CONTRIBUCIÓN AL ESTUDIO DE LA INGENIERÍA INVERSA DE COMPORTAMIENTOS EMERGENTES EN SISTEMAS MULTI-AGENTE

by

María Cristina Parpaglione

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR

Major Subject: Informatics Engineering

at

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Buenos Aires, Argentina September, 2012

CONTRIBUTION TO THE STUDY OF REVERSE ENGINEERING OF EMERGENT BEHAVIORS IN MULTI-AGENT SYSTEMS

by

María Cristina Parpaglione

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR

Major Subject: Informatics Engineering

at

INSTITUTO TECNOLÓGICO DE BUENOS AIRES

Buenos Aires, Argentina September, 2012

Instituto Tecnológico de Buenos Aires

School of Engineering

Department of Informatics Engineering

The undersigned hereby certify that they have examined, and recommend to the Faculty of Graduate Studies for acceptance, the thesis entitled "**Contribution to the study of reverse engineering of emergent behaviors in multi-agent systems**" by **María Cristina Parpaglione** in partial fulfillment of the requirements for the degree of **Doctor**.

Dated: ____________________

Supervisor: ______________________________
Dr. Juan Miguel Santos

Examiners: ______________________________
Dr. Daniel Heymann

____
Dr. Ricardo Rodriguez

____
Dr. Roberto Perazzo

Instituto Tecnológico de Buenos Aires

School of Engineering

DATE: ________________

AUTHOR:	María Cristina Parpaglione
TITLE:	**Contribution to the study of reverse engineering of emergent behaviors in multi-agent systems**
MAJOR SUBJECT:	Informatics Engineering
DEGREE:	Doctor
CONVOCATION:	September, 2012

Signature of Author

To my dad who was, is and

will be my mentor

Table of Contents

Abstract

Each time a problem needs to be solved using a multi-agent system two questions must be answered about the agents involved:

1. What sensing capability must each of them have?
2. Which individual actions must each agent have for solving the problem efficiently?

Answering each of these questions is a difficult matter when solving a problem.

When sensing capability and behaviors are known, the problem to be solved is only one of the possible emergent behaviors of the multi-agent system. The purpose of this thesis is to find a method to discover what the answer to each question is in order to obtain a given emergent behavior. That is, it aims to solve a problem with a group of simple agents with very low communication among them. To achieve this objective, the sensing capability of each agent is modeled by a parameterized function, finding the value of these parameters using an optimization technique such as Genetic Algorithms. After obtaining these values each agent is thoroughly trained using Reinforcement Learning to obtain the appropriate individual behaviors. This thesis proposes a method to obtain both, the sensing capability and the corresponding behavior for each agent in a swarm to reach the wished emergent behavior within the group of pattern formation problems. That is to say, solving this kinds of problems using the *ant-robotic* paradigm.

Resumen

Cada vez que se necesita resolver un problema utilizando un sistema multi-agente se deben responder dos preguntas respecto de los agentes involucrados en el mismo:

1. ¿Qué capacidad de sensado debe tener cada uno de ellos?
2. ¿Qué acciones individuales debe tener cada agente para resolver el problema de manera eficiente?

Responder cada una de estas preguntas es una tarea difícil cuando se está resolviendo un problema.

Cuando se conocen la capacidad de sensado y los comportamientos, el problema a ser resuelto es solamente uno de los posibles comportamientos emergentes del sistema multi-agente en cuestión. El propósito de esta tesis es encontrar un método que permita descubrir cuál es la respuesta a cada una de las preguntas anteriores, en orden de obtener un comportamiento emergente dado. Es decir, poder resolver un problema con un grupo de agentes simples con muy baja comunicación entre ellos. Para alcanzar este objetivo, la capacidad de sensado de cada agente es modelada con una función parametrizable, encontrando el valor de estos parámetros usando una técnica de optimización, tal como Algoritmos Genéticos. Después de obtener estos valores, cada agente es entrenado utilizando Aprendizaje por Refuerzo para obtener los apropiados comportamientos individuales. Esta tesis propone un método para obtener tanto la capacidad de sensado como los comportamientos que debe tener cada agente en un enjambre para alcanzar el comportamiento emergente deseado, dentro del grupo de problemas de formación de patrones. Es decir, resolver esta clase de problema utilizando el paradigma ant-robotic.

Acknowledgements

First of all, I want to thank my advisor, Dr. Juan Miguel Santos, for the time and effort he invested during the development of this thesis.

I am very grateful to the authorities of ITBA and, especially, the head of the Informatics Engineering Department, Lic. Silvia Gomez, for giving me the opportunity and the institutional space to conduct the research that led to this thesis.

I want to give thanks with all my heart to all those who, in one way or another, made the realization of this thesis possible. Because they shared rewarding moments of scientific discussions with me and because they gave me support in times of increased loneliness and distress as well. Without them this thesis would have been impossible.

To my daughter, Ximena, and my nephews, Diego and Catalina, for being there unconditionally.

To Dr. Roberto Perazzo, Dr. Ricardo Sanchez Peña and Dr. Marcelo Frias for giving me not only valuable knowledge but also their unselfish support in times of greatest crisis. All three are an inspiration to me. In particular, I thank Dr. Frias for having facilitated the necessary processing equipment without which carrying out all experiments would have been an impossible task.

To my friends Ana Arias Roig and Ricardo Dokyu for their company, their words of affection, their supportive embrace in the moments when I most needed. Especially, for the moments we shared together all this time.

To Gabriela Feito, my English teacher, without whom the translation of this thesis would have been an arduous task.

To Stella Rouseff, for an excellent ear and the space to express myself.

To Ing. Matías Williams, for his excellent contributions at the time of writing corrections.

To Susana Otero, who made the administrative formalities easy for me.

To my coworkers for their help.

And, in particular, to my cats, Arwen and Frodo, for the company they gave me on long working nights, sleeping next to me on the C.P.Us.

List of Tables

List of Figures

Symbols and Abbreviations

Abbreviations

ANN: Artificial Neural Network

Avg: Average.

ObIndBe: Obtain Individual Behaviors.

ObSenPar: Obtain Sensing Parameters.

PSO: Particle Swarm Optimization

Std: Standard deviation.

Symbols

ε_global: Control when explore or exploit for all configurations.

$\varepsilon\_individual$: Control when explore or exploit a configuration.

ε_{ij}: Control when explore or exploit a configuration for each agent.

α: Learning rate.

γ: Discount factor.

β: Closeness. Sensing parameter value that measures closeness among agents.

δ: Density. Sensing parameter value that measures the density of each layer.

δ_1: Density of the first layer.

δ_r: Density of layers except layer nearest the agent who is sensing.

θ: Aperture. Sensing parameter value that indicates the aperture angle.

ν: Expansion. Sensing parameter value that indicates the sensing area.

ρ: Scope. Sensing parameter value that indicates the sensing radius.

μ: Expected value.

σ: Variance.

S: Set of reachable states.

s: A particular state.

A: Set of possible actions that an agent can perform in a particular state s.

a: A particular action.

Q: Matrix of rewards. It has a reinforcement per action per state.

r: A particular reward.

FR: Reinforcement function.

$Conf$: Number of configurations.

$Limit_{\alpha}$: Ceiling to start decreasing the parameter α.

top_{ε}: Ceiling to start decreasing the parameters $\varepsilon\_individual$.

threshold_config: Determines when a configuration is considered saturated.

threshold_action: Determines when an action is considered saturated.

Q_percentage: Value to consider a pair (s, a) for exploration.

$Q(s_t, a_t)$: Reinforcement matrix.

Chapter 1

Introduction

Driving vehicles in formation [29], enclosing a prey [3, 33, 37], forming rescue teams and patrols [2, 8, 9, 20] or transporting heavy elements [15, 17], are all problems that require joint work. Some of these problems are typically distributed and others can be solved in a distributed way, although that is not its essence. An example of this is the problem of transporting heavy objects. This problem can be solved using a specialized single agent or with a swarm (group) of simple agents that are unable to perform the task individually but reach the goal due to the interaction among them. When a single agent is used, if it fails for any reason, the job can not be performed. However, if the problem is solved with a swarm of simple autonomous agents with low communication among them, the task can always be solved even if a few agents fail to do the job. Task achievement is due to two reasons: the number of agents exceeds the requirement of the task or it is possible to add agents because of low-cost.

When a swarm of agents is used it is essential to answer the following questions:

1. Can agents act in an autonomous way or do they need to be coordinated in a central way?

2. Must all agents have the same capabilities or must different types of agents be used to solve the problem?

3. Must each agent know how to solve the whole problem or may the problem be solved by the interaction of many or all of the agents in the swarm?

In the first question when a central coordination is implemented, if it fails, it is impossible to solve the problem. But, when the problem is accomplished by a swarm of autonomous and redundant agents, even if some of them fail, the whole task can be completed successfully. This may be possible because either the agents are enough to solve the whole problem, or the failed agents can be easily replaced by new ones. This is known as *fault-tolerance*. Fault-tolerance is necessary to minimize the following problems:

1. The cost involved when it is impossible to complete the task, and
2. The cost of building self-sufficient fully specialized agents.

The second question addresses agents capabilities. It is necessary to define what kinds of actions agents will be able to perform. These actions depend on the stimulus received and it is named *behavior* or *policy*. When all agents act in the same way in response to the same stimulus, it is said that the agents are *homogeneous*. Furthermore, if each agent or subgroups of agents may behave differently from one another, it is said they form a swarm of *heterogeneous* agents. For example, if the problem consists in forming a cluster of agents without any special geometry in an enclosed space in any place, the problem can be solved with a group of homogeneous agents (see Section 5.1). However, if the problem is to create a formation (see Sections 5.2:5.3.2), as rescue groups or patrols, at least two different types of behavior will be needed.

The third question is answered by designing a multi-agent system that complies with the *ant-robotic* paradigm [11], which emphasizes the virtues of massive parallelism of many simple agents (as in the case of ants in nature) with limited sensing capability and low precessing capacity, but with the ability to interact with the environment in which they act. These systems must include the following characteristics:

- Each agent has incomplete information or a limited number of capabilities for solving the problem and a limited view,
- There is no overall control system,
- The data is decentralized, and
- The processing is asynchronous.

All these points are taken into account in this thesis.

The term *ant-robotic* is often used to denote distributed robot systems that use indirect interaction strategies through the environment.

This thesis proposes a method to obtain both, the sensing capability and the corresponding behavior of agents to solve pattern formation problems. In other words, solving this kinds of problems using the *ant-robotic* paradigm.

Research within swarm robotics includes self-organization. An interesting sub-problem of self-organization is pattern formation [5]. The term pattern formation in literature is used in at least two different ways. Firstly, to define an area of study within multi-robot systems that covers distinct aspects of patterns such as the establishment, maintenance and reconfiguration of patterns. Secondly, to report the natural phenomenon of flocking whereby loose or deformed geometric patterns emerge, and not necessarily strict geometric patterns [1]. In this thesis, the first usage of the term is adopted. That particular formation to be achieved is an emergent behavior of the multi-agent system.

Before starting the description of the thesis itself, it is necessary to clarify some terms. There is a fundamental difference between "*training*" and "*learning*". According to Merriam-Webster Encyclopedia Britannica, "*training*" is defined as: "the skill, knowledge, or experience acquired by one that trains." While "*learning*" is defined as: "Knowledge or skill acquired by instruction or study.". Notwithstanding, "training agents to solve a problem" or "allowing agents to learn how to solve a problem" will be used interchangeably, as both influence the behavior of the agent.

The scope of the experiments is restricted to pattern formation, both with or without specific geometric shapes, within finite-dimensional discrete toroid.

These problems could be used to enclose a prey, build rescue patrols or set special formations of agents. In particular, experiments are performed on a discrete toroid, whose dimension is set at the beginning, and with a focus on two types of problems:

1. Grouping or clustering agents anywhere in the toroid.

2. Obtaining a specific shape anywhere in the toroid (i.e. square, triangular or diagonal).

The first formation is useful to prove that it is not necessary to train all agents involved in the multi-agent system but only a minimum number, and then, it is possible to replicate the sensing capability and behaviors learned by these agents to the rest of the agents in the system.

The second part of the experiments, involving geometric shapes, is carried out to verify that differentiation of behaviors is necessary among the agents involved.

The complexity of the problem is given by several constraints, namely:

- The environment is sensed partially.
- Performance measure is global (i.e., it is the emergent behavior to achieve).
- Agents operate independently.
- They have low communication among them. In general, the communication is given by the change in the environment.
- They move in random order. The movement of the agents is not controlled by any central agent, and they can all move simultaneously in different random sequences.

Learning is accomplished applying the Q-learning technique [27]. To achieve this it is necessary to determine what sensing capability each agent or group of agents need to reach the goal. To this effect a five parameter sensing function is defined (see Section 4.1). Genetic Algorithms [13, 10] are used to find the best settings for these parameters in each problem. The fitness function used is the performance achieved by the swarm with each set of parameters encoded in the chromosome (see Section 4.3). If the required behavior is homogeneous, then all the agents can use the same sensing capability; but if the task requires heterogeneous agents, then different sensing parameters are needed to solve the problem.

Agents used here are autonomous without any clock or central coordination to determine the order in which they move. Furthermore, they are inexpensive and with rudimentary movements (they can move forward one position, rotate 90° clockwise or rotate 90° counterclockwise, see Chapter 5). An agent is considered inexpensive if it can be built with simple low-cost sensors and it is computationally cheap. For example, *kilobots* [25] are a US$14 micro-robot and it takes 5 minutes to assemble. This a low-cost robot designed to test algorithms for groups of hundreds or thousands of them.

From now on, the terms agent and robot will be used interchangeably.

When an agent decides to perform an action, the environment changes for all agents involved in the multi-agent system. Therefore, the agent must sense the environment again the next time it decides to act. This process is repeated by each agent in a random manner, causing the system to be purely stochastic. During the learning process, agents are based on utility. A utility-base agent is one that is able to decide which sequences of actions can lead to a faster, safer or cheaper completion of the task, depending on the chosen criteria. After learning each agent behaves as a simple reactive agent [26].

Es interesante reformular el objetivo de la tesis de la siguiente manera: se quiere encontrar una capacidad de sensado para los agentes involucrados en el sistema multi-agente que permita resolver el problema de formacin de patrones, de manera tal que tenga un bajo nmero de configuraciones posibles pero que a su vez minimice el aliasing perceptual conflictivo y que permita encontrar los comportamientos de los agentes de manera tal de resolver algn problema de formacin de patrones en un toroide discreto de dimensiones finitas que sea un comportamiento emergente del sistema, de manera eficiente utilizando el paradigma ant-robotic.

Es necesario destacar que esta tesis no es una simple aplicacin de Aprendizaje por Refuerzo y Algoritmos Genticos.

La contribucin realizada es la posibildad de parametrizar la capacidad de sensado de manera tal de encontrar los valores de forma automtica y con una cantidad de posibles necesarias para los agentes en el sistema con, la caracterizacin del aliasing perceptual conflictivo y la modificacin de la tcnica Q-learning introduciendo exploracin dirigida de manera tal que permita tratar con aliasing perceptual presente cuando la capacidad de sensado es minimal, tan conflictivo a la hora de resolver un problema.

