makes a 30–70 distribution. The number of Forbidden and Preferred constraints has been adjusted along different datasets to cover different complexity situations.

Table 2.3: Student datasets

Data Set	Preferred groups (P)	Forbidden groups (F)
Test	100%	0%
Ds0	1P/(RS,agent)	0%
Ds1	30%	0%
Ds2	1P/(RS,agent)	20%
Ds3	30%	20%
Ds4	1P/(RS,agent)	50%
Ds5	30%	50%
Ds6	1P/(RS,agent)	70%
Ds7	30%	70%

The difference between the percentage of P and F is the precentage of I. This way, the most restrictive datasets are DS6 and DS7 because the number of I is reduced and in some cases, many cases, there will be no I in the student preferences.

2.3.2 Results

Two systems were tested using these datasets, a CSP [36] and the multiagent system described in the previous section. Tables 3 and 4 show the results obtained for both systems. Since the happiness functions were obtained by using different preference values, they are not comparable across systems but they give a good estimation of their performance with different restrictions. The key values for the comparison are the percentage of P, I, and F in the final allocation.

As seen in Table 2.4, the CSP gives a good result for the first datasets, but when the restrictions are increased the overall happiness decreases and the percentage of F assigned is very high. Altough the execution time was not registered, it was comparatively high when comparing with the MARA approach. While MARA took only a few seconds, the CSP approach need hours to do the assignment. For datasets 3 and 4, it took so long that it was stopped before completion, after 6 hours of execution.

The results obtained for the MAS model are shown in Table 2.5. This method is able, by using the negotiation-based approach, to maintain the global happiness of the solutions found. Although “happiness”

Table 2.4: CSP experimental results for datasets considered.

Data Set	Mean Happiness	Happiness Deviation	Mean Distribution	Distribution Deviation	P's (%)	I's (%)	F's (%)
Test	9.85	0.07	81.5	0,66	100	0	0
Ds0	9.69	0.15	81.3	21.8	86.9	13.1	0
Ds1	9.78	0.1	81.2	15,2	94.2	5.8	0
Ds2	9.32	0.25	81	17.7	89.8	1.07	9.19
Ds5	7.82	1.09	80.6	17.9	61.3	15.6	23.1
Ds6	7.03	1.12	80.8	15.8	66	8.2	25.8
Ds7	6.02	1.59	80.9	19.5	71.9	0	28.1

Table 2.5: MAS experimental results for datasets considered.

Data Set	Mean Happiness	Happiness Deviation	Mean Distribution	Distribution Deviation	P's (%)	I's (%)	F's (%)
Test	9.85	0.07	81.5	0,66	100	0	0
Ds0	9.3	0.29	81.3	4.21	85.3	14.7	0
Ds1	9.7	0.14	81.4	0.64	94	6	0
Ds2	9.4	0.24	81.1	4.36	86.9	12.75	0.35
Ds3	9.8	0.12	80.4	2.04	95.8	3.67	0.49
Ds4	9.2	0.33	80.8	6.94	85.7	12.5	1.8
Ds5	9.7	0.19	81.0	0.89	95.5	2.42	2.05
Ds6	9.1	0.43	81.0	5.92	86.0	11.1	2.9
Ds7	9.7	0.29	80.9	0.91	95.8	0	4.2

values cannot be directly compared between CSP and MAS solutions (because their equations are different), the percentage of allocated F can be compared. In the worst situation (DS7) only the 4% of the students needs to be assigned to a forbidden ER (F), and the 95% of student teams are satisfactory allocated. Finally, the low variation of the happiness among the different datasets can be remarked compared to the variation of this value for the CSP solutions. This is due to the facility (given by the Multi-agent approach) to change preference values, or to exchange the current ER allocation with other agent in the system.

2.4 Conclusion

This first approach to building a light multiagent system was successful. The goal of the whole work was to find ways to build very simple agents that are capable of solving some problems. The agents in this system were very simple as they didn't use some reasoning mechanism, neither they have some complete world representation or any characteristic of heavy agents. Even though, these agents can be simplified more, for example, by eliminating the negotiation mechanism and building pure reactive agents. This can be an arduous task as we can say that each agent here is represented in multiple planes, one plane for each RS she is enrolled into. The next step, described in the next chapter, would be to use simple reactive agents to solve a problem on a single plane.

We will review this system again in chapter 4, were it will be explained using Complexity Science terminology.

3 Swarm of Static Agents

In the previous chapter we defined a multiagent system to solve a complicated problem. The main objective of the present dissertation is to explore the intricacies of building such systems and find concepts and ideas that can be used to facilitate the creation of multiagent systems to solve problems. As could be seen in the results of the previously described system, it is possible to build very simple systems to solve complicated problems with very good results. This encourages the question: How simple can be an agent and still be capable of solving a problem?

It is very important to simplify the agents in a multiagent system. Two fundamental problems when building multiagent systems is the processing needs of the agents and the load of the communication channels. In the student system defined earlier, the agents were very simple, but their communication needs were very high. Each student needed to communicate with a complete group when they needed to negotiate. This means that in a worst case scenario, in a situation with 1000 students and a mean group size of 25, around 25000 messages will be sent and this messages need to be processed and answered.

The problem of allocating groups to students is very difficult to solve, and simplifying the agents will prove to be very hard. When undertaking a difficult task as this one, it is best to start from small simple things. As the first step trying to build very light agents capable of solving an allocation problem, we will build simple agents capable of solving a classical benchmark problem: the N–Queen problem.

3.1 Ideas behind the design

When designing the system to solve the student–group allocation problem we stated that it was inspired by the interactions of students. Generally, copying natural or social systems is a good starting point when building this kind of systems. Natural and social systems are complex and, in some way, solve problems with the interactions of its elements. In nature there are systems where a huge number of individuals have to cooperate, allocate roles or some other tasks requiring a lot of effort by the individuals or a lot of message passing. There are some ideas that can be borrowed from them.