It should be pointed out that this thesis is not a simple application of reinforcement learning and genetic algorithms; in fact, the development is based on these algorithms because they are known to work optimally in the resolution of a wide variety of problems. What is new is the parameterization of sensing capability (see Section 4.1.1), the characterization of conflictive perceptual aliasing (see Section 4.1) and the modification introduced in how Q-learning technique explores the space of possible actions. The modification introduced, named *action directed exploration* (see Section 4.2.2.1), allows treating the conflictive perceptual aliasing, which is very problematic when solving a problem, that appears when a minimal sensing capability is used.

The rest of the thesis is structured as follows, Chapter 2 presents previous studies. Chapter 3 defines concepts used here. Chapter 4 is devoted to theoretical explanation of this thesis together with the introduction of the proposed methods. Chapter 5 introduces new key concepts related to agent behaviors (essential for a complete understanding of the thesis), as well as the experiments and results. Finally, Chapter 6 presents conclusions and future research.

Chapter 2

Background

Following, some research focused on the discovery of different emergent behavior of collections of agents will be reviewed.

It has been impossible to find sensing characterization in the sense presented in this thesis. In general, work on multi-agent systems is based on the discovery of emergent behavior with groups of agents whose ability to sense and / or act is known in advance.

In this sense, Panait and Tuyls [21] present the dynamics of multiple agents trained with Reinforcement Learning from the perspective of evolutionary game theory. The authors introduced the concept of lenient agents "*who ignore low rewards due to actions chosen by teammates that are poor matches to the agent's current action*". In particular, lenience is important at the beginning of the game, where agents have not yet identified their best moves. To allow agents to disregard poor actions of other agents, it is necessary to have enough information about them, including the actions they are taking. Agents use Boltzmann exploration to pick actions, which control probabilities with a temperature variable. The authors demonstrated that straightforward extensions of Q-learning to multi-agent systems fail to reach the optimal policy in fairly simple domains.

In this thesis, agents learn how to proceed in each situation using an algorithm based on a straightforward application of Q-learning for a single agent. The proposed algorithm (see Chapter 4) has many differences with respect to the original Q-learning algorithm. One of these differences is the use of a reinforcement matrix for each agent and not a joint matrix. Agents have very low communication among them and therefore cannot base their decisions on actions taken by other agents. Instead, agents are allowed to explore low-visited states, using *action directed exploration* (see Section 4.2.2.1), even when the learning is in an advanced stage. This would avoid premature convergence and is a way to implement lenience indirectly, i.e. without knowing the actions taken by other agents. The results shown in Chapter 5 confirm that it is possible to use an algorithm based on the direct application of Q-learning for a single agent to train a group of agents in a multi-agent system.

Berezhnoy [4] presents a multi-agent environment form with two different groups of agents in a virtual two dimensional plane. The objective is to determine if any behavior that emerges from the interaction among the agents exists. The multi-agent system proposed is a well-known game named "Warmers and Chillers". In this game each cell has two values, the normal temperature and the local temperature. There are two types of agents:

- Agents which always try to reach areas of the board with higher temperature to heat the atmosphere by generating heat. (Warmer agents)
- Agents which try to go to cold places, lowering the temperature of the surroundings. (Chiller agents)

Agents move around the board raising or lowering the temperature of the environment around them. The sensing capability of each agent is inversely proportional to an agent's degree of satisfaction. An agent is satisfied when it reaches its objective; that is to say, a warmer agent is more satisfied if he can raise the temperature around him and vice versa. There is no central control; agents act and move in random order on the board. The grouping of two distinguishable types of agents was the emergent behavior achieved (i.e. warmer agents together and separately, the group of chiller agents). That is, the emergent behavior is achieved by the interaction of agents whose nature is known in advanced.

In this thesis, the aim is to discover what type of agent should form the multi-agent system to achieve a desired emergent behavior, such as making a cluster (see Section 5.1). The grouping problem is one of the possible emergent behaviors sought (see Sections 5.2 and 5.3). Agents are trained to obtained not only the behavior but also the required sensing capability in order to reach the formation desired. A minimum number of agents is trained replicating the knowledge learned to the rest of the agents in the multi-agent system.

Quinn et al. [24] present homogeneous agents controlled by an Artificial Neural Network (ANN), where the sought emergent behavior is grouping moving robots;

namely, a flock. It is important to highlight that they only use three agents in their experiments. The ANN architecture and parameters are found using Evolutionary Algorithms. All the robots use the same ANN, which runs in a central computer. They have infrared sensors allowing them to detect proximity of other agents. The robots start in random positions with one condition: each robot must detect at least one of the others. There are different tasks needed to keep the flock of three agents. In particular, the agent who is leading the flock has to walk backwards, while the other two have to keep watching the head of the flock. ANN is trained to perform any of these subtasks. As all agents use this common network and are indistinguishable, although they do not perform all the subtasks for which they would be trained, they are considered homogeneous agents anyhow. With these characteristics in mind it is easy to explain fault tolerance.

In the case of the thesis presented here it is possible to work with both homogeneous and heterogeneous agents. If the agents are heterogeneous the flock can be achieved by having a moving distinguishable agent where the desired emergent behavior consists in reaching it; as happens when a prey is enclosed. If enclosing is impossible, the prey acts as a distinguishable agent making the swarm be in continuous movement. When the prey is removed, the agents meet somewhere in the board and remain there forming a cluster. There is no limit to the number of agents involved. The behavior and sensing capability are learned by each agent. Chapter 6 shows that fault tolerance can be overcome by using both homogeneous and heterogeneous agents. In addition, there is no restriction on the pattern to be reached, it can be geometric or not.

Peter Wavish [32] focuses his research on answering how to exploit the emergent behavior in the design of multi-agent systems. The method was devoted to design individual agents maintaining the symbolic representations of the emergent behavior, which can then be used as a basis for building higher-level behaviors. In his work the designer has an important role during the construction of the symbolic behaviors. The solution presented in his paper is to create and maintain explicit symbolic representations of the emergent behaviors by coupling them to other symbolic behaviors within the agent. He stands that, while most of the work on Artificial Intelligence is based on the explicit representation of knowledge, his work is based on the explicit representation of behavior. He argues that the activity of an agent must be given by its interaction with the environment and not by a process of reasoning occurring only in the agent. To deal with this he designed two programming language, ABLE (Agent Behavior LanguagE) and RTA (Real Time ABLE), both based on PROLOG. In RTA the set of possible behaviors is finite and is determined at compile time. Agents are represented by the RTA compilers asynchronous digital logic circuits, where behaviors are implemented as registers, logical operators as logic gates, and delays as monostables. To test the programming language he simulate a world with six robots. In this world one of the robots (the dog) has to herd the remaining five robots (the sheep) towards a movable shepherd. The world also contains stones that can be seen as obstacles. Each robot can move in any of eight directions, and has two kind of sensing, corresponding to sight and touch. Both types of sensing are divided into eight sectors. In each sector, the agent senses either the presence or the absence of an object. In the case of sight, what the robot can distinguish can be predetermined. The only robot that constructs its behavior is the dog, the remaining agents have their behavior preset. Eventually, the dog can distinguish whether he is trapped between rocks, or surrounding the sheep to guide them to the stable. With this to types of symbolic behavior Wavish states that more complex behavior can be construct. There is no proof with more than one agent.

The main difference with the thesis presented here is that in this case the agents are trained to achieve the necessary performance to meet the desired emergent behavior. The behavior obtained is neither a symbolic representation nor an internal representation. The agent senses the environment and acts based on what it believes. Moreover, the sensing capability of each agent involved in the multi-agent systems is learned using the proposed algorithms (see Chapter 4).

The works of Suzuki and Yamashita [28] and Gordon et al. [12], was reviewed together because the work of Gordon et al. is based on Suzuki and Yamashita's work. Suzuki and Yamashita have investigated a number of geometric pattern formations in a plane with a multi-agent system. They present algorithms to group agents in a single point in a finite number of steps. All agents execute the same algorithm and they are indistinguishable. They move at random sequences of time. Each agent is a mobile processor with infinite memory and a sensor for detecting the positions of other agents. They can be either active or inactive, unpredictably. They are allowed to occupy the same physical space, avoiding the problem of handling collisions. Each agent, with its infinite range of visibility, must observe the remaining agents in the system in order to solve the problem. If the algorithm used is oblivious, the new position is determined only by the observed positions of the remaining agents at that time. Otherwise, the algorithm is non-oblivious, and the new position may also depend on the observations made in the past. The authors claim that the agents can converge to a certain geometric pattern if all agents handle the same coordinate system. If not, the task is impossible to complete. They show that the formation problem for a point can be solved by a non-oblivious algorithm with two or more agents, but when the algorithm is oblivious they need three or more agents. They state that the problem with two agents and a oblivious algorithm is unsolvable, arguing that it is due to the stochastic environment given by the asynchronism of the agent's movement. The other problem presented is that agents cannot break the symmetry that exists in their initial distribution. The only solvable formations are: a point and a regular n-gon. Only deterministic algorithms are used to determine where agents need to move. They uphold that it is impossible to use a Markovian algorithm to make two agents go to the same point.

Gordon et al., [12], investigate the problem of gathering a swarm of multiple robotic agents on the plane using very limited local sensing capabilities. The gathering problem is defined as the problem of getting agents on the plane into a point or small region. The problem is under the ant-robotic paradigm so agents are anonymous, homogeneous, with low communication and limited sensing capabilities. They use real robots made from LEGO parts and very simple sensors not providing distance measurements. The proposed algorithm always converges to a small dense cluster while retaining mutual visibility among agents. The world consists of the infinite plane with n agents living in it. At discrete time steps each agent may be either active or inactive, in the same sense proposed in [28]. Agents do not have any control over the random scheduling of its activity times. When they are active, they sense the environment, process some calculations and optionally they move to another point into the plane within a defined distance. They have a predefined distance to detect other agents. They are allowed to occupy the same physical space, avoiding the problem of handling collisions. Contrary to the work of Suzuki and Yamashita [28], Gordon et al. guarantee asynchronicity among agents arguing that this allows agents to achieve asymmetrical shapes.

With respect to the thesis presented here, the homogeneity of agents depends heavily on the problem to be solved. Whereas, building cluster assemblies (see Section 5.1) does not need a behavior differentiation, building square, triangular or diagonal shapes (see Sections 5.2 and 5.3) does. In the former, homogeneous agents can be used; but in the latter, it is necessary to distinguish at least two different types of behaviors, so the agents are heterogeneous and they learn the necessary behaviors to complete the task using the proposed algorithm in this thesis (see Chapter 4). These learned behaviors can be used to solve any problem whose objective is to achieve the same grouping pattern used during training without retraining. Because it is possible to learn homogeneous and heterogeneous behaviors there is no restriction with the targeted geometric pattern formation, it can be symmetrical or asymmetrical. It is also possible to learn a stochastic behavior to achieve the desired goal and there is no limit in the minimum number of agents needed to reach the objective. Even though the formation needs n agents only a minimum number of them require training. Once trained, the learned knowledge will be replicated in the remaining agents that are needed to achieve the pattern. In this work, only one agent can be in one position. In order to make the solution more realistic, superposition is not allowed. In the scope of this thesis, the learning process is in fact reached with a Markovian algorithm [27] in a finite number of steps, with both homogeneous and heterogeneous agents, further explained in Chapter 5.

In the work of Huaxing Xu et al., [36], a solution to a pattern formation problem inside a grid is presented. Agents involved in the multi-agent system are homogeneous, cooperative, indistinguishable and non positional. They have two working modes: they can explore the grid or disperse in it. A pheromone mechanism is used to message communication and coordination among agents. Initially, the grid is subdivided into sub-areas with some cells marked as part of the goal. The agents are uniformly assigned within each sub-areas; namely, all the sub-areas have agents within them. During the exploration mode, agents explore the sub-areas where they have been assigned to distinguish the presence of any marked cells, and identify if any of them are free; i.e., marked cells without agents on them. When all cells are occupied by agents or there are no marked cells in that sub-area, the agent communicates the situation to the other agents via pheromones, and uses PSO to detect the closest sub-area with free marked cells. Once the agent knows the sub-area with free marked cells it starts working in dispersion mode to find the marked grid where to stay, in order to achieved the general goal.

In this thesis a minimum amount of agents in the multi-agent system go through a learning stage. After this stage it is possible to:

- Replicate the knowledge learned by agents to the rest of the agents in the multi-agent system.
- Achieve the wished pattern anywhere within the discrete toroid regardless of its size.
- Train homogeneous and/or heterogeneous agents.
- Obtain both kinds of policies, deterministic or stochastic.

Agents are indistinguishable and positional. They must occupied a specific place within the pattern. For example, if the triangle shape is considered a pyramid, the agent which is at the top should be someone experienced and with a specific body type. If the same shape is considered an attacking formation, the agent at the top position should be someone who knows how to manage the group. Genetic algorithms are used during the learning stage to obtain the best sensing capability for each type of agents involved in the multi-agent system to solve the proposed problem.

Dimarogonas and Kyriakopoulos, [7], are interested in decentralized planning, specifically in decentralized conflict resolution in air traffic management and the field of micro robotics, where a team of autonomous micro robots must cooperate to achieve manipulation precision in the sub micron level. Decentralization is given, primarily, in that each agent knows only his own destiny, but not the destiny of the remaining agents. The workspace is bounded and spherical. The sensing capabilities of each agent are limited to a circle of specified radius around it, so only spherical agents are considered. Consequently, each agent knows the position and/or the velocity of every agent within its sensing zone at each time instant. Furthermore, each agent only knows its own desire destination; but they all know the exact number of agents in the whole system. Collisions among agents are avoided.

The major difference between this work and the thesis proposed here is that agents learn how to sense to solve the problem at hand. This sensing capability and the necessary behaviors can be replicated in the rest of the agents in the system and use to solve similar problems without any need for retraining.

As stated earlier and unlike all the algorithms mentioned above, the purpose of the methods presented here is to find individual behavior and sensing capability of each agent involved in the multi-agent system that allows them to solve a predefined emergent behavior. There are several definitions of emergent behavior, the one which comes closest to our thought is defined in [16]: "*It is essentially any behavior of a system that is not owned by any of the components of it and emerges due to interactions between them.*" Any problem can be considered as an emergent behavior of a multi-agent system, in particular, problems like geometric pattern formation treated in this thesis.

Chapter 3

Preliminary Knowledge

3.1 Definitions

Any problem can be considered as an emergent behavior of a multi-agent system, in particular the geometric pattern formation problems. To be able to solve this type of problems is necessary to know how the agents involved in the multi-agent system should be in order to reach the goal.

Once the problem to be solved is known, that is, the emergent behavior of the multi-agent system is identified, the agents involved in the system must learn how to behave in the environment. The way to obtain individual behavior of a group of agents to achieve a goal is called *reverse engineering*.

From now on, the state of the agent after sensing the environment is called *configuration*. A configuration is the agent's belief about the world, from its own point of view. This view of the agent depends on the number and type of sensors with which it is equipped. Computationally, the configuration is the representation of the sensed state.

It should be clarified that a sensing capability with a low number of possible configurations, allows using inexpensive simple agents. When an agent decides to perform an action the configuration to be achieved by the movement may differ from the configuration at the moment of deciding the next action, due to the randomness of the environment arising from the other agents' actions. As a remainder, agents are autonomous, without any clock or central coordination to determine the order in which they move, and the actions performed by each one of them modify the environment. Therefore, it is necessary that the agent senses the environment again to pick an action at that moment. As a result, a new configuration is obtained

In this work two methods are presented to find simultaneously what the sensing capability should be and what the behaviors should be for each agent involved in the system. From now on, the sensing capability and the behaviors learned by the agents using the proposed methods will be named *learned knowledge*.

3.1.1 Introduction to Reinforcement Learning

The algorithm of reinforcement learning used to train agents is Q-learning (see [31]). Q-learning for multi-agent systems case has been implemented as a direct extension of Q-learning for single agents. It should be considered that, while learning is done independently for each agent, the information observed by each individual of the population depends on actions taken by the remaining robots.

Given a multi-agent system where S is the set of reachable states for each agent and A is the set of possible actions that each agent can perform, Q-learning associates an utility Q with each pair (s, a), where s is a state of the environment and a is an action that the agent can take in that state. At each step, the agent receives a reward $r \in \mathbb{R}$. The objective of the agent is to maximize the discounted sum of future rewards [27]. The reinforcement role is to increase or decrease the corresponding value in the matrix Q, depending on whether the agents reached or not the target.

Each iteration agents update the values of Q based on the reinforcement that has received and the maximum Q value for the successor state, according to Eq. 3.1.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{3.1}$$

where α is the learning rate and γ is the discount factor.

Parameter α is initially set to 0.1 and is decreased linearly reaching 0 when the learning process ended. The number of iterations needed to start decreasing α is set into the variable $Limit_\alpha$ at the beginning of the training process. Two phases can be distinguished in most of the searching algorithms: exploration of possible actions and exploitation of that actions that appear to promise a good solution to the problem. As initially nothing is known about the solution of the problem, exploration allows a global search into the possible actions to test them. Once the learning process is advanced it is useful to reaffirm the best results so far. It is difficult to know how long to explore or exploit actions. Q-learning distinguishes these phases using $\varepsilon-greedy$. The phases of exploration/exploitation are determined by the parameter ε. This parameter declines with the iteration number. The smaller the value of ε, the bigger is the probability to exploit good actions. Both parameters, α and ε, are related. In general, the parameter α starts declining when the parameter ε reaches 0. It is important to notice that the algorithm proposed here uses *action directed exploration* (see Section 4.2.2.1), so some actions can be explored even when the training process is near to the end.

The aim is to find a policy (behavior) for each agent. These policies can be obtained from Q using Eq. 3.2.

$$\Pi(s) = arg \max_{a} Q^{\Pi}(s, a). \tag{3.2}$$

The rewards are obtained with a global reinforcement function, FR, involving the performance of the whole group. As it was appointed before the environment is stochastic, so

$$P^{a}_{ss'} = Prob\{s_{t+1} = s'/s = s_t, a = a_t\}, \tag{3.3}$$

where s is the current state, a the action to be taken for the agent and s' the new state after execute the action a. That explains why the policies obtained through different training process can associate different actions to the same state.

The Q-learning technique associates exploration and exploitation with actions and in turn every action leads to at least one state (see Section 4.1). Therefore from here onwards actions and states will be used interchangeably when dealing with exploration and/or exploitation.

3.1.2 Policies

Policies , Π, obtained by Q-learning, can be either:

1. Stochastic, or

2. Deterministic.

When the policy is Stochastic, each action for each configuration has a probability of being selected for execution. That is to say, the policy Π is Stochastic if $0 \leq \Pi(s, a_i) \leq 1$ and, $\sum_{i=1,N} \Pi(s, a_i) = 1$, where $\Pi(s, a_i) = Pr(a_t = a_i/s_t = s)$.

It is calculated as a proportion of the Q values for each configuration and each action. The Q value for an action a_i in a configuration c is calculated according to Eq. 3.4. This Q value is used to associate a probability of selection an action with each individual configuration.

$$prob(c, a_i) = \frac{Q(c, a_i)}{\sum_{j=1}^{\#actions} Q(c, a_j)} \tag{3.4}$$

Instead, the Deterministic Policy unambiguously establishes the action that an agent should apply for each possible configuration. In short, if A is the set of actions for an agent and $a_i \in A, \quad \forall i$ such that $1 \leq i \leq N$, where N is the number of possible actions then, the policy Π is deterministic if $\Pi(s, a_i) = 1$ y $\Pi(s, a_j) = 0, \quad \forall j \neq i$ that is to say, $\Pi(s) = a_i$. It is calculated as the action that produce the maximum value of Q for each configuration, following Eq. 3.5.

$$action = arg \max_a Q(s, a). \tag{3.5}$$

When the problem to be solved is grouping agents anywhere in a toroid without any predetermined shape, it is possible to obtain a Deterministic Policy that allows agents to reach the goal. It is because there are many possible solutions to this problem and no special one is required. The enormous difference between a Deterministic Policy and a Random Policy is that the first has knowledge about the problem that the second one does not, allowing the solution to be reached with better effectiveness and lower number of steps (see Chapter 5). This is not the case when a predetermined shape is required, because there is only one solution to the problem. In the grouping problem, without any predetermined shape, agents do not need differentiation of behavior. When more than one agent is trained, the learned knowledge of each can be considered as a different point of view of only one agent. Therefore, a homogeneous policy can be obtained combining the learned knowledge by each agent in each training process. There are different ways to obtain these policies, all of these will be explained in Chapter 5.

3.1.3 Introduction to Genetic Algorithms

A Genetic Algorithm is an heuristic search that mimics the process of natural evolution, allowing finding useful solutions to optimization problems using inheritance, selection, crossover and mutation.

The algorithm was proposed by John Holland [13, 10]. Potential solutions, named *individuals*, are coded into a *chromosome*. Each chromosome is composed by *genes*, which can be simple (e.g. a bit) or complex (e.g. a structure). Each individual has assigned a *fitness value*, depending on the problem to be solved. This value is a measure of how good, or apt, the solution coded into the chromosome is.

Given an initial set of random individuals, named *initial population*, a group of them are selected with some criteria. There are several ways to select and/or replace individuals in the population. The way applied here, is to choose a percentage of the more apt individuals in the population and to choose the rest randomly based on their fitness value. Random choice allows the introduction of diversity, avoiding premature convergence. These individuals are crossed to obtain the offspring. These offspring can suffer mutation of some of their genes. The individuals obtained replace the same number of individuals in the original population, creating the new generation. This process is repeated until a termination condition is found. Some of these termination conditions are the following:

- The solution is found.
- The maximum number of generations is reached.
- Population remains the same from generation to generation.
- Manual inspection of the population is conducted.

Evolution usually starts form a random initial population where each individual represents a potential solution of the problem to be solved. Each population evolves using genetic operators form one generation to another. The basic genetic operators are the following:

- *Selection*: Allows choosing which individuals will be involved in the generation of the offspring. There many ways to do this, in the work presented here a combination of the following will be used:

  - *Elitist*: Allows some of the best individuals from the current generation to be carried over to the next generation, based on their fitness value.
  - *Roulette*: Chooses the individuals taking into account their fitness level. The fitness level is used to associate a probability of selection to each individual. If f_i is the fitness of individual i in the population, its probability of being selected is

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

    where N is the number of individuals in the population.

- *Crossover*: Two individuals are combined to generate the offspring. The combination of apt individuals has a great probability to generate apt individuals. In fact, the crossover operator allows exploring locally the space of potential solutions.
- *Mutation*: Allows randomly changing genes in the chromosome. This operator allows a global exploration of the space of potential solutions, adding diversity to the population.
- *Replacement*: Less apt individuals from the population are replaced by the offspring.

In each new generation the fitness value of each new individual is calculated.

Chapter 4

ObIndBe and ObSenPar: Proposed Methods

In this chapter two methods are introduced: the algorithm that allows obtaining individual behaviors for a group of agents trying to complete a task (ObIndBe) and the algorithm that enables acquiring the sensing capability for this purpose (ObSenPar).

In order to explain some of the concepts set out below, graphics like Fig. 4.1 and 4.2 will be used. Both figures show a 7×7 discrete toroid representing a possible state in which the agents could be, either during the learning or the evolution of the system. Each agent has a number that identifies it. The darker area in the square, with the number, represents the agent's face and indicates the direction where it is facing. Fig. 4.2 shows a possible sensing area for Agent 1. This Agent uses four sensors: front, right, back and left. The sensing area is represented by the black thick lines whose center is the agent who is sensing. For example, Agent 1 sees Agent 2 with the front sensor. Agent 7 is inside of the right sensor sensing area and Agent 4 is inside of the back sensor sensing area. There are no agents inside the left sensor sensing area.

4.1 Sensing Capability

Finding the correct sensing capability to solve a problem using multi-agent systems is a difficulty in itself. The impossibility to solve a problem may arise from the agents having an incorrect sensing capability and, consequently, they could require more steps to solve it than the number of steps needed when the appropriate sensing capability is used, or not solve the problem at all. For example, solving a

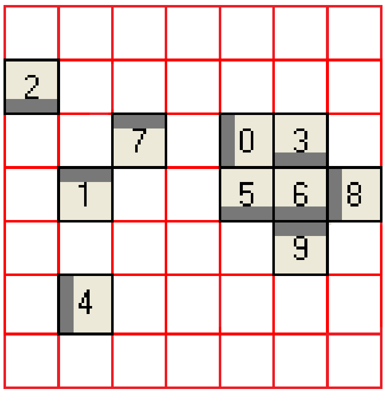


Figure 4.1: *Scenario with ten agents, distributed randomly in a 7x7 discrete toroid.*

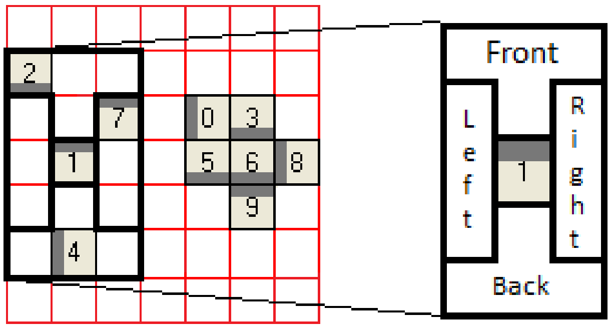


Figure 4.2: *Sensing area for Agent 1.*

problem using blind agents is the same as finding a solution using random walk. Blind agents will need more steps than agents that can see one position in front of them. But, perhaps, these last agents will encounter complicated situations that can be avoided by having correct sensing. It is also true that incorrect sensing can lead to situations where the agents remain stuck and are unable to reach the solution. In Chapter 5 a comparison between the policies obtained with the proposed methods and a Random Policy is shown.

One way to find the correct sensing capability is to solve the problem manually. Namely, determining the different sensing areas, learning the individual behaviors for each agent, analyzing the problems that could not be solved, and making the necessary changes to repair the problems. This was done in [22] where the conclusion was that finding the best sensing capability by trial and error is extremely tedious and time consuming. In Chapter 5 a detailed explanation on this issue is presented.

Because the agents are simple and with a bounded sensing capability, a problem named perceptual aliasing arises. Perceptual aliasing is the incapacity to distinguish states of the world because the same configuration is used (see Chapter. 3) to represent them. Given a behavior, it is possible to differentiate two groups of states in the sense of perceptual aliasing:

1. Those that need the same action.

2. Those that need different actions.

The first group is interesting because the space state is reduced in a considerable way. The second group presents a problem that is difficult to solve. For example, if the previous state to reach the goal is confused with another state of the world, far away from the objective, and both need different actions, it is very likely that the problem will have no solution. This situation is named *conflictive perceptual aliasing*. As in [19], perceptual aliasing is considered "a blessing and a curse". Let's suppose that the multi-agent system needs to reach the formation shown in Fig. 4.3. Fig. 4.4(a) shows a previous state to reach the goal and the sensing area for Agent 0. If Agent 0 moves one position to the front, it reaches the goal (Fig. 4.3). Instead, in Fig. 4.4(b), although the Agent 0 is far away from the goal, the sensing value is the same as Fig. 4.4(a). In this case, if the policy used is deterministic and the action is "go one step forward", Agent 0 reaches the position shown in Fig. 4.5, far away from the goal. In this case, the policy needs to be stochastic to allow the agent to reach the solution.

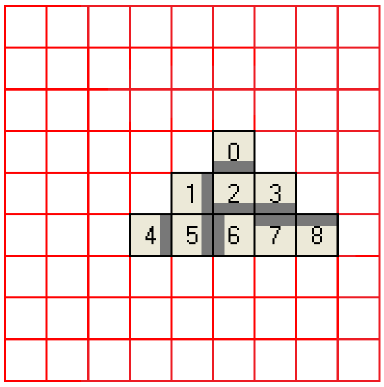
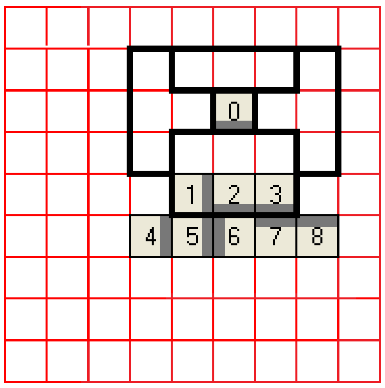
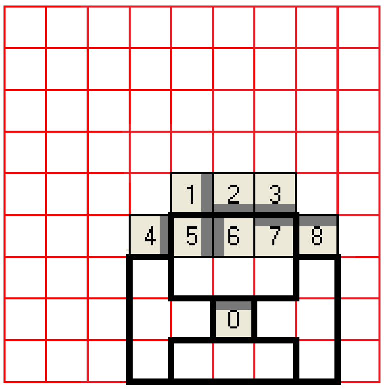


Figure 4.3: *Triangular shape in an* 9 × 9 *toroid with nine agents.*

When conflictive perceptual aliasing is presented it is impossible to learn a Deterministic Policy. This can be seen in [19, 14, 6] and mainly in [35]: *"Perceptual aliasing interferes with the decision system's ability to learn the optimal policy"*. There is a relationship of compromise between finding the best sensing capability and using a Stochastic Policy. It must be remembered that a sensing capability



(a) *Agent 0 one step from the goal.*



(b) *Agent 0 far away to reach the goal but with the same configuration as in Fig 4.4a.*

Figure 4.4: *Both figures show a* 9×9 *discrete toroid with nine agents and the sensing area of Agent 0. The goal is shown in Fig. 4.3. Fig 4.4a shows Agent 0 one step to reach the goal. In Fig. 4.4b Agent 0 is far away from the goal.*

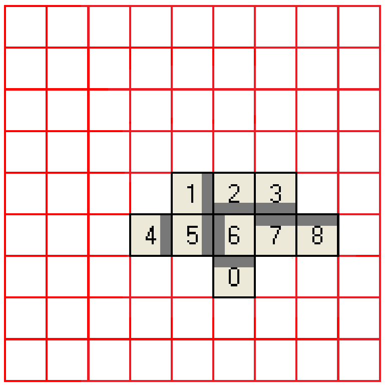


Figure 4.5: *State reached for Agent 0 after moving forward one position starting in the situation shown in Fig. 4.4(b)*

with a low number of possible configurations is desired. It is important to have in mind that, the lower the number of configurations, the higher the perceptual aliasing; and the higher the perceptual aliasing, the more likely the conflictive perceptual aliasing. In cases of higher conflictive perceptual aliasing, the best results are obtained using Stochastic Policies against Deterministic ones, but the number of steps that the agent needs to reach the goal, increases considerably.