Two mechanisms from natural systems can help to reduce the channels' load and the computational costs: memory and message discrimination. Examples of this mechanisms can be found in the neural systems. The main memory mechanism in neurons is the voltage accumulation. A neuron can fire in response to a train of spikes depending on the excitation level generated. Also, recent studies on neuroscience show the importance of neural signatures on information processing [46]. This signature reading capability enables each neuron to recognise the origin of a message and change its behavior based on the origin. In self-organizing groups of neurons, the signatures are essential to maintain the order [47].

This two characteristics are not only found in neuroscience. In mirmecology, the science that studies ant's behavior, it is known that ant foraging is a result of local interaction where ants become foragers based on the

information extracted from a pheromone trail: direction and intensity (discrimination) and number of ants seen following the trail (memory) [10].

Using this two mechanisms, memory and message discrimination, and taking into account the topology or structure of the system, it is possible to build multiagent systems keeping the agents simple and reducing the message loads. Having identified this two mechanisms, we can propose a framework or methodology that, based on establishing simple methods for memory and communication, gives way to the design of simpler multiagent systems.

3.2 Key concepts of the model proposed

The ideas described in the introduction are loosely defined and can be applied in different ways. For example, these biologically inspired mechanisms has been used in artificial neural networks as in [48]. Before discussing the implementation of the system in the practical case, the N–Queen problem, there are some general characteristics of the system that can be abstracted as a framework and used in other problems.

As in any other multiagent system, there is a set of agents with its own properties and possible actions. Here, the agents are fixed, they cannot move, and the only actions possible are: send messages locally, process messages and change state. Depending on the problem, the agents can have diferent states and they change from one state to another according to the messages received.

The effect of the messages is accumulated in a variable that we call “tension”. It works as a memory of the messages received. The tension is altered by two mechanisms, it is reinforced by the messages received and relaxed some fixed amount at each time step. The amount the tension increases with a message depends on its properties, the agents must have some message discrimination mechanism.

It is important to mention that the message passing capability of the agents is local and thus it prevents the saturation of the channels. The agents can only send messages to its neighbours, limiting this way the use of the communication channels, although the topology of the network of agents is defined by the problem and it can be the case that it requires the agents to broadcast their messages.

To sum up, the system proposed is based on simple static agents with different states. The state of an agent depends on a tension variable, which changes as a result of the local message passing. The solution to the problem is the arrangement of the agents in the topology, taking into account the states of the agents.

There are several decisions to be made to solve a problem using this approach, in the next section we use a practical case study to illustrate how to make them.

3.3 An application to the N–Queen Problem

The N–Queen problem was originally proposed by a chess player, Max Bezzel, in 1848. The goal is to find an arrangement of N queens in a chess board of $N \times N$, where no queen attacks the others following the chess rules. Altough the problem is solved in [49], it has been used as a search algorithm benchmark in AI and combinatorial optimization.

The typical solution of this problem is using depth first search (DFS) [50]. This search can be optimized

by using heuristics [51], but the branching factor and the number of backtracks and memory needed increases exponentially with the size of the board.

In 1990, an approach that finds solutions in polynomial time was published [52], based on permutations and randomness. Current research involves the use of multiagent systems [53], artificial life [54] and genetic algorithms [55] to find solutions to this problem.

The system is built based on the ideas of the previous section. The agents are fixed and can only exchange messages and change their state. As the problem to solve is the N–Queen problem, the topology is a $N \times N$ board where each square is an agent.

The agents have two states, "queen" and "empty". We can see a solution to the N–Queen problem in figure 3.1. Each square has an agent and the agents in queen state don't attack each other. An attack is defined here as receiving a message from another queen. It's important to notice that in any moment an agent can be in any state, this means that there can be more (or less) than N queens on the board.

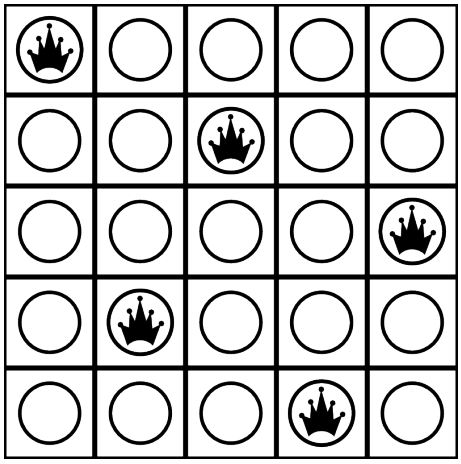

Figure 3.1: A solution of the problem in terms of the agent system defined. The empty circles are agents in the empty state and the circles with a queen are agents in the queen state.

An agent is in queen state can initiate messages, passing the message to the adjacent squares. On the other hand, on empty state it can only propagate the messages received in the opposite direction it received the message, that is, a message received from the left is propagated to the right and a message received from the upper–left corner is propagated to the bottom-right corner. The queen agents don't propagate the messages received, they just modify their tension and, if necessary, change state.

The messages only have two fields, direction and distance, with this two parameters any agent can identify its procedence. The agent receiving a message, either a queen or an empty square, modify its tension variable according to the distance and the procedence. If the message was originated near the agent the impact on its tension will be greater, as can be seen in figure 3.2.