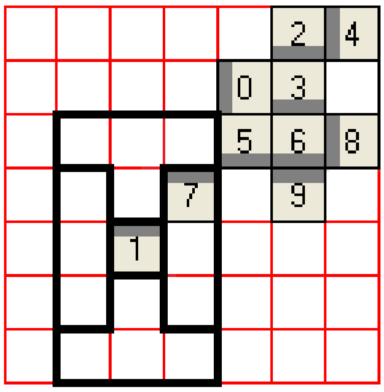
A question to be posed is why the sensing capability must change if it is always possible to define a Stochastic Policy? In general, solving a problem using a Stochastic Policy requires a greater number of steps to reach the goal. Therefore, in cases in which it is possible to find a Deterministic Policy, this should be preferred instead of a Stochastic one.

When sensing capability presents a strong conflictive perceptual aliasing, a Stochastic Policy would be necessary to allow agents to perform actions that enable them to overcome problematic world states. Instead, if sensing capability has slightly conflictive perceptual aliasing, the number of steps needed to reach the goal decreases considerably (see Chapter 5). There is a trade-off between finding the best sensing capability, which is completely unknown, and using a Stochastic Policy. Moreover, in a stochastic environment, where conflictive perceptual aliasing is present, it is very difficult to reach the solution with a Deterministic Policy.

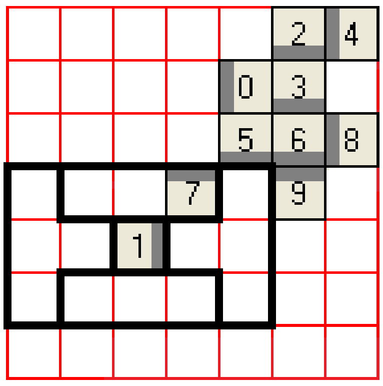
Several problems can appear when the sensing capability is not the appropriate. Some of them are:

Symmetries: When an agent goes from one state to another doing opposite actions without being able to break the cycle for itself. Two situations must be considered with respect to the policies used. If the policy is stochastic, this situation cannot last because all possible actions in that configuration have a non-zero probability of being executed, so this is not a problem. But, if the policy is deterministic, the only way to go through the situation is if the other agents, involved in it, perform an action that changes the resulting configuration. To explain this, let's consider an example. Let Agent 1 be in the state show in Fig. 4.6(a) where the action associated with this configuration is "rotate clockwise if at least one agent is insight its right sensor sensing area". Agent 7 is in the sensing area of Agent 1's right sensor, so Agent 1 turns right and reaches the state shown in Fig. 4.6(b). Now, Agent 7 is sensed with the Agent 1 left sensor. If the action associated with this configuration is "rotate counterclockwise if at least one agent is in its left sensor sensing area", when Agent 1 makes its movement, the state shown in Fig. 4.6(a) is reached. If none of the rest of the agents makes a movement, this situation is unsolvable.

Facing: When at least two agents are faced preventing each others' movement; the policy is deterministic; and the action for all agents involved in this situation is "move forward". Fig. 4.7 shows the problem of facing among agents.



(a) *Agent 1 upwardly facing, sensing Agent 7 on the right.*



(b) *Agent 1 facing right, 7 on the left.*

Figure 4.6: *States of the environment before and after Agent 1 rotates* 90° *clockwise, with the corresponding sensing area. Fig. 4.6a shows the Agent 1 before rotating. Fig. 4.6b shows Agent 1 after rotating.*

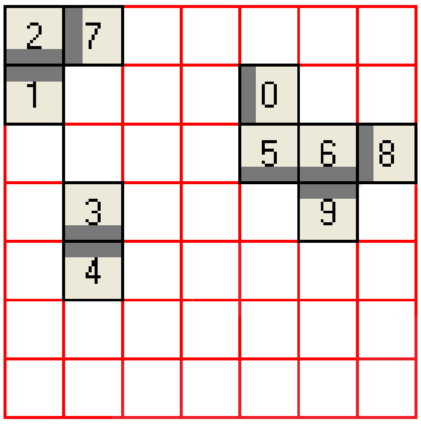


Figure 4.7: *Facing example. It is possible to observe three groups of agents facing each other. Agents 1, 2 and 7 form the first group. Agent 3 and 4 the second group. Agents 6, 8 and 9 the third group. Agents 0 and 5 do not have problems in this situation.*

Remoteness: When it is necessary to know proximity of agents. That is to say, if there are agents near or far.

All the aforementioned problems may be avoided by making changes to the sensing capability until finding one with minimal number of possible configurations with a low probability of conflictive perceptual aliasing.

Improving sensing capability using trial and error is a cumbersome matter, and lot of time is needed to find the appropriate sensing capability to solve a problem efficiently (in both, homogeneous and heterogeneous behaviors; being worse in the case of heterogeneous agents). Hence, the aim is to find a method to discover the minimal sensing capability (in terms of the number of possible configurations) such that the agents that use it can learn a policy for a given problem. This policy may be either deterministic or stochastic.

One way that an agent can learn its sensing capability is by defining a parameterized function, using a global searching method to look for values that allows learning the behaviors to solve the original problem. Characteristics of several sensors such as infrared proximity detectors, ultrasonic sensors, scanner and video camera, were taken into consideration when the parameters where defined. For example, an ultrasonic sensor has limited scope but can measure the distance to an object, while a CCD camera has unlimited scope but gives no information about the distance of the objects in front of it. Once the type of parameters is determined, their values must be found. To deal with this, an optimization algorithm such as Genetic Algorithms (see [16-17]) is used. If the behavior is homogeneous (see Sec. 1) all agents can use the same sensing capability. But, if the behaviors needed, to solve the problem, are heterogeneous, then different sensing capabilities, for each agent or group of agents, must be found.

In this sense, it is necessary to consider which is the minimal number of agents that need to be trained to solve the problem, replicating that knowledge to the remaining agents in the multi-agent system.

4.1.1 Sensing Parameterization

Five main parameters are used to model the most common characteristics of several sensors:

1. **Scope (ρ)**: denoting the radius of the sensing area.

   Integer value in $[1, min(height, width)$, where *height* and *width*, are the discrete toroid sizes.

2. **Aperture (θ)**: denoting the sensing aperture angle. Integer value in $[0, \rho]$.

3. **Expansion (ν)**: denoting the shape of the sensing area. Integer value in $[-\rho + 1, \rho]$, without 0. If $\theta = 0 \Rightarrow \nu = \rho$.

4. **Closeness (β)**: denoting the number of layers into which the sensing area is divided. The layers are numbered starting in 1, referring the layer nearest to the agent and so on. Integer value in $[1, \rho]$.

5. **Density (δ)**: denoting how to measure the quantity of agents in each layer. Integer value in $[1, N]$ where N is the number of agents involve in the problem solution. There is a value for the first layer (δ_1) and another for the rest of the layers (δ_r); therefore, the quantity of parameters to be considered, from now on, will be six.

The shape of the sensing area is determined by the values of ρ, θ and ν. Fig. 4.8 shows, for one sensor, several examples for different values of these three parameters.

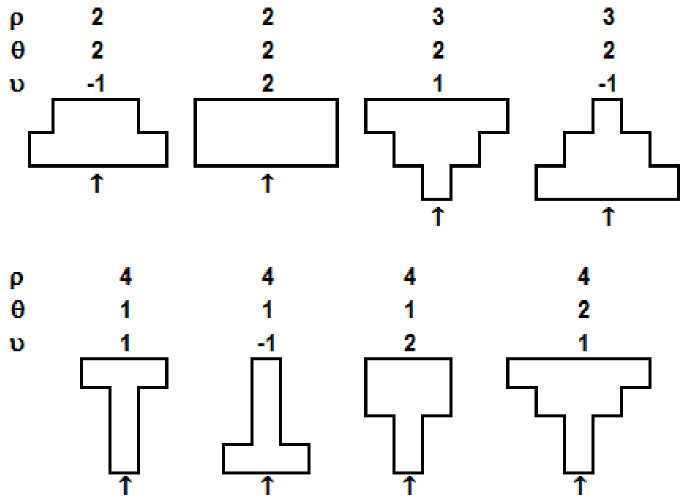


Figure 4.8: *Scope, Aperture and Expansion examples. The arrow (↑) represents the agent. The arrowhead points to the front sensor.*

The range of the value of each parameter can depend on the values of some of the other parameters. The maximum value of β depends on ρ; the number of possible configurations of the agent is determined as a function of β, δ_1 and δ_r for each sensor (front, right, back, left).

The sensing area is divided starting from the row closest to the agent which is sensing. All layers will have a single row except the layer farthest from the agent, which have the remaining rows completing the area. The division is performed in this way to have greater discernment near the agent which is sensing. Table 4.1 displays the number of rows in each layer. For example, if $\beta = 3$ and $\rho = 4$ the first layer has one row, the second layer has one row and the third layer has the remaining two rows.

Fig. 4.9 shows four possible values for β when $\rho = 4$. Regarding the density of the first layer, the nearest layer to the agent, it has more variety in values than the

Table 4.1: *Number of rows in each layer depending on the value of β and ρ.*

	ρ				
β	**1**	**2**	**3**	**4**	...
1	1	2	3	4	...
2	-	1,1	1,2	1,3	...
3	-	-	1,1,1	1,1,2	...
4	-	-	-	1,1,1,1	...

remaining layers. Table 4.2 displays the set of possible values for the first layer. Remaining layers only have two values as displayed in Table 4.3. In both tables column *Meaning* shows the different possibilities for the parameter.

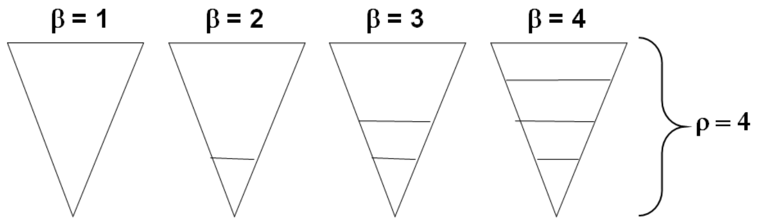


Figure 4.9: *Closeness examples for $\rho = 4$.*

Table 4.2: *Density for the first layer (δ_1).*

$\delta_1 = i$	
Value	**Meaning**
0	No agents.
1	At least 1 agent on either side.
2	At least 2 agents on either side.
3	...
$i-1$	At least $i-1$ agents on either side.
i	1 agent in front.
$i+1$	1 agent in front and at least 1 agent on either side.
$2i-2$	1 agent in front and at least $i-2$ agents on either side.
$2i-1$	1 agent in front and at least $i-1$ agents on either side.

An example of sensing area and the corresponding parameter values for only one sensor is shown en Fig. 4.10. In Fig 4.11 can be seen an example of sensing area and the corresponding parameter values for four sensors.

Table 4.3: *Density for the remaining layers (δ_r).*

$\boldsymbol{\delta_r = 1}$	
Value	**Meaning**
0	No agents.
1	At least 1 agent.

$\boldsymbol{\delta_r = 2}$	
Value	**Meaning**
0	No agents.
1	At least half of the positions with agents.
2	More than half of the positions with agents.
3	All positions with agents.

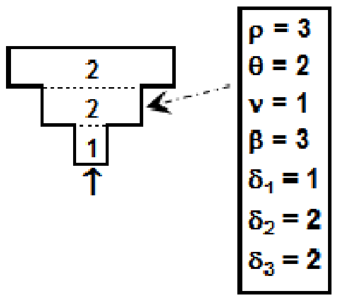


Figure 4.10: *Parameterization example for only one sensor. The smallest arrow represents the agent, and its arrowhead points to the front.*

The state of each agent is represented by a 5-tuple (F, R, B, L, G), where F, R, B and L are the output values of the Front, Right, Back and Left sensors. The element G represents the grouping of all agents somewhere in the toroid. When an agent joins with agents, they start to exchange information about whether they are seeing other agents or not. If all agents are together, then the component G is set to 1, otherwise it is 0. Agents have an internal memory where they record information obtained from agents with whom they are connected, at the time of the sensing, in order to determine the value of component G. In the experiments performed in this thesis there are two different ways to obtain these values. In the case of clustering problems, the internal memory is an array of as many positions as the number of agents involved in the multi-agent system. A given position in the internal memory is active when the agent that owns that memory is in contact with the agent represented by this position. In this case, the agent which is sensing

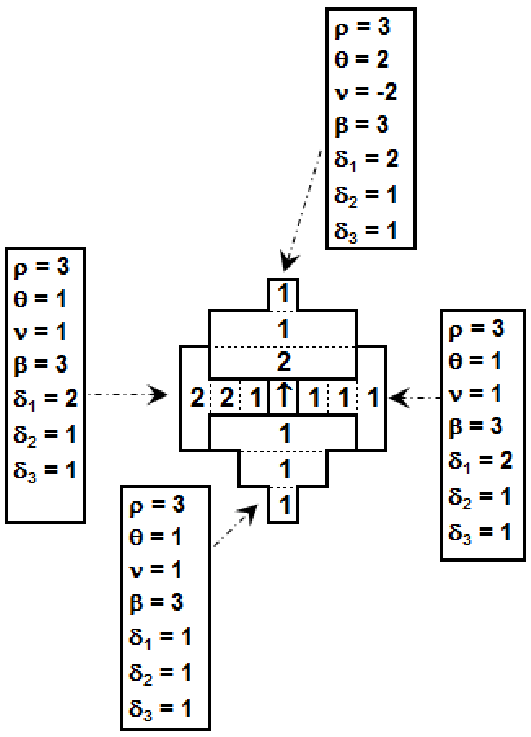
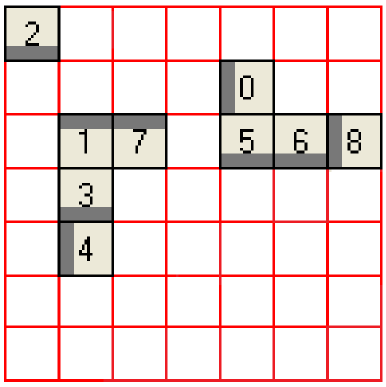


Figure 4.11: *Parameterization example for four sensors. The smallest arrow represents the agent, and its arrowhead points to the front.*

recognizes the agents with which it is connected, and collects information about which agents they, in turn, are connected to. And so on recursively. Fig. 4.12 shows nine agents distributed in a toroid and a table with the internal memories of each agent. In Fig. 4.12(a), Agent 1 knows that Agent 7 and Agent 3 are connected to him; Agent 7 knows that he is connected to Agent 1, and Agent 3 knows that he is connected to Agent 5. Hence, Agent 1 knows that he is connected to Agent 4 after having exchanged the corresponding information with Agent 3. When this recursive process has ended, Agent 1 knows that he has reached the goal.

Let Agent 1 be the agent which is sensing. It exchanges information with the agents touching him. When this process has ended, Agent 1 is able to determine if all agents are connected (see Fig. 4.13).

In the other geometric patterns, the agent has the same array as stated before plus one position. This last position is active when the rest of the positions are active. When agents are in contact they exchange this last position in the same way that was explained for the clustering problem in previous paragraphs.



(a) *Nine agents distributed in a discrete toroid.*

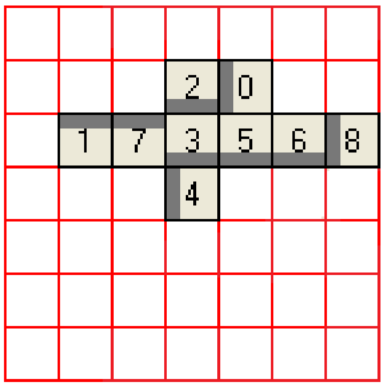
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0						x	x		x
1				x	x			x	
2									
3		x			x			x	
4		x		x				x	
5	x						x		x
6	x					x			x
7		x		x		x			
8	x					x	x		

(b) *Internal memory for each agent presented in Fig.4.12a. Rows represent agents and columns represent the agents in contact with.*

Figure 4.12: *Possible state for nine agents distributed in a discrete toroid and their internal memories. Fig.4.13a shows nine agents in one possible state during the resolution of the problem. Table 4.12b displays the values of the corresponding internal memories. In this example, Agent 0 knows that he is connected to Agents 5, 6 and 8; Agent 1 knows that he is connected to Agents 3 and 7; Agent 2 knows that he is alone; Agent 3 knows that he is connected to Agents 1, 4 ant 7; and so on. Therefore, Agent 1 knows that he is connected to Agent 4.*

Fig. 4.14a shows a possible intermediate state with nine agents and a table with the internal memory for each agent. The first nine columns in the table will be active if the agent (corresponding to the row) is in contact with the agents named in the columns. The last column will be active if the agent represented by the row is in contact with all the necessary agents to complete the desired shape.

Fig. 4.15a shows a final state where the desired formation, a triangular shape, is reached and a table with the corresponding internal memories.



(a) *Another distribution of nine agents into a discrete toroid.*

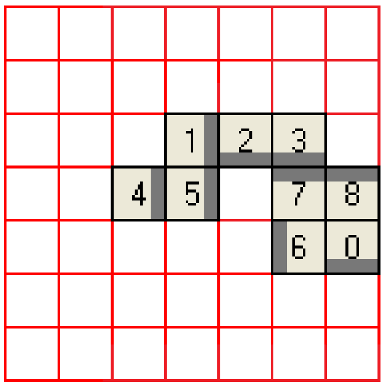
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0		x	x	x	x	x	x	x	x
1	x		x	x	x	x	x	x	x
2	x	x		x	x	x	x	x	x
3	x	x	x		x	x	x	x	x
4	x	x	x	x		x	x	x	x
5	x	x	x	x	x		x	x	x
6	x	x	x	x	x	x		x	x
7	x	x	x	x	x	x	x		x
8	x	x	x	x	x	x	x	x	

(b) *Internal memory of each agent involved in the state shown in Fig. 4.13a. Rows represent agents and columns represent positions.*

Figure 4.13: *Another possible state for nine agents distributed in a discrete toroid and their internal memories. Fig.4.13a shows the connections among agents, and Table 4.13b displays the corresponding internal memories. As all agents are connected, all positions in the internal memories are active.*

The remaining elements of the five-tuple are calculated as follows. Let $s_F(layer)$, $s_R(layer)$, $s_B(layer)$ and $s_L(layer)$ be the given values of front, right, back and left sensors, respectively, for each layer of an agent in particular. Let $\boldsymbol{s_F}$, $\boldsymbol{s_R}$, $\boldsymbol{s_B}$ and $\boldsymbol{s_L}$ be the corresponding arrays of the sensed values for all layers of each sensor, and $\boldsymbol{\delta_F}$, $\boldsymbol{\delta_R}$, $\boldsymbol{\delta_B}$ and $\boldsymbol{\delta_L}$, the density arrays for each layer for each sensor. Then, the sensor output value for this agent is calculated according to Eqs. 4.1 to 4.4.

Given the parameter values of each sensor the number of possible configurations is calculated according to Eq. 4.5.



(a) *Possible intermediate state with nine agents when the final state is a triangular shape.*

	0	1	2	3	4	5	6	7	8	Exchange
0										
1			x			x				1
2		x		x						
3			x					x		1
4						x				1
5		x			x					
6	x							x		
7				x			x		x	1
8	x							x		

(b) *Internal memory for each agent present in Fig.reffig:arrowA1. Rows represent agents and columns represent the agents in contact with.*

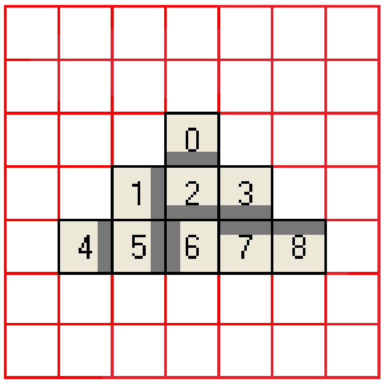
Figure 4.14: *Possible intermediate state with nine agents when the final state is a triangular shape and the corresponding internal memories. If Agent 1 is the agent which is sensing, it asks Agent 2 and 5 for information. Agent 5 completes its job but Agent 2 does not. Therefore, the internal memory for Agent 1 is not complete.*

$$F(\beta_F, \boldsymbol{\delta_F}, \boldsymbol{s_F}) = s_F(\beta_F) + \sum_{i=1}^{\beta_F - 1} 2^{\beta_F - i} * s_F(i) * \prod_{j=i+1}^{\beta_F} \delta_F(j), \tag{4.1}$$

$$R(\beta_R, \boldsymbol{\delta_R}, \boldsymbol{s_R}) = s_R(\beta_R) + \sum_{i=1}^{\beta_R - 1} 2^{\beta_R - i} * s_D(i) * \prod_{j=i+1}^{\beta_R} \delta_R(j), \tag{4.2}$$

$$B(\beta_B, \boldsymbol{\delta_B}, \boldsymbol{s_B}) = s_B(\beta_B) + \sum_{i=1}^{\beta_B - 1} 2^{\beta_B - i} * s_B(i) * \prod_{j=i+1}^{\beta_B} \delta_B(j), \tag{4.3}$$

$$L(\beta_L, \boldsymbol{\delta_L}, \boldsymbol{s_L}) = s_L(\beta_L) + \sum_{i=1}^{\beta_L - 1} 2^{\beta_L - i} * s_L(i) * \prod_{j=i+1}^{\beta_L} \delta_L(j). \tag{4.4}$$



(a) *Final state with nine agents when the goal is a triangular shape.*

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Exchange**
0			x							1
1			x			x				1
2	x	x		x			x			1
3			x					x		1
4						x				1
5		x			x		x			1
6			x			x		x		1
7				x			x		x	1
8								x		1

(b) *Internal memory for each agent present in Fig.4.15a. Rows represent agents and columns represent the agents in contact with.*

Figure 4.15: *Final state with nine agents when the goal is a triangular shape and the corresponding internal memories. Because the last column is "active" for all agents, the goal is reached.*

$$Conf(\boldsymbol{\beta}, \boldsymbol{\delta_F}, \boldsymbol{\delta_R}, \boldsymbol{\delta_B}, \boldsymbol{\delta_L}) = 2^{(\beta_F+\beta_R+\beta_B+\beta_L+1)} \prod_{i=1}^{\beta_F} \delta_F(i) \prod_{i=1}^{\beta_R} \delta_R(i) \prod_{i=1}^{\beta_B} \delta_B(i) \prod_{i=1}^{\beta_L} \delta_L(i)). \tag{4.5}$$

For example, the number of possible configurations for the four sensors shown in Fig 4.11 is 32768.

It should be remembered that a parameterization of a sensing capability with a low number of possible configurations, and minimizing conflictive perceptual aliasing is wanted.

The maximum density per layer is a function of ρ, θ, ν and β and it is defined by Eq. 4.6.

$$\delta(l) = 1 + 2 \sum_{i=l}^{f(l)+l-1} ap(i) \qquad \forall 1 \leq l \leq \beta \tag{4.6}$$

where l identifies the layer; the function $f(l)$ defines the number of rows in each layer and it is calculated according to Eq. 4.7; and $ap(l)$ defines the aperture of each row and it is calculated according to Eq. 4.8. If $l \geq 2$ and $\delta(l) > 2 \Rightarrow \delta(l) = 2$.

$$f(l) = \begin{cases} 1 & l \neq \beta \\ \rho - \beta + 1 & l = \beta \end{cases} \tag{4.7}$$

$$ap(i) = \begin{cases} \theta & Cond_1 = True \\ 0 & Cond_2 = True \\ ap(i + sgn(\nu)) - 1 & otherwise \end{cases} \tag{4.8}$$

where

$$Cond_1 = \begin{cases} True & i = g(sgn(\nu)) \ \vee \ sgn(|\nu| - \rho + i * sgn(\nu)) = sgn(\nu) \\ False & otherwise \end{cases}, \tag{4.9}$$

and

$$Cond_2 = \begin{cases} True & i \neq g(sgn(\nu)) \ \wedge \ ap(i + sgn(\nu)) - 1 < 0 \\ False & otherwise, \end{cases}. \tag{4.10}$$

Function $g(x)$ used in Eq 4.9 and 4.10 is calculated according to Eq. 4.11.

$$g(x) = \frac{\rho - 1}{2} * x + \frac{\rho + 1}{2}. \tag{4.11}$$

4.2 Method to Obtain Individual Behaviors (ObIndBe)

The learning process is episodic and is as follows: agents start at random positions in the toroid carrying out actions (decided by exploration and/or exploitation) until the goal is reached or the maximum number of iterations is reached. If applicable, a reward is given. The process continues with agents starting in other random positions, and so on. The method uses the Algorithm ObIndBe to do this.

4.2.1 Algorithm ObIndBe

```
Input: Q matrix for each agent.
Output: Individual policies.
begin
    Place agents randomly in the toroid.
    for each agent, named i do
        Initialize matrix, $Q_i$.
    end
    while the maximum number of learning process not completed do
        while the goal was not reached or the maximum number or
        iterations not completed do
            if the goal was reached then
                Place agents randomly in the toroid
            else
                for each agent i picked randomly do
                    Obtain the current configuration.
                    Pick an action using action directed exploration
                    Apply the action.
                    Obtain the reward.
                    Update matrix $Q_i$.
                end
            end
        end
    end
    Obtain the individual behaviors.
end
```

Algorithm 1: ObIndBe. Algorithm to obtain individual behaviors.

This algorithm is an improved version of Q-learning. Q-learning for multi-agent systems was implemented as a direct extension of Q-learning for single agents. Therefore, each agent has its own Q matrix, choosing its action independently and concurrently. It is important to notice that, although the learning is carried out independently for each agent, the observed information for each agent depends of the actions leads by the rest of the agents in the multi-agent system. It should be borne in mind that, while the learning is done independently for each agent, the observed data for each individual in the population depends on the actions taken by other agents.

The reward function is one of the most important components of Q-learning with respect to learning. In the proposed method a purely delayed positive reward function was used. That is to say, the agents are rewarded with a value of 100 if they reach the goal. Otherwise, the reward is 0. Analytically, the reward function is represented by Eq. 4.12.

$$FR(s_j(t)) = \begin{cases} 100 & \forall s_j = (F_j,\ R_j,\ B_j,\ L_j,\ G_j),\ G_j = 1, \\ 0 & otherwise \end{cases} \tag{4.12}$$

where s_j is the configuration for Agent j at time t after sensing the environment. Although the reward function is global, individual rewards are given to each agent. Only the agent that reaches the goal will be rewarded.

4.2.2 Exploration

By definition, Q-learning, using *ε_greedy*, has only one parameter to manage the balance between exploration and exploitation, named ε. Therefore, the best action is selected with probability $1 - \varepsilon$. From here on, this parameter will be named *ε_global*. In the experiments presented in this thesis, this parameter decreases linearly from 1 to 0 with the iteration number. This linear decrement can generate the following problem: in a very advanced learning stage there might be configurations that were never seen before, losing the chance to explore them. To avoid this potential problem a new parameter is added for each possible configuration (see [23]). This new parameter is named *ε_individual* and is independent from *ε_global*. *ε_individual* is initially set to 1 and is linearly decreased to 0 in the range of 0 to top_ε, where top_ε is the maximum number of visits allowed a configuration to be explored; aiming to sufficiently explore all configurations needed to reach the goal. Despite this, it is possible that some configurations will not be explored enough to enable the agent to determine the most appropriate action, as shown in Chapter 5. If the behavior to learn is homogeneous, there is a low probability that certain configurations will remain without exploration, because the knowledge acquired by each agent, during learning, will contribute to the knowledge of all agents (see Chapter 5). In contrast, if the behavior needed is heterogeneous, the problem of not visiting some important configurations is more likely.

Because the environment is stochastic and due to the existence of perceptual aliasing, it is possible that different actions, from the same configuration, give the same successor configuration. Fig. 4.16 shows a possible state for Agent 1 and Table 4.4 displays the sensing capability parameter values used to obtain the corresponding configuration. With these parameter values the configuration for Agent 1 is "01:01:00:00:0" (meaning "F:R:B:L:G").