The equation that rules over the tension takes into account the messages the agent receives increasing the tension and, to reduce the effect of this increase, a relax factor is discounted every step. The increment of tension depends not only on the distance but also on the insistence of the agent sending it, identified by direction and distance. This means that the first message from an agent will produce less tension than the subsequent messages. The equation for the tension increment is as follows:

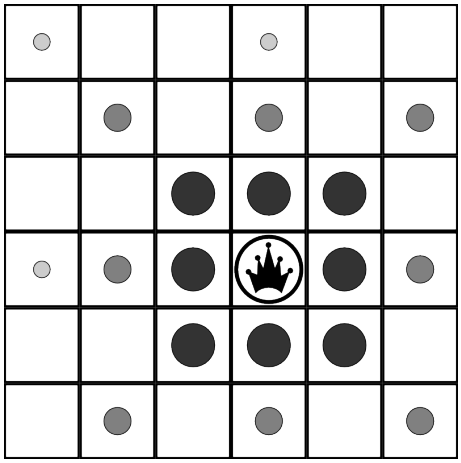

Figure 3.2: Message influence attenuation. The queen agent sends a message (grey circles) in all directions and its influence diminishes, illustrated by the size and intensity of the dot, as the message is propagated.

$$\Delta T = -R + \sum_{\forall m} I(m) \cdot [1 - (dist(m) - 1) \cdot S]$$

Where R is the relax factor, m is the message, I(m) is the number of times the agent received a message with the same origin, dist(m) represents the distance and S is the amount the message reduces its effects on the tension for each distance step. The maximum increment a single message can produce is 1, before multiplying by the insistence I(m). This increment is applied when the message originated in an adjacent cell, that is, the distance is 1. At distance 2, the message can produce an increment of $1 - S$, and so on. The minimum increment for a message is, therefore, $1 - (N - 2) \cdot S$.

In figure 3.3 is represented an example of how the tension varies after processing a message. The empty state agent modifies its T only by -R as it does not receive any message. The other two agents, in queen state, both receive just one message and their tension is increased the same amount: $\Delta T = -R + (1 - S)$. Taking $R = 0.07$, $I(m) = 1$ for both agents and $S = 0.33$ we have that both agents increase their tension by 0.65.

Two thresholds are defined for the state change. When the tension increases above the upper threshold the agent changes its state to “empty” and when it decreases below the lower threshold the agent changes its state to “queen”. For tension values between the upper and the lower limit, the agent stays in the same state. In physics this is called a hysteresis process and here it is used to avoid oscillation: small fluctuations on the tension will not change the state repeatedly. Also in figure 3.3, it’s represented 3 possible cases of this change of state. The diagrams represents a simplified board situation, where the agents are only influenced by the agents drawn.

There are going to be five parameters: the board size, the relax factor, the tension step and the thresholds. Each of these parameters are analysed in this section. Also, to reduce the message load of the system, the queen agents send the messages with a fixed probability. The optimal parameters cannot be extracted without previous study, after all this is a complex system, but for the experiments we are going to run we use some values trying to avoid extremes and looking for some symmetry. The only value that varies between experiments with different board sizes is the relax factor and it is chosen through a trial and error process. Table 3.1 summarizes the values used in the experiments. The slightest change in the parameters can change

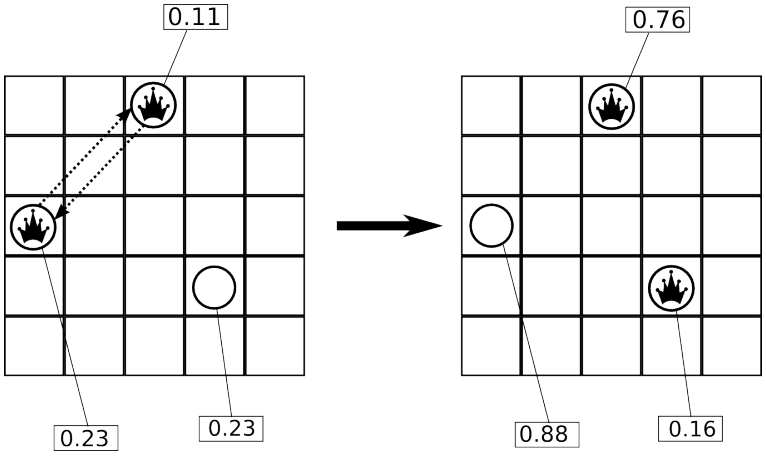

Figure 3.3: Three possible change of state, only considering the agents depicted. The dotted line represents a message passing through the agent between the queens. The numbers on the boxes are the values of the tension for that agent. When a message increases the tension over the upper threshold (0.8 in the example) the agent changes to empty state. If the relax factor substracted brings the tension below the lower threshold (0.2), the agent changes to the queen state. In any other case the agent remains in the same state.

completely the behavior of the system. By testing some values, we couldn't find out a direct relation between the parameters and some system properties as oscillation, convergence speed or messages passed until a stable state is found. As the main objective of this work was to evaluate the feasibility of a system with these characteristics, it is left for future experiments to determine this relations or a method to set the parameters' to optimal or near optimal values.

Table 3.1: Parameters used to solve the N–Queen problem.

Parameter	Value
Lower threshold	0.2
Upper threshold	0.8
Sending probability	0.5
Minimum tension per message	0.01
Maximum tension per message	1

In figure 3.4 can be seen that the time needed to solve the problem increases exponentially with the size of the board. This result is not necessarily bad as the growth is very slow and the times are reasonable, it is a good result for a system that is completely autonomous. Also, it must be noticed that the time is anotated when the system is stable in a solution but in the process the system can find solutions that are unstable.

A problem that arises from a self-organizing system is when to consider that a global state is a stable solution. For this purpose it is always needed a supervisor, some agent or process that analyses the whole system and checks if the solution has been found. This must not be confused as some kind of central authority, the role of this supervisor is only to stop the simulation when the system doesn't leave a state for some number of simulation steps. It has no control over the agents' behavior and, after the parameters are tested and a solution is proved to be found, the system can be used without this supervisor.

In the case of these experiments, the agents in queen state report to the supervisor when they have not

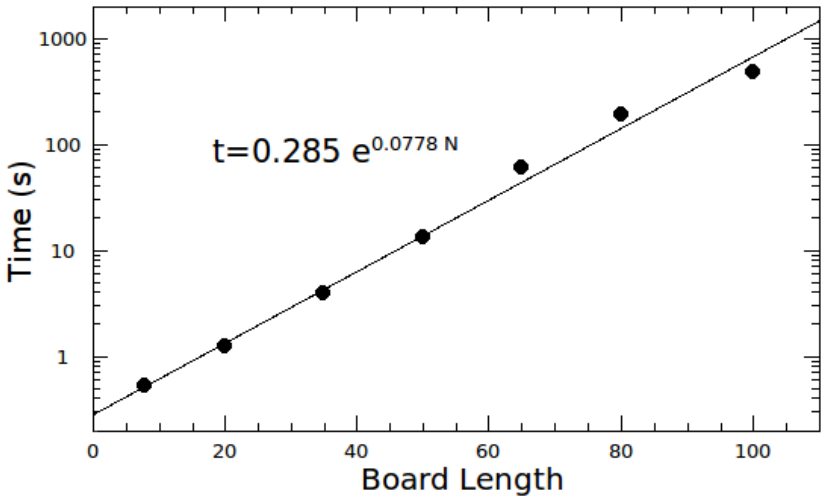

Figure 3.4: Mean time taken to get to a stable state. 50 measures for each size.

received any messages in a step of the simulation. A state of the board is considered a stable solution when the number of queens reporting is the number of queens needed for a solution (8 for an 8x8 board, 20 for a 20x20 board, ...) and this number doesn't change for a number of steps equal to the length of the board. This criterium is taken because the maximum number of steps for a message to reach another queen or the end of the board is exactly the length of the board minus one, as the message propagates one square on each step and this is the time taken for a message to go from one corner to another.

One of the goals of this system is to reduce the message load and this is achieved in two ways. First, the message is very small, only two integers, so if we are using a computer network the contents can be passed on a single packet. On the other hand, the message sending is randomly distributed with a probability of 0.5. The sending probability could be reduced further if needed, but as the rate decreases the system becomes more unstable and an adjustment to the other parameters is necessary.

The relax factor, the only free parameter in these experiments, follows a negative exponential curve, as seen on table 3.5. A small change in this value for a given board size made the system behave irregularly, sometimes increasing the convergence time and other times making the system look completely random.

This information about the relax factor makes clear two things. First, the system is really a complex system as a slight variation of a parameter makes the system behavior different. Second, as the minimum tension a message produces is 0.01, the relax factor can't decrease below that value. This means that the relax factor is not the only value relevant for the system behavior, for big board lengths the exponential curve goes below that value.

3.4 Conclusions

In this chapter we presented a possible way of simplifying multiagent systems. Concentrating on two mechanisms of the system, memory and message processing, and trying to make them as simple as possible we

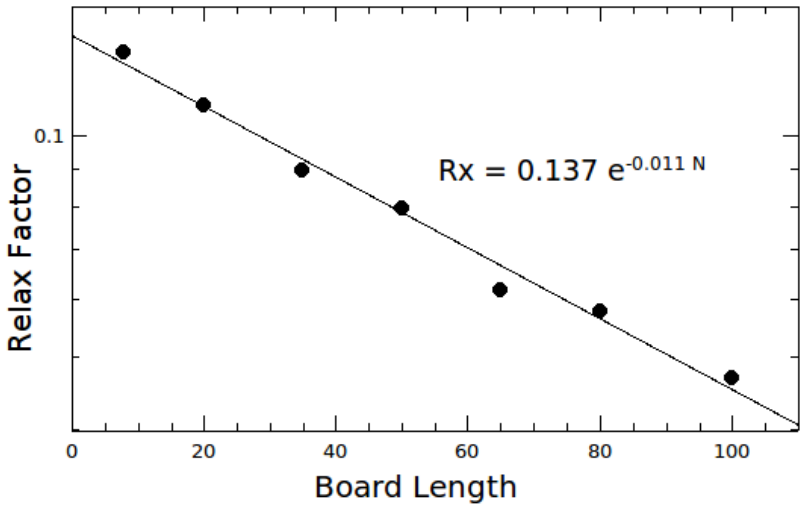

Figure 3.5: The relax factor used for the different board length follows a decreasing exponential curve.

were able to build a solution for a simple problem. The results obtained were not optimal because the system was not fine tuned. Fine tuning the system would be a difficult task and to find the correct parameters we would need to use some method, as genetic algorithms or neural networks. As the main goal of the work was to find a simple way to build multiagent systems to find solutions to some problems, the optimization step was not yet taken.

The mechanisms proposed for the system were memory by accumulation and processing a message by its distance and direction. The biggest simulation done with this system was a 100x100 board, a total number of 10,000 agents. This simulation took a little more than 10 minutes to get to a solution on a single processor computer. Using a distributed environmet would probably throw some better results as in our simulations the agents action were executed sequentially, one step at a time.

As repeated throughout this dissertation, this a first step towards obtaining a methodology for building multiagent systems based on complexity science. By now we haven't mentioned much the complex approach, but it is clear that both systems built are complex by our definition on the introduction. On the next section we will try to see both systems under the light of complexity science.

4 A Complexity Science Approach

In the previous chapters we discussed two systems built to solve optimization problems. The first approach, the negotiating agents system, tried to find an allocation of the students in groups, maximising their happiness and balancing each group. Although it was succesful finding an allocation and it used very simple agents, the communication load could increase as the groups grow bigger and the number of courses and groups augmented. Also, depending on the happiness function, the processing needs of the agents can make the system expensive computationally.

With the second system we established a base to start a process of simplification. The agents were very light and the messages passed were small. It can also be said that the agents may need to send bigger messages or greater processing needs, but what was proposed was not that precise system but to center the attention on those two mechanisms, memory and message discrimination, in order to simplify the system. The results show that it is possible to build systems with simple memory mechanisms and message processing capabilities that are capable of solving problems.

In this chapter, we will try to describe the systems or parts of them using the complexity science terminology. This will help us to further analyse the built systems and get some notions of why did they work and some intuition on how it could be improved. The analysis is based on the definitions found in [29, 32, 34].