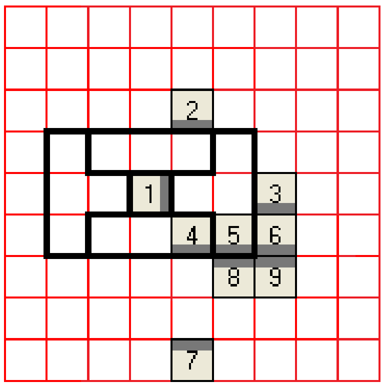


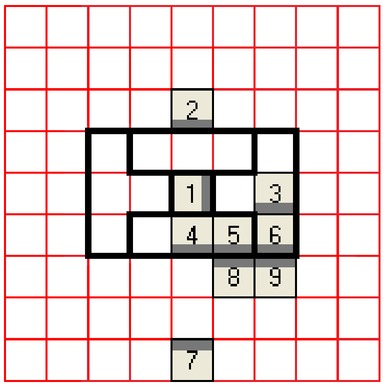
Figure 4.16: *Possible state for Agent 1.*

Table 4.4: *Sensing capability parameter values for Agent 1.*

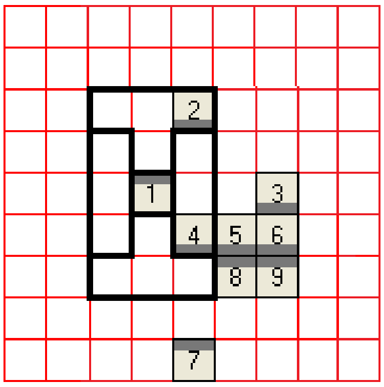
	ρ	θ	ν	β	δ_1	δ_r
Front	2	1	1	1	1	1
Right	1	1	1	1	1	1
Back	2	1	1	1	1	1
Left	1	1	1	1	1	1

Fig. 4.17 shows the three possible successor states for the same agent. Fig. 4.17a shows the resulting configuration after the agent moves forward, whose value is "01:01:00:00:0". Fig. 4.17b shows the resulting configuration after the agent rotates counterclockwise 90°, whose value is "01:01:00:00:0". And Fig. 4.17c shows

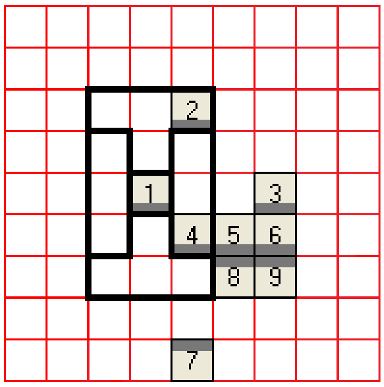
the resulting configuration after the agent rotates clockwise 90°, whose value is "00:00:01:01:0".



(a) *Successor state if Agent 1 moves forward.*



(b) *Successor state if Agent 1 rotates counterclockwise* 90°.



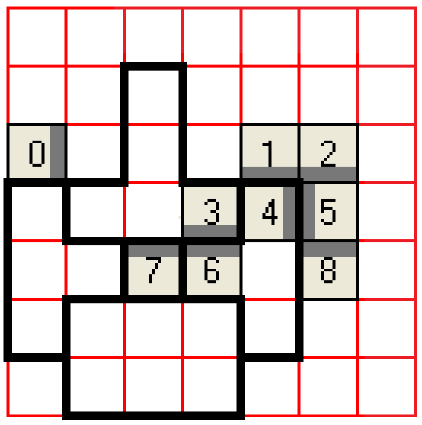
(c) *Successor state if Agent 1 rotates clockwise.*

Figure 4.17: *Possible successor states obtained from the state in Fig. 4.16 for Agent 1.*

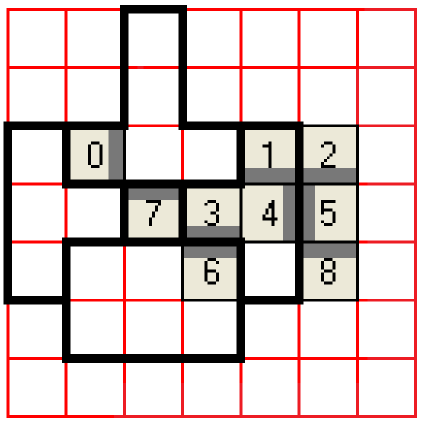
It can be seen that, although the states obtained are different, moving forward a position or rotating clockwise 90° give the same configuration. Once more it is clear that the configuration is totally dependent on the parameterization of the sensing.

On the other hand, the same action taken by the agent in a configuration can lead to different successor configurations. That explains why it is important to evaluate the state each time the agent decides to perform an action. Fig. 4.18(a) shows Agent 7 before moving one position forward obtaining a configuration represented by "00:06:01:01:0", using the parameter values displayed in

Table 4.5. When Agent 7 has to move again, the environment has changed because of Agent 0 and the configuration obtained after sensing is "08:06:01:00:0", as shown in Fig. 4.18(b).



(a) *Sensing area for Agent 7 before moving.*



(b) *Sensing area for Agent 7 after moving forward.*

Figure 4.18: *Sensing area for Agent 7 before and after moving forward. Fig. 4.18a shows the agent before performing an action. Fig. 4.18b shows the same agent after moving forward.*

Table 4.5: *Parameter values used in Fig. 4.18.*

	ρ	θ	ν	β	δ_1	δ_r
Front	4	1	-1	2	2	1
Right	2	1	1	1	3	1
Back	2	1	2	1	1	1
Left	2	1	1	2	1	1

Under these conditions the following questions arise: if the agent wishes to move towards configurations not visited or low-visited, how can it select the action that leads to such configurations? Given the stochastic nature of the environment, the realization of a given action does not guarantee the visit of a given successor configuration. Therefore, perhaps rather than exploring low-visited configurations, agents should pick actions ensuring that most configurations have been selected a minimum number of times. To solve this problem *action directed exploration* is implemented in the method.

4.2.2.1 Action Directed Exploration

Under action directed exploration an agent decides which action to apply taking into account three aspects:

1. The successor configurations,
2. Previous actions carried out from the current configuration, and
3. The Q values for the current configuration.

The algorithm uses three parameters (set at the beginning of the execution):

- *threshold_config* for all possible configurations for all agents,
- *threshold_action* for all possible actions for all agents, and
- *Q_percentage*, the value that a pair (s, a) must have in matrix Q to be considered for exploration.

Each time an agent performs an action from a configuration, the number of visits to this configuration is increased. There is a different counter for each configuration for each agent. Each time an action is used to make a movement, the counter of times the action was used is increased. There is a different counter for each action for each agent. Based on these counters and parameters, the algorithm considers a configuration as *saturate* when it was visited a *threshold_config* times. Additionally, the algorithm considers an action as *saturate* when it was used a *threshold_action* times.

Bearing in mind that, without perceptual aliasing and/or in a deterministic environment, the number of times that an action is used and the number of visits to the successor configuration are the same. In this case both thresholds, configurations and actions, are equivalent. For example, if *threshold_action* is 5 and *threshold_conf* is 3, the corresponding configuration will be explored at least 3 times and, once it had been saturated, only 2 more explorations will be done to saturate actions. That is to say, when an action is saturated the corresponding successor configuration was visited the same number of times that the action was applied. But when perceptual aliasing is presented the action saturation and the configuration saturation could be slightly different. In this case, *action directed exploration* is important.

The directed exploration is performed with the following algorithm.

```
Input: Let $c_i$ be the current configuration of the agent,
$A_{c_i}$ be the set of possible actions from configuration $c_i$,
$C_{i+1} = c_{i+1}, c'_{i+1}, c''_{i+1}, \ldots$ be the set of possible successor configurations for
$c_i$.
Output: An action.
begin
    Calculate the set of non-saturate successor configurations according to:
        $CnoSat = \{x/x \in C_{i+1} \wedge Visits(x) < threshold\_config\}$
    if $CnoSat \neq \emptyset$ then
        Apply the action $a_i$ which leads to $c_{i+1} \in CnoSat$, choosing
        randomly among all the elements in $CnoSat$
    else
        Calculate the set of non-saturate actions according to:
            $AcnoSat = \{x/x \in A_{c_i} \wedge Used(x) < threshold\_action\}$
        if $AcnoSat \neq \emptyset$ then
            Choose action $a_i$ randomly among all actions in $AcnoSat$
        else
            Obtain the list $L$ with all possible actions $a_i \in A_{C_i}$, in increasing
            order according to $Q(c_i, a_i)$.
            Let $head$ be the first element of $L$ and $second$ be the second
            element of $L$.
            if $Q(c_i, head) > Q\_percentage * Q(c_i, second)$ then
                $a_i = head$
            else
                Choose $a_i$ randomly among all elements in $AcnoSat$
            end
        end
    end
end
```

Algorithm 2: Action Directed Exploration Algorithm.

4.3 Method to Obtain the Sensing Parameter Values (ObSenPar)

The Algorithm ObSenPar is used to obtain the parameter values for the sensing function. This algorithm implements a Genetic Algorithm (see [13] and [10]) with the following characteristics:

1. Genes, in the chromosome, represent the parameter values for each sensor.

2. Genetic operators:

   - Selection: 10% of the individuals were chosen with Elitist and the remaining 90% using Roulette.
   - Crossover operator picks a parent from the set of chromosomes obtained with Elitist and the other parent from the set obtained with Roulette.
   - The Algorithm ObIndBe (see Section 4.2) is added as a new genetic operator. It is used to calculate the fitness value for the chromosome. The number of iterations and learning process is established at the beginning of the algorithm. The values of Q matrix obtained at the end of each learning process are used as initial values for the next learning process for each agent in each chromosome.

3. The fitness function is the result of the testing of Individual Policies obtained through the operator explained in the previous item (2) weight by the number of possible configurations represented in the corresponding chromosome. Mathematically,

$$Fitness(x) = \tau Effectiveness(x) + (1-\tau)e^{\frac{(Conf(x)-\mu)^2}{2\sigma}}, \qquad (4.13)$$

   where x is the chromosome; $Effectiveness$ is the value obtained applying the new operator involving Algorithm ObIndBe; $Conf(x)$ is the number of possible configurations for the sensing capability represented by chromosome x; τ is a proportionality constant; μ is the expected value of the number of possible configurations; σ is the variance of the number of possible configurations.

The required number of test repetitions for fitness value is established at the beginning of the method. The test is performed using the sensing capability represented in the chromosome and the policy obtained during the learning. Each time the problem is solved, the testing process counts it as a success.

4.3.1 Algorithm ObSenPar

```
Input: An initial random population.
Output: Sensing capability for each agent.
begin
    while the maximum number of generations is not completed or the
    population does not remain the same do
        Select chromosomes from the population.
        Apply the crossover operator in a single point.
        Apply the mutation operator to the offspring.
        Apply Algorithm ObIndBe
        Obtain the fitness value for each new chromosome.
        Apply replacement operator to obtain the new generation.
    end
end
```

Algorithm 3: ObSenPar. Algorithm to obtain the sensing parameter values.

How to apply the methods presented here will be seen in the next chapter. To this end, a series of experiments will be presented and their results will be analyzed. This, together with some new definitions, will contribute to a more comprehensive understanding of the problems and solutions proposed in this thesis.

Chapter 5

Experiments and Results

As it was stated previously, the scope of the problems to be solved is restricted to pattern formation in a finite discrete toroid.

Three different experiments are conducted to validate the proposed methods (see Chapter 4), which involve the ability to obtain sensing capabilities and behaviors of agents in the multi-agent system:

1. Experiments with grouping problems: Obtaining a group or cluster of ten agents anywhere in a toroid. Fig 5.1 shows some different ways of grouping ten agents, each of which is considered a goal for this experiment within the set of all possible solutions to this problem.

   In this experiment, the way in which the sensing capability is obtained will be separated in three different studies using:

   (a) Trial and error (see Section 5.1.1).

   (b) The proposed methods to obtain only three out of six parameters (see Section 5.1.2). Parameters ρ, θ and ν are fixed at the beginning of the experiment, and β, δ_1 and δ_r are learned.

   (c) The proposed methods to obtain all six parameters (see Section 5.1.3).

2. Experiments with square shape: Obtaining a square shape anywhere in a discrete toroid, using nine agents, as shown in Fig. 5.2.

   In this experiment, the way of obtaining the sensing capability will be divided into two different studies using:

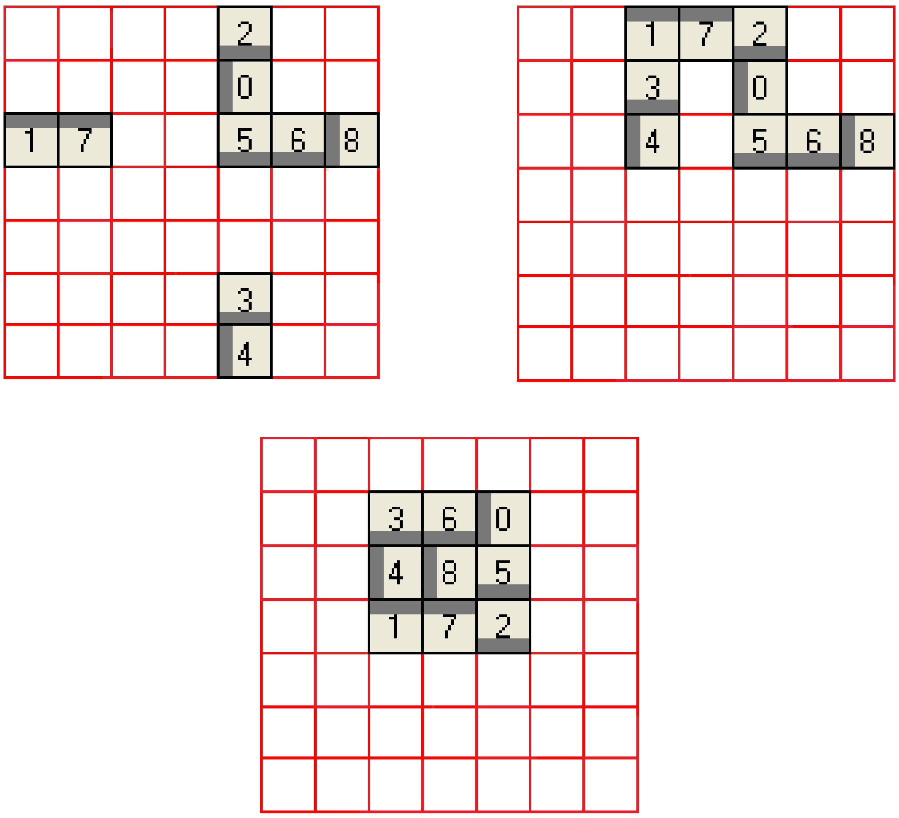


Figure 5.1: *Possible grouping formations with ten agents.*

(a) The proposed methods to obtain only three out of six parameters (see Section 5.2.1). Parameters ρ, θ and ν are fixed at the beginning of the experiment, and β, δ_1 and δ_r are learned.

(b) The proposed methods to obtain all six parameters (see Section 5.2.2).

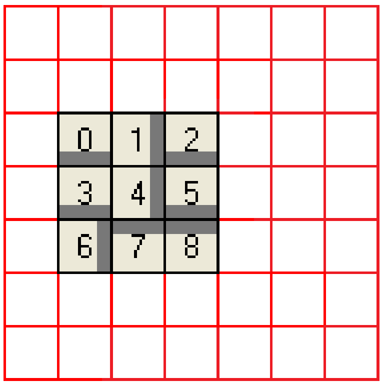


Figure 5.2: *Square shape to be obtained in this experiment.*

3. Testing proposed methods: Obtaining two different shapes:

(a) A triangular shape anywhere in a discrete toroid, using nine agents, as shown in Fig. 5.3 (see Section 5.3.1).

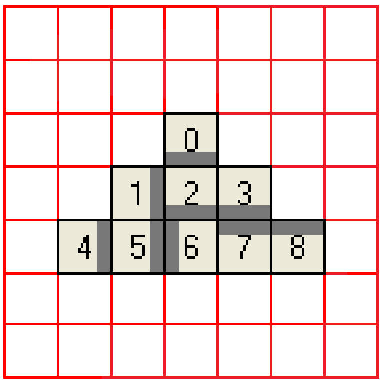


Figure 5.3: *Triangular shape to obtain in Experiment 3a.*

(b) A diagonal shape anywhere in a discrete toroid using ten agents as shown in Fig. 5.4 (see Section 5.3.2).

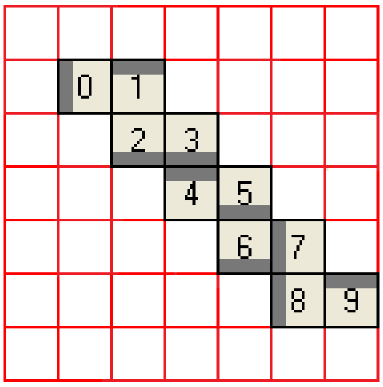


Figure 5.4: *Diagonal shape to obtain in Experiment 3b.*

Experiment 1a, in Section 5.1.1, is carried out to obtain the individual behaviors of a group of ten homogeneous agents using Algorithm ObIndBe. It is useful to prove that it is not necessary to train all agents involved in the solution of the problem. In fact, only a reduced subgroup can be trained, then replicating the knowledge learned to the remaining agents in the multi-agent system. The sensing capability is set before the Algorithm ObIndBe starts and it is found by trial and error.

Experiment 1b, in Section 5.1.2, aims to obtain the sensing capability of the agents using Algorithm ObSenPar (see Algorithm 3 in Section 4.3.1) and the individual behaviors using Algorithm ObIndBe (see Algorithm 1 in Section 4.2.1).

The number of agents used to prove the methods is the same as in Experiment 1a, but only three out of six parameters are found by the Algorithm ObSenPar.

Experiment 1c, in Section 5.1.3, uses both proposed algorithms to obtain all six parameters. This experiment is divided into three phases determined by the use of initial distributions and the type of policy that will be obtained (Deterministic or Stochastic).

Experiment 2 in Section 5.2 is performed to force policy differentiation.

Experiment 3a in Sections 5.3.1 and 5.3.2 are examples of how the proposed methods work when the goal is a triangular shape and a diagonal shape, respectively.

In all experiments, several tests are carried out changing the number of agents and/or the size of the toroid, using the policies and sensing capabilities learned before. That is, without having to make additional training processes. A comparison with a Random Policy is carried out in all experiments.

All the experiments have the following common features:

- Agents have the following characteristics:
  - Autonomous behavior.
  - Three possible movements: one position to the front, rotate clockwise 90° and rotate counterclockwise 90°.
  - Default action, setting at the beginning of the method.
  - Four sensors: front, right, back and left.
  - Low-communication among them.
  - Own reinforcement matrices Q.
- Agents have the following requirements:
  - Must move into a 7×7 discrete toroid during the learning processes.
  - Two agents are neighbors if both have a common side.
- Algorithm ObIndBe settings:
  - Parameter ε_global is initialized in 1.
  - Parameter α is initialized in 0.1.

  - Parameter $Limit_\alpha$ is set to 2660 iterations. That is to say, the parameter α starts decreasing at the iteration 2660 reaching 0 at the end of the learning process.
  - Parameters ε_{ij}, ($\varepsilon\_individual$), $\forall i, j\ 1 \leq i \leq Conf$, $1 \leq j \leq Nagents$, initialized in 1; all linearly decreasing. $Conf$ is calculated following Eq. 4.5. $Nagents$ is the number of agents involved in the learning process.
  - Parameter γ is set to 0.9.
  - The threshold_config is set to 1.
  - The threshold_action is set to 5.

- Algorithm ObSenPar settings:
  - The chromosome length is determined in each experiment because it depends on the number of parameters to be learned.
  - The population is composed of 50 individuals.
  - The fitness value, or effectiveness, is reached by testing the obtained policies 300 times after seven learning process with 1400 iterations each.

- Once the optimal set of parameters is obtained, the policies are improved with extra learning processes. For this to happen, fifteen learning processes with 11200 iterations each are conducted.

- The number of testing processes accomplished after the learning process is 300 with 500 iterations each.

- During the testing process, if any agent obtained a configuration that did not appear in any of the learning processes, the agent carried out the action obtained from the closest configuration. Closeness between configurations is calculated using 1-norm distance.

5.1 Experiments with grouping problems

In this experiment the Algorithm ObIndBe is used to find individual behaviors that allow solving the grouping problem with ten agents anywhere into a discrete toroid. In this case, matrix Q for each agent, is reinitialized in each learning process. The sensing capability is found by trial and error.

In this kind of problem, it will be shown that a Deterministic Policy can be used. Why is it possible to use a Deterministic Policy when it is well known [19, 14, 6, 35] that, if perceptual aliasing is present, a Stochastic Policy must be used? This can be explained looking at the number of possible solutions. Bearing in mind that agents are indistinguishable and any state where all agents are connected side by side is a solution, the number of final states represents a high percentage of the number of possible states for ten agents in the toroid. For example, if the problem is grouping nine agents in a 7×7 discrete toroid, the number of possible solutions is $2.3464e^9$. Because there are many final states the probability of states with conflictive perceptual aliasing is negligible against the probability of states representing a solution; in other words, the system is less sensitive in the case that conflictive perceptual aliasing is present. Therefore, when a Deterministic Policy is used to solve the problem, some choices made by any agent, have a high probability of reaching some of the possible final states mentioned above.

Moreover, taking into account that, the environment is purely stochastic, each agent has its own matrix Q and these matrices are reinitialized between learning processes, the policies obtained in each learning can associate different actions to the same configuration (see Chapter 1). In other words, the policies obtained after each training can be slightly different, and because agents are homogeneous, the knowledge obtained by each agent during each learning is another way to act under the same state. Remember that policies are obtained by choosing the action that gives the best expectation of future rewards (the action with the greatest value of Q) for each configuration. All configurations that were visited during exploration but never changed any value of Q (remaining in zero) will be assigned the default action. The policies thus obtained are named *Original Policies*. Therefore, there are different ways to obtain a Deterministic Policy from a group of independent training processes. All of these allow solving the stated problem. The way to obtain these policies is explained below. For this purpose, a hypothetical example will be used, and the process used to obtain the Deterministic Policies will be explained using several tables.

Let be a problem with only five configurations, where the number of learning process needed to train the agents is four with 30 iterations each. Table 5.1 displays the number of visits that each configuration had for each agent in each training in row "Visits", and the best action for each configuration in row "Action". Agents perform only three actions, a_1, a_2 and a_3, being a_1 the default action.

Table 5.1: *Original Policies, actions and number of visits per configuration. Each agent can do three different actions: a_1, a_2 and a_3.*

			Configuration				
	Agent		c_1	c_2	c_3	c_4	c_5
Training 1	**0**	**Action**	a_2	a_1	a_2	a_1	a_3
		Visits	10	6	7	0	7
	1	**Action**	a_1	a_3	a_1	a_2	a_2
		Visits	0	9	0	14	7
	2	**Action**	a_1	a_2	a_3	a_2	a_1
		Visits	3	10	15	2	0
Training 2	**0**	**Action**	a_2	a_3	a_2	a_2	a_2
		Visits	16	2	1	8	3
	1	**Action**	a_2	a_1	a_3	a_2	a_3
		Visits	4	11	4	7	4
	2	**Action**	a_1	a_3	a_2	a_1	a_1
		Visits	1	12	4	4	9
Training 3	**0**	**Action**	a_1	a_2	a_1	a_2	a_2
		Visits	10	15	0	3	2
	1	**Action**	a_1	a_3	a_2	a_2	a_3
		Visits	0	12	6	8	4
	2	**Action**	a_1	a_3	a_3	a_1	a_2
		Visits	4	9	2	12	3
Training 4	**0**	**Action**	a_3	a_2	a_1	a_1	a_2
		Visits	7	20	0	2	1
	1	**Action**	a_3	a_2	a_3	a_1	a_1
		Visits	7	6	9	3	5
	2	**Action**	a_1	a_1	a_3	a_1	a_2
		Visits	10	11	4	0	5

A configuration with zero visits means that it has never been explored. For example, in Training 1 Agent 0 visited configurations c_1, c_2, c_3 and c_5, but it never visited configuration c_4. It can be seen that, in different training processes, not only the number of configurations visited varies, but also the same configuration can be visited a different number of times. For instance, Agent 1 visited configuration c_2 nine times during Training 1 and twelve times during Training 3. In addition, the same agent can decide to use different actions in different training processes. For example, for configuration c_4, Agent 2 chose a_2 in the first training and a_1 in the last three. Therefore, there are twelve Original Policies to use, each of which may have information about situations that other policies might not have. A way to combine all the information held in these Original Policies is by taking the action that is most often chosen for each configuration by each agent in all training processes. That is to say, for each agent and configuration, the number of times an action is chosen is accumulated for all training processes. The action with the greatest counter value represents the action which has been more often the best (ties are broken randomly). This action is used to generate a Deterministic Policy for that agent. The resulting policy is named *Individual Policy.* Table 5.2 shows the number of times each action was chosen for each configuration for each agent and, in the last column, the Individual Policy obtained in this example. For instance, for configuration c_4, Agent 1 chose action a_1 once and a_2 three times. It never chose action a_3. Consequently, the Individual Policy for Agent 1 has a_2 as the action chosen in configuration c_4. In the particular case of Agent 0 and configurations c_3 and c_4, the number of times actions a_1 and a_2 were chosen is the same. To create the Individual Policy, in this case, one of the actions is chosen randomly. Once the Individual Policy is obtained, this is deterministic and there is no random choice of actions.

Individual Policies may also be different among agents, as can be seen in Table 5.2. Therefore, because the knowledge acquired by each agent is considered different views of the same agent, all the information learned by each of them, in all learning processes, can be used to build a new policy, named *General Policy.* It is built as the action that is chosen more often among all training processes for all agents for each configuration. Ties are broken randomly. Table 5.3 displays the number of times an action was chosen for each configuration, for all agents in all training processes. The last column shows the General Policy for this example.

Because no special shape is required, all agents involved in the solution of the problem, can share the learned knowledge. This is equivalent to saying that agents are homogeneous. Because of this, it is not necessary to train all agents. A minimum number can be trained, to later replicate the obtained knowledge in the remaining agents. To establish this number some incremental experiments are performed, keeping in mind that the number of agents should be as small as possible without losing important configurations that may appear when ten

Table 5.2: *Examples of Individual Policies for each agent. Columns a_1, a_2 and a_3 show how often an action was chosen because it was the best action. The last column shows the Individual Policy for each agent. The values are based on Table 5.1*

Agent 0				
Configuration	$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	$\boldsymbol{a_3}$	**Individual**
$\boldsymbol{c_1}$	1	2	1	a_2
$\boldsymbol{c_2}$	1	2	1	a_2
$\boldsymbol{c_3}$	2	2	0	a_2
$\boldsymbol{c_4}$	2	2	0	a_1
$\boldsymbol{c_5}$	0	3	1	a_2

Agent 1				
Configuration	$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	$\boldsymbol{a_3}$	**Individual**
$\boldsymbol{c_1}$	2	1	1	a_1
$\boldsymbol{c_2}$	1	1	2	a_3
$\boldsymbol{c_3}$	1	1	2	a_3
$\boldsymbol{c_4}$	1	3	0	a_2
$\boldsymbol{c_5}$	1	1	2	a_3

Agent 2				
Configuration	$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	$\boldsymbol{a_3}$	**Individual**
$\boldsymbol{c_1}$	4	0	0	a_1
$\boldsymbol{c_2}$	1	1	2	a_3
$\boldsymbol{c_3}$	0	1	3	a_3
$\boldsymbol{c_4}$	3	1	0	a_1
$\boldsymbol{c_5}$	2	2	0	a_2

Table 5.3: *General Policy. Columns a_1, a_2 and a_3 show how often an action was chosen among training processes for all agents. The last column shows the General Policy. The values are based on Table 5.1*

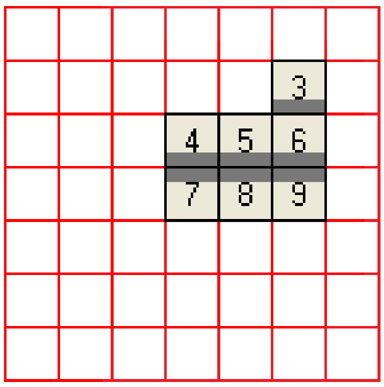
Agent 0				
Configuration	$\boldsymbol{a_1}$	$\boldsymbol{a_2}$	$\boldsymbol{a_3}$	**General**
$\boldsymbol{c_1}$	7	3	2	a_1
$\boldsymbol{c_2}$	3	4	5	a_3
$\boldsymbol{c_3}$	3	4	5	a_3
$\boldsymbol{c_4}$	6	2	0	a_1
$\boldsymbol{c_5}$	3	6	3	a_2

agents work together to achieve the goal. Grouping a single agent in an any dimension discrete toroid has an effectiveness of 100% whithout any restriction on the sensing capability. But when this knowledge is used by the rest of the agents, the effectiveness is around 0%. When two agents are involved in the learning processes, the effectiveness is higher than the one obtained using a single agent. But the number of configurations present during the training is still lower than the number of configurations that appear when ten agents work together. And the effectiveness remains at around 0%. Experimentally, the number of agents needed to be trained is three. This is the number of agents that must be used in Experiments 1a, 1b and 1c.

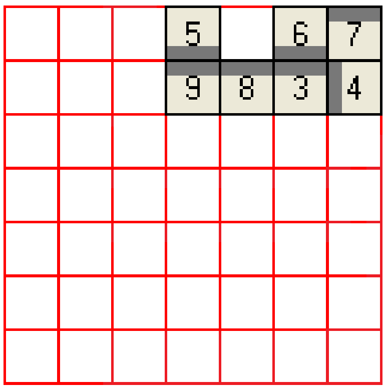
Hereinafter, the experiments for the grouping problem will be presented.

5.1.1 Experiment 1a

As it was stated before, no special shape is required in this experiment. Because of this, all agents involved in the solution of the problem, can share the learned knowledge. This is equivalent to saying that agents are homogeneous. As a result of this, it is not necessary to train all agents. A minimum number can be trained, to later replicate the obtained knowledge in the remaining agents. To establish this number, some incremental experiments are performed. From these experiments it will be concluded that at least three agents are needed to be trained, leaving the remaining seven agents fixed. Replicating the learned knowledge in these seven agents allows solving the grouping problem with a negligible error rate. The seven agents are grouped in two initial distributions to be used during the learning process. These distributions are shown in Fig. 5.5. From now on, the first initial distribution (see Fig. 5.5a) will be named D_1 and the second (see Fig. 5.5b) will be named D_2.



(a) *Initial distribution D_1.*



(b) *Initial distribution D_2.*

Figure 5.5: *Initial distributions of seven agents used during the learning process.*

With regard to sensing capability, the parameter values are found by hand and they are displayed in Table 5.4. According to Eq. 4.5, the number of possible configurations with this parameterization is 1024.

Table 5.4: *Sensing parameter values obtained by trial and error.*

	ρ	θ	ν	β	δ_1	δ_r
Front	2	1	2	2	2	1
Right	2	1	1	2	1	1
Back	2	1	2	2	1	1
Left	2	1	1	2	1	1

Fig. 5.6 shows, graphically, the sensing area obtained with the parameter values displayed in Table 5.4. The dotted line determines the number of layers into which the sensing area of each sensor is divided. The numbers inside them represent the density for each layer.