4.1 Complex systems and self–organization revisited

In the introduction we gave a brief overview of concepts and ideas from complexity science and why this approach was necessary for a better understanding of swarm intelligence and multiagent systems. Now, in this section, these concepts will be expanded so we can later identify them in the systems developed.

As stated before, we will label a system as complex if it is a system composed by different agents with some relations between them that cannot be reduced to its parts to analyse. The behavior of the whole system is different from the sum of its parts, the relations play a big role in the system as they can change the behavior of the agents or modulate the effect of one agent over another. This interactions are local and can vary through the systems evolution, but the effect they achieve is global.

There are some characteristics associated with complex systems, some positive and some negative. As negative characteristics we have that they are unpredictable and uncontrollable. At most, and only in some systems, we can find statistical regularities or understand some qualitative behavior. As positive characteristics, we have that these systems are flexible, autonomous and robust. The positive characteristics are desirable for any system we build, but the negative ones makes these systems inappropiate as we can't say a priori if it will achieve the goal we have in mind and we can't direct the system toward that goal.

In complex systems we can identify positive and negative feedback. The positive feedback amplifies the effect on the system of some changes while the negative feedback reduces its effect. This makes the system

to be non–linear: the effects are not proportional to the causes. The positive feedback makes the system more unstable and unpredictable, small changes in the initial conditions can be amplified so the equilibrium state of the system is completly different from the equilibrium achieved before. This is one of the characteristics of chaos. On the other hand we have negative feedback, that makes the system stable. Negative feedback reduces the effect of a change in the initial conditions, gradually suppressing them.

Systems with positive feedback can be controlled in some measure as introducing a change in it we can move the system from its state, but these systems are very unpredictable as small differences can produce different outcomes. With negative feedback we have the opposite case, the systems are very predictable as perturbations barely affects them but they cannot be controlled as our efforts will be suppressed by the system.

Everything said above is present in our systems, we will go through it in the next section. As both systems try to find an optimal solution we need to dive into self–organized systems. In the introduction we stated that self–organization depends on the observer as a definition of “organized” is needed. In both of the systems we will say that the systems are organized if they are in a solution state, the further they are from this solution the more disorganized they are.

In a self–organized system we have that the global order is achieved by local interactions, there is no agent in charge and agents don’t have a representation of the global state. Thus, the system is robust and resilient as there’s not a single point of failure. One thing that can be remarked here, as a cause or consequence of this distributed control, is that the agents coevolve, they adapt to each other. The agents have preferences over the possible states of the system, at least over the limited view they have of the system, and this coevolution process let’s them converge to a state where every agent is satisfied.

Self–organizing complex systems have very desirable properties for any engineering project. We can summarize them has follows:

- Flexibility
- Adaptivity
- Autonomy
- Robustness
- Resilience
- Distributed control

But these properties have two inconvinients, these systems are uncontrollable and unpredictable. Finding a way to improve the control or the predictability we could build systems that benefit from these properties. In the case of systems that cannot be redisigned, it has been proposed the use of a mediator [34]. This mediator is not a central authority. Sometimes it is an agent that mediates between agents in the system, other times are properties of the agents or restrictions on the system. The function of a mediator is to reduce the “friction” in the system. When designing a system from scratch, we can say one of two things: (1) there is no mediator because every property comes from the system by itself or (2) that we introduce this mediator in the design to achieve more control.

In any case, to achieve a better control and predictability in the system we need to look for a balance between positive and negative feedback. As we told before, positive feedback makes the system more controllable while the negative feedback makes it more predictable. Identifying both feedbacks in a system could lead to a better understanding of it and how to make it more controllable or predictable.

Full control or precise predictions of a system like this is impossible as the system can evolve and respond differently to inputs introduced in a previous time. With this conclusion we arrive to the paradox that to have a system fully under control, the controller need to contain a copy of the system it is controlling [56]. This is similar to the solution given to the halting problem: to know if a system will halt we need to let it run for infinite time and see if it does. As we are not trying to build a controller for a system but the system itself this consideration will not affect this work, it just reinforces the idea that a good design is needed because it is harder to build a control mechanism afterwards.

In the next sections we will identify this concepts in our systems and explain how they were taken care of. This chapter will be used as a foundation for further experiments and future development of a methodology based on complexity science.

4.2 The complexity of the negotiating agents

In the case of negotiating agents we can identify two sources of complexity. On one hand we have the problem itself. The courses and groups are related to each other, and also are the students. The relationships between the different parts of the problem makes it a complex environment and if we introduce some change in it, e.g. change a group's time slot or the student preference, the reaction is unpredictable as we can't determine how other students will react or how the constraints will change during a simulation, among other things.

On the other hand we have the built system. As cyberneticists say, the best way to deal with a complex system is with another complex system. The built system is completely based on the problem, each part of it is a model of some aspect or aspects of the problem. This makes the built system to be adaptive and flexible: after a solution is found, we can translate any change in the original environment into the system and let it find a new solution. This adativity and flexibility is not available in other methods as, for example, backtracking. If we want to introduce a change in the problem, we would need to run the backtracking algorithm from the begining because some branches that where not useful before can lead to solutions now.

The built system is composed of different agents, each one of them can be differentiated from the others by the courses they have and the preferences over the groups in each course. The interaction between the agents is local, they can only communicate with a limited part of the system. At most, an agent can establish communication with other agents in the same courses it has, but normally they will communicate with only a subset of these agents. Although this communication is local, we have a global situation that the agents are not aware of.

The outcome of the system, the stable state, cannot be traced back to its agents. We can say that agent X enrolled in a group she can't attend, i.e. has an F for that group in her preference vector, because the group she wanted was full and no one there is interested in other groups she has. But then we need to go back and see why nobody is interested or why she couldn't enter the group in the first place because everyone had an equal chance to get into it. This causal chain is long, complicated and full of suppositions and, although we could find some probabilities over the links, we cannot fully explain why the resulting state is the one it is.

Before we said that complex self–organizing systems have feedback mechanisms. Feedback mechanisms defines how outputs are related to inputs. Before defining inputs and outputs we must define where are the borders of the system, in other words, what is part of the system and what is part of the environment. As

we don't expect changes in the courses or groups, we define the system as the group of agents and the environment as the courses and groups. This way we have that the problem defines an environment with the following variables:

- List of courses
- Number of groups for each course
- Size of the groups
- Groups' hours
- Number of agents
- Enrolled courses for each agent
- Preference vectors for each agent

This variables can be understood as how the agent models its environment and its goals, they belong to both, the environment and the system, but they are not fed into the system nor they are extracted from it. Previously we discussed the adaptivity of this kind of systems and how we could change the environment while the system is running. This stays true after this separation, what changes is how we introduce these modifications.

The rest of variables depends directly on the agents and they can modify them as they wish. From this variables we will define as an output, i.e. variables that the system shows to its environment, the assigned groups for each agent and the only input would be the distribution of students in groups. To define the state of an agent we use the list assigned groups, the number of time it has attempted to enter a group and its happiness.

To recapitulate, we have that the problem is defined by variables that determine the courses and groups and their relationship with the agents. We can consider this variables as immutable and part of the agents' internal representation of the world. The list of groups assigned, the number of times an agent has attempted to enter a group and its happiness are the only variables necessary to determine the agents state. The system constantly receives information about the state of the groups, meaning that it knows what students are already in a group or if a group is full. In exchange, the system informs about changes it makes in the groups.

We can see clearly that there is a feedback loop, the state of the groups is fed to the agents and the agents inform about any changes in them. This loop doesn't determine by itself if it is a positive or negative feedback, it depends on how the system treats this information. In our case, this loop is managed in two ways: with the insistence factor and the f_2 function, both in the happiness equation (see section 2.2).

The insistence factor is applied when the agents try to enter any group more than once without any success. The relation with the feedback is a little difficult to see, first the agent receives the status of the groups for the courses it has enrolled. The agent evaluates if there is a group that would increase its happiness and, if the group is full, it tries to negotiate with other student to enter. It is possible that the list of groups an agent receives doesn't change from one attempt to the next and the agent finds that the group that improves it happiness is the same full group each turn. To dissuade the agents, the insistence factor is applied. This makes the feedback loop negative as it turns the output to be, after some time, the same as the input, whatever the input is, and the agents to stop trying to change groups.

For the other way to manage feedback, the f_2 function, it is clearer that it affects the feedback, but wether it does in a positive or negative way is hard to decide. When a group is below what was defined as the optimal

occupancy this function encourages the agents to enroll this group. Once this quota is filled, the function dissuades agents from entering and sending them to other groups that are less full by reducing very fast the happiness this group can provide.

The value for the insistince factor and the shape of the f_2 function must be chosen carefully and with a goal in mind. This two mechanisms reduce the agent activity and drives the system to a stable state. The insistence factor is negative in nature as it forces the students to stop their activity. The f_2 can have a positive and negative effect, depending if the occupancy of the group is below the optimal or over it. This function also increases the variety of options, as agents does not only search for preferred group but also for groups that are not overcrowded.

As we can see, the built system is complex and self–organizes. This gives great flexibility when finding a solution, but makes the system hard to control. By introducing mechanisms to control the feedback we can ensure that the system will come to an equilibrium where the agents would stop searching, but a fine tuning of the system or proving that it will find the optimal solution are very difficult tasks if not impossible.

4.3 A complex swarm

The swarm system is complex as is the system discussed before. It benefits from the same positive characteristics and suffers from the negative ones, the swarm is flexible, adaptive, autonomous, robust and resilient, but also uncontrollable and unpredictable. Being the agents so simple, the system is based more on the interactions between the agents than on the agents themselves.

In the previous system it was clear that the agents were trying to maximize their happiness. To have a better understanding of how the swarm agents behave, we can model them as maximizers of the following function:

$$U(a_i) = \delta(1 - s_i)(1 - T) + \delta(s_i)T$$

U is the utility function of the agent, δ is the Kronecker delta that returns 1 if the argument is null and 0 otherwise, s_i is the state of the agent and T is the tension. The value of s_i is 1 for queen state and 0 for empty state. Thus, we have that the agents on the board try to minimize the tension when in queen state and maximize it when in empty state. If we define more states we must have clear what they are trying to maximize in each state, supposing we are following the architecture defined in section 3.2.

Having this clear we have to define, as in the previous case, the boundaries of the system. The environment in this case is the board and the only properties it has is its size. The agents come with its properties already assigned: thresholds, sending rate and relax factor. We will suppose that the agents communicate through the environment, each agent places the messages in the square next to itself and reads messages from the square they are in. This way we have that the input and output of the system are the messages. The state of the system is defined by the state and the tension of each agent.

Again, there is a clear feedback as input and output coincide. As before, we can't determine if the feedback is positive or negative if we don't look at how this feedback is processed by the agents. The only sources of messages are the agents in queen state, the other agents just propagate the messages until the messages

reach the agents in the sides and the corners. If there were no messages, all agents would be in queen state as the messages are source of tension. We then can measure the activity of the system as the number of messages moving around the board.

Having messages that do not increase the tension of its receiver, the system would be very active as every agent would be sending messages in queen state. On the other hand, if the messages increase the tension to its maximum value we would have an oscillating system that goes through phases where a lot of agents, if not every agent, would be sending messages and inmediatly turning to empty state because of a received message. The oscillating behavior would depend heavily on how the message is propagated.

The system needs to stay in a middle ground, where the agents can increase the tension from others but without forcing them to change their state. This can only be achieved controlling how the message is processed. As was told in the first section of this chapter, to achieve an equilibrium we need a balance between positive and negative feedback. We have that the system by itself, without considering the effect of the messages in the tension, tends to be very active so we need a way to reduce this activity. We need the messages to act as a negative feedback to keep agent from activating constantly.