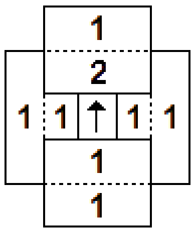
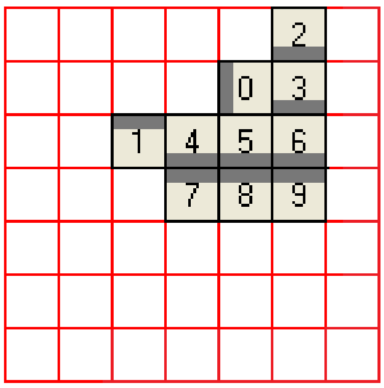


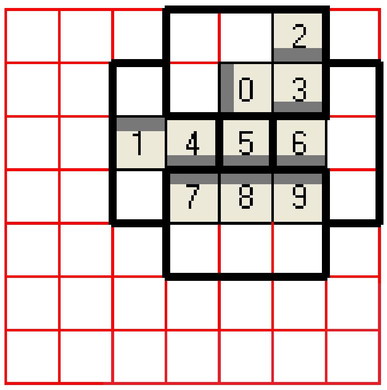
Figure 5.6: *Sensing area using parameter values from Table 5.4.*

Fig. 5.7a shows a possible goal state and Fig. 5.7b shows the sensing area for Agent 5 in that final state. Given the parameter values shown in Table 5.4 the configuration for Agent 5 is "03:03:05:01:1", meaning: there is one agent near, in front (Agent 8) and at least one other agent on either side (Agents 7 or 9); there is one agent near, on the right (Agent 4) and at least one other agent far, on the right (Agent 1); there are at least two more agents near, at the back (Agents 0 or 3) and at least one agent far, at the back (Agent 2); there is one agent near, on the left (Agent 6); the last position in "1" means that all agents have neighbors. Thus, the conclusion is that the goal is reached.

Several number of iterations (i.e. 11200, 22400, 44800 and 56000) are used to test the Algorithm ObIndBe. Each learning process using one of these numbers is named *Series of Learning Process*. It is named series because, for each number of iterations, several learning processes will be performed.



(a) *Possible final state with ten agents.*



(b) *Sensing area for Agent 5.*

Figure 5.7: *Sensing area and grouping examples. Fig. 5.7a shows a possible final state for the grouping problem with ten agents. Fig. 5.7b shows the sensing area for Agent 5 in the state shown in Fig. 5.7a.*

There are some parameters that need to be set before applying the Algorithms. The parameter top_ε, named *decay value*, (see Section 4.2.2) for all ε_{ij} is set according to the number of visits per configuration. It was obtained by experimentation to ensure that the configuration will not only be explored but also exploited.

Four experiments with fifteen learning processes each are carried out using the number of iterations and values of top_ε given in Table 5.5 for each initial distribution, D_1 and D_2.

Table 5.5: *Maximum number of iterations and decay value for parameters ε_{ij}.*

Max. Iteration	**11200**	**22400**	**44800**	**56000**
$\boldsymbol{top_\varepsilon}$	45	90	180	230

After training three agents fifteen times with both initial distributions (D_1 and D_2), 90 Original Policies are obtained for each series (11200, 22400, etc.). Each policy is assigned to agents and tested on its effectiveness starting with agents in random positions. The Individual Policy for each agent is obtained based on the 30 corresponding Original Policies. Then, the General Policy is created, based on the three Individual Policies.

To test the goodness of the policies learned by the agents, these are tested by assigning them as follows:

Orig: Each agent uses its own Original Policy.

Pol 0: All three agents use the Original Policy for Agent 0.

Pol 1: All three agents use the Original Policy for Agent 1.

Pol 2: All three agents use the Original Policy for Agent 2.

It is possible to see in Table 5.6 that, for different Series of Learning Processes, the effectiveness is similar. Observing column D_2 for all series, it can be seen that the effectiveness varies depending on the behavior of the agent used. This means that the knowledge acquired by each agent during training varies among agents and must not be ruled out. This justifies the need for policies which cover everything learned by each agent in each training process.

Table 5.6: *Effectiveness obtained using the Original Policies during testing.*

	11200		**22400**		**44800**		**56000**	
	D_1	D_2	D_1	D_2	D_1	D_2	D_1	D_2
Orig	99.00	94.20	99.00	95.00	99.00	95.00	99.00	95.00
Pol 0	98.64	96.00	99.00	94.00	99.00	97.00	99.00	96.00
Pol 1	96.80	92.00	99.00	96.00	99.00	96.00	99.00	96.00
Pol 2	98.02	93.00	99.00	93.00	99.00	95.00	99.00	96.00

Table 5.7 displays the average of the number of configurations visited during each Series of Learning Processes. It can be seen that, although the number of iterations is increased, the number of configurations visited remains slightly the same among them. This means that, the configurations visited during the series of learning processes are part of the path to the solution. Moreover, the remaining configurations, those that were not visited, are not necessary to reach the goal.

Another interesting value to analyze is the number of steps needed to reach the goal. From now on, these values will be shown as average and standard deviation. In relation to the number of steps needed to reach the solution during testing, the results displayed in Table 5.8 show that they are similar for all series. As is the case with the effectiveness (see Table 5.6), which is slightly the same throughout the series.

Since there are no appreciable differences among series with respect to the number of configurations or number of steps, from here on, 11200 will be taken as the number of iterations in each experiment.

Table 5.7: *Number of configurations visited for each agent in each training series.*

	Agent 0		Agent 1		Agent 2	
	D_1	D_2	D_1	D_2	D_1	D_2
11200	717	820	718	822	717	821
22400	718	822	719	822	719	821
44800	719	822	719	822	719	822
56000	719	822	719	822	719	822

Table 5.8: *Number of steps needed to reach the goal using the Original Policies during testing.*

		Orig		Pol 0		Pol 1		Pol 2	
		Avg	Std	Avg	Std	Avg	Std	Avg	Std
11200	D_1	3.74	2.10	4.13	3.09	3.71	2.97	3.73	2.05
	D_2	3.91	2.43	4.05	2.47	3.91	2.29	3.52	1.75
22400	D_1	3.61	1.83	3.58	1.80	3.78	2.11	3.74	2.09
	D_2	4.12	2.69	3.71	1.99	4.07	2.43	4.20	3.68
44800	D_1	3.84	2.06	3.40	1.81	3.40	1.87	3.53	1.77
	D_2	3.88	2.27	3.91	2.35	3.78	2.24	3.82	2.23
56000	D_1	3.77	1.96	3.63	1.94	3.79	2.10	3.78	2.18
	D_2	4.11	2.43	3.91	2.34	3.91	2.34	3.76	2.25

Table 5.9 shows the effectiveness and the number of steps needed to reach the goal for each initial distribution and each policy. It is interesting to observe that all the generated policies (Individual and General Policies) have an effectiveness similar to the Original Policies.

Table 5.9: *Effectiveness and number of steps needed to reach the goal for 11200 iterations and both initial distributions.*

	Effectiveness		
	Original	Individual	General
D_1	99.00	97.60	97.00
D_2	94.20	95.60	94.60

	Number of steps					
	Original		Individual		General	
	Avg	Std	Avg	Std	Avg	Std
D_1	3.74	2.10	3.78	2.14	3.76	2.20
D_2	3.91	2.43	3.86	2.31	3.92	2.38

It is necessary to know how the General Policy behaves if the number of agents required in the solution of the problem increases and if the dimension of the toroid varies. To answer the first question, ten and fifteen agents are used in a 7×7 toroid. To answer the second question, ten and fifteen agents are used in a 8×8 toroid.

Finally, the General Policy is compared with a Random Policy using ten agents.

Table 5.10 shows the effectiveness and the number of steps needed to reach the goal when the General Policy is used with ten and fifteen agents in a 7×7 and 8×8 toroid.

Table 5.10: *Effectiveness and number of steps obtained when the General Policy is used with ten and fifteen agents in a* 7×7 *discrete toroid and* 8×8 *discrete toroid.*

7 × 7 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	68.50	8.12	4.57
15	66.20	8.91	5.83

8 × 8 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	47.40	9.94	5.72
15	42.33	11.12	6.96

Table 5.11 displays the effectiveness and number of steps needed to reach the goal with ten agents if a Random Policy is used.

Table 5.11: *Effectiveness and number of steps obtained when a Random Policy is used with ten agents in a* 7×7 *discrete toroid.*

Effectiveness	**No. of steps**	
	Avg	**Std**
19.50	289.43	173.52

When the General Policy is used with ten agents, the number of steps required to achieve the goal is slightly increased. It is important to keep in mind that the General Policy is deterministic and there could be situations among the ten agents that might not have appeared during the training processes with only three agents. These situations are solved performing the action of the closest configuration, which may not always be the correct action.

These experiments show that replicating the knowledge learned by three agents in a greater number of agents, does not, appreciably, deteriorate the effectiveness obtained.

5.1.2 Experiment 1b

In this experiment the problem to be solved remains the same as in Experiment 5.1.1. The difference between them is that, in this experiment, the parameters β, δ_1 and δ_r are learned using Algorithm ObsenPar (see Algorithm 3). Agents are trained using Algorithm ObIndBe (see Algorithm 1).

Parameters ρ, θ and ν have the same values as in Experiment 5.1.1. Table 5.12 displays these values and Fig. 5.8 shows the corresponding sensing area.

Since agents are homogeneous and only three out of six parameters must be learned, the chromosome is composed of twelve genes.

Comparative results between both experiments are shown below.

Table 5.12: *Fixed parameters, ρ, θ and ν, used to solve the grouping problem.*

	ρ	θ	ν
Front	2	1	2
Right	2	1	1
Back	2	1	2
Left	2	1	1

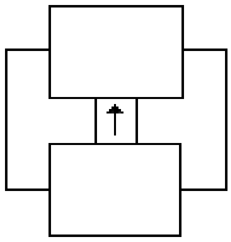


Figure 5.8: *Sensing area using the parameters displayed in Table 5.12.*

Table 5.13a displays the values of β, δ_1 and δ_r for the sensing capability obtained by hand in Experiment 5.1.1, and the values for the sensing capability obtained using the Algorithm ObSenPar in this experiment.

Table 5.14 displays the effectiveness and the number of steps needed to reach the goal when the Original, Individual and General Policies are used.

Table 5.13: *Parameters β, δ_1 and δ_r used in Experiments 5.1.1 and 5.1.2.*

(a) *Values obtained by hand used in Experiments 5.1.1*

	β	δ_1	δ_r
Front	2	2	1
Right	2	1	1
Back	2	1	1
Left	2	1	1

(b) *Values obtained with the proposed method.*

	β	δ_1	δ_r
Front	2	2	1
Right	1	3	1
Back	1	1	1
Left	2	1	1

Table 5.14: *Effectiveness and number of steps needed to reach the goal for 11200 iterations and both initial distributions.*

(a) *Effectiveness for 11200 iterations and both initial distributions.*

	Effectiveness		
	Original	**Individual**	**General**
D_1	99.00	99.00	99.00
D_2	93.00	96.00	96.00

(b) *Number of steps needed to reach the goal.*

	Number of steps					
	Original		**Individual**		**General**	
	Avg	**Std**	**Avg**	**Std**	**Avg**	**Std**
D_1	4.22	2.22	4.49	3.15	4.31	2.41
D_2	4.26	2.52	4.64	3.12	4.53	2.71

Table 5.15 displays effectiveness and number of steps needed to reach the goal when ten and fifteen agents are used in a 7×7 discrete toroid, and the same in a 8×8 discrete toroid.

Table 5.15: *Effectiveness and number of steps obtained when the General Policy is used with ten and fifteen agents in a 7×7 discrete toroid and 8×8 discrete toroid.*

7 × 7 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	80.00	12.03	6.89
15	67.00	8.92	5.01

8 × 8 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	67.80	14.06	8.67
15	55.00	10.90	6.93

It is important to notice the difference in the number of possible configurations in both sensing capabilities, calculated according to Eq. 4.5. Whereas the sensing capability obtained by hand in Experiment 5.1.1 has 1024 configurations, the sensing capability obtained with the proposed method has only 768 configurations. The considerable decrease in the number of configurations does not affect the resolution of the problem. When the Individual and General Policies are compared, they have better effectiveness when the parameters are obtained with the proposed method than those obtained by hand in Experiment 5.1.1 (see Tables 5.9 and 5.14a).

Furthermore, the effectiveness obtained when the manual sensing capability is used for grouping ten agents in a 7×7 discrete toroid is 68.50%, in contrast with an effectiveness of 80% reached when the sensing capability obtained using Algorithm ObsenPar (see Table 5.13b) is used to solve the same problem.

With respect to a Random Policy, Table 5.16 displays the effectiveness and number of steps needed to reach the goal. It can be observed in the tables that effectiveness decreases considerably, while the number of steps to reach the solution increases substantially when a Random Policy is used.

5.1.3 Experiment 1c

This experiment is divided into three phases. In all of them the six sensing parameter values are found using Algorithm ObSenPar, but they differ in the number of agents used to be trained and the policies that will be obtained.

Table 5.16: *Effectiveness and number of steps needed to reach the goal with a Random Policy using the sensing parameter values displayed in Tables 5.12 and 5.13b.*

Effectiveness	**No. of steps**	
	Avg	**Std**
22.00	287.15	177.04

In Phase 1, a Deterministic Policy is sought using three agents and initial conditions. Phases 2 and 3 are carried out to see what will happen if there is no possibility to find initial distributions to train agents. With this idea in mind and considering what was exposed at the beginning of Experiment 1, five agents will be used. Using less then five agents, during training, has the drawback that there will be configurations that will never appear during this process, making it impossible for the agents to learn how to behave in those situations.

Another question to be answered is: knowing that the effective policy in a stochastic environment is a Stochastic Policy, is it possible to find a Homogeneous Stochastic Policy to solve this kind of problem using the proposed methods? And, if it is possible, it is interesting to compare results with a Deterministic Policy in the same conditions.

5.1.3.1 Phase 1

In this case all six parameters are obtained using three agents, for these reason the chromosome has 24 genes.

Table 5.17 displays the sensing parameter values obtained with Algorithm ObSenPar. Fig 5.9 shows the sensing area obtained with these parameters. The number of possible configurations with these values is 512.

Table 5.17: *Parameter values obtained with Algorithm ObSenPar to solve the grouping problem automatically.*

	ρ	θ	μ	β	δ_1	δ_r
Front	2	1	1	1	2	1
Right	2	2	1	1	1	1
Back	1	0	1	1	1	1
Left	2	1	-1	2	3	1

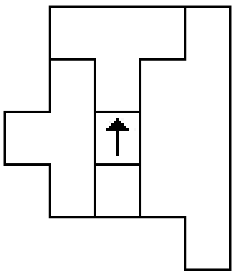


Figure 5.9: *Sensing area using the parameters shown in Table 5.17.*

Table 5.18 displays the effectiveness and the number of steps needed to reach