The decisions on how this negative feedback works depends on how we want the system to behave. Different choices of parameters will produce different behaviors. If we choose a constant value for the amount of tension a message increases instead of a decaying value, we would not be considering the delay of the message as the source agent maybe in empty state once the message reaches the border, and different values for maximum and minimum values for this increment will make it possible to have more than one agent in a line of attack.

In any case, as we are in control of the everything, system and environment, we can control other variables as the relax factor or the thresholds. This values affect the behavior of the system and must be chosen carefully. For example, if we make the agents less prone to turn to queen state, if we reduce its relax factor, the convergence of the system will be very slow. If we choose thresholds very close to each other, or even just one threshold, the system will oscillate more as little variations in the tension could be triggering the queen state on and off continually.

The adjustment of these parameters is very complicated, further tests are needed to determine the effect of thes parameters on some qualitative aspects of the system and how they relate to each other, but being a complex system with 6 free parameters it maybe impossible to build a complete map of values–behavior.

4.4 Conclusions

Using terminology and ideas from complexity science we improved our understanding of the systems. Although the analysis presented was only superficial, only identifying concepts and ideas in the systems, we could explain why some of the behaviors of the agents in both systems lead the system to a stable state.

The analysis consisted on four steps. First, system and environment were delimited, establishing what was part of the system and what was part of the environment. The delimitation was made considering only the simulation elements. After identifying the elements, tha variables of both, system and environment, were divided as problem variables, the system's inputs and outputs and the state variables.

With the inputs and outputs identifyed we can see the relations between them, finding the feedback loops. These feedback loops are essential in a complex system as they make the system non–linear. Finally, the key components of the systems were found by looking at how the system dealt with the feedback.

In both cases, the feedback loop was easily identifiable because the inputs were equal to the outputs, and both systems had mechanisms to transform the feedback into a positive and negative feedback. The balance between positive and negative feedback makes the system complex, the positive feedback amplifies the input and increases the activity of the agents while the negative feedback reduces the effect of an input and stabilizes the agents.

This analysis is very shallow but was useful. Using more techniques and ideas from complexity science will give a better understanding of the inner workings of both systems and the relations between its parts and between the parameters and the resulting behavior.

5 Next Steps in the Research

The idea behind this work is to create a framework or methodology for the creation of multiagent systems, understanding that these systems are complex and that they should be studied using complexity science. The development of a methodology or framework is very complicated and it should be done in small steps.

First, before making assumptions and false steps, it is necessary to be familiar with the systems we are going to work with and this was the first stage of the development. We took a complicated enough problem, the student–class allocation problem, so we could understand how these systems are built and what problems may arise. Once we found a solution, instead of finding improvements or deploying the system in a real environment we started the process of simplification.

For the simplification process we used a simple problem because the student–class allocation problem was very complicated to identify the relevant parts of the system. The problem chosen, the N–Queen problem, was simple and we could relate both problems as we can define both, this problem and a regular scheduling problem as “structural” problems. With “structural” we mean that solutions for both problems can be seen as layouts of elements in a grid, in this case a board and in scheduling problems a timetable.

The results for both systems were satisfactory and an initial analysis of them was made. This analysis will be useful for further study and development, we can base the next steps in the research on these results.

From here, the next logical step would be to apply the second system, the swarm, to a more complicated problem. This problem can be a completely different one or some related problem. For example, we could apply the swarm to a multidimensional N–Queen problem, which is closer to the initial problem of student–class allocation, or we could use the system to solve the map coloring problem or the traveling salesman problem.

There are open spots in this work that could be reinforced in case that the application of the swarm to new problems fail or if the results were not interesting. The optimization of both systems is still open, they can be improved, and also we could run more experiments in both of them to find, for example, the relation between the variables and behaviors in the swarm or how the agents of the first system would react to changes in the environment.

In the case that there is some interest in the swarm, we propose two experiments. The first one is to use heterogeneous agents in the swarm, each agent having thresholds and other parameters that don’t need to be the same for every agent. This, based on the law of requisite variety [56], should give us better results than using homogeneous agents, but to prove this point will be hard.

The other experiment we propose, or rather, set of experiements, is to study the behavior of the swarm on three different aspects: message passing, oscillation and convergence speed. This set of experiment will be needed sooner or later if the methodology is to be complete. These three aspects represents characteristics

that are desirable for an engineer deploying this solution in a real environment. This engineer could have some restriction to the use of communication channels, could need to find a solution fast no matter how or could need a system that don't oscillate as the machines the agents represent are very sensitive. Usually, the needs of the engineer would be a mix of these requirements.

After all this and some other work is done the results could be gathered together into a final framework or methodology for the design of multiagent systems, more precisely swarms, applied to the solution of different problems.

BIBLIOGRAPHY

[1] J. Searle. Minds, brains and programs. *Behavioral and Brain Sciences*, 3, 1980.

[2] D.A. Forsyth and J. Ponce. *Computer Vision: A modern approach*. Prentice Hall Professional Technical Reference, 2002.

[3] D. Jurafsky, J.H. Martin, and A. Kehler. *Speech and Language Processing: An introduction to natural language processing, computational linguistics and speech recognition*. MIT Press, 2000.

[4] S.J. Russell and P Norvig. *Artificial Intelligence: A modern approach*. Prentice Hall, 2009.

[5] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley and Sons Ltd., 2009.

[6] V.R. Lesser. *Encyclopedia of Computer Science*. John Wiley and Sons Ltd., 2003.

[7] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, game theoretic and logical foundation*. Cambridge University Press, 2008.

[8] N. Gilbert. *Simulation for the Social Scientist*. Open university Press, 2005.

[9] H.A Simon. *Simulation for the Social Scientist*. Open University Press, 2005.

[10] D.M. Gordon and N.J. Mehdiabadi. Encounter rate and task allocation in harvester ants. *Behavioral Ecology and Sociobiology*, 45(5), 1999.

[11] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[12] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1), 1992.

[13] J.G. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[14] J.G. Gaschnig. *Performance Measurement and Analysis of Certain Search Algorithms*. PhD thesis, Carnegie Mellon University, 1979.

[15] R.M. Haralick and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artifitial Inteligence*, 14, 1980.

[16] R.M. Stallman. Forward reasoning and dependency directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 81, 1996.

[17] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1992.

[18] G. Beni and J. Wang. Swarm intelligence in cellular robotic systems. In *Proceedings of NATO Advanced Workshop on Robots and Biological Systems*, 1989.

[19] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From natural to artificial systems*. Oxfor University Press, 1999.

[20] J Kennedy and R.C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[21] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm. Technical report, Manufacturing Engineering Centre, Cardiff University, 2005.

[22] C. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, 1987.

[23] A. Huth and C. Wissel. The simulation of the movement of fish schools. *Journal of Theoretical Biology*, 156(3), 1992.

[24] A.P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley and Sons Ltd., 2005.

[25] E. Bonabeau, G. Theraulaz, and J.-L. Denubourg. Fixed response thresholds and the regulation of division of labour in insect societies. *Bull. Math. Biol.*, 60, 1998.

[26] J.D. Farmer, N.H. Packard, and A. Perelson. The immune system, adaptation, and machine learning. *Physica D*, 22(1-3), 1986.

[27] E. Rashedi, H. Nezamabadi-pour, and S. GSA Saryazdi. A gravitational search algorithm, information sciences. *Information Sciences*, 179(13), 2009.

[28] S.H. Strogatz. *Nonlinear Dynamics and Chaos: With applications to physics, biology, chemistry, and engineering*. Perseus Books, 1994.

[29] F. Heylighen. *The Science of Self-organization and Adaptivity*. The Encyclopedia of Life Support Systems. Inventado, 2002.

[30] I. Prigogine. *The End of Certainty: Time, chaos and the new laws of nature*. Bantam Books, 1984.

[31] F. Heylighen. Classical and non-classical representations in physics. *Cybernetics and Systems*, 21, 1990.

[32] F. Heylighen. *Complexity and Self-organization*. Encyclopedia of Library and Information Sciences. Taylor & Francis, 2008.

[33] C. Gershenson and F. Heylighen. When can we call a system self-organizing? In *Advances in Artificial Life, 7th European Conference*, volume 2801 of *LNAI*. Springer Verlag, 2003.

[34] C Gershenson. *Design and Control of Self-Organizing Systems*. PhD thesis, Vrije Universiteit Brussel, 2007.

[35] J. I. Cano, L. Sánchez, D. Camacho, E. Pulido, and E. Anguiano. Using preferences to solve student-class allocation problem. *Lect. Notes Comput. Sci.*, 5788, 2009.

[36] J. I. Cano, L. Sánchez, D. Camacho, E. Pulido, and E. Anguiano. Allocation of educational resources through happiness maximization. In *Proceedings of the 4th International Conference on Software and Data Technologies*, 2009.

[37] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report*, TR-301, 1984.

[38] E.P. Gilbo. Optimizing airport capacity utilization in air traffic flow management subject to constraints at arrival and departure fixes. *IEEE Transactions on Control Systems Technology*, 5(5), 1997.

[39] B.Y. Choueiry, B. Faltings, and G. Noubir. Abstraction methods for resource allocation. Technical Report TR-94/47, Dèpartement d'informatique, Institut d'informatique fondamentale IIF (Laboratoire d'intelligence artificielle LIA), 1994.

[40] P.J. Modi, H. Jung, W. Shen, M. Tambe, and S. Kulkarni. A dynamic distributed constraint satisfaction approach to resource allocation. *Lecture Notes In Computer Science*, 2239, 2001.

[41] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaître, N. Maudet, J. Padget, S. Phelps, J. A. Rodríguez-Aguilar, and P. Sousa. Issues in multiagent resource allocation. *Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30, 2006.

[42] R. Aumann and S. Hart. *Handbook of Game Theory with economics applications*, volume 2. Elsevier„ 1994.

[43] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 1980.

[44] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan. Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 SwarmFest Workshop*, 2004.

[45] S. F. Railsback, S. L. Lytinen, and S. K. Jackson. Agent-based simulation platforms: review and development recommendations. *Simulation*, 82(9), 2006.

[46] R. Latorre, F. B. Rodríguez, and P. Varona. Neural signatures: Multiple coding in spiking bursting cells. *Biological Cybernetics*, 95(2), 2006.

[47] A. Tristán, F. B. Rodríguez, E. Serrano, and P. Varona. Networks of neurons that emit and recognize signatures. *Neurocomputing*, 58-60, 2004.

[48] R. Latorre, F. B. Rodríguez, and P. Varona. Signature neural networks: Definition and application to jigsaw puzzle solving. Submitted to IEEE Transactions on Neural Neworks.

[49] E.J. Hoffman, J.C Loessi, and R.C. Moore. Constructions for the solution of the m queens problem. *Mathematics Magazine*, 1969.

[50] N. Wirth. Program development by stepwise refinement. *Comm. ACM*, 14(4), 1971.

[51] H.S. Stone and J. M. Stone. Efficient search techniques - an empirical study of the n-queens problem. *IBM J. Res. Develop.*, 31(4), 1987.

[52] R. Sosic and J. Gu. A polynomial time algorithm for the n-queens problem. *SIGART*, 1(3), 1990.

[53] J. Liu, H. Jing, and Y. Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136(1), 2002.

[54] J. Han, J. Liu, and Q. Cai. *From Alife Agents to a Kingdom of N Queens, Intelligent Agent Technology: Systems, Methodologies, and Tools*. The World Scientific Publishing Co. Pte, Ltd., 1999.

[55] M. Bozikovic, M. Golub, and L. Budin. Solving n-queen problem using global parallel genetic algorithm. *EUROCON*, 2, 2003.

[56] W. R. Ashby. *An Introduction to Cybernetics*. Chapman & Hall London, 1956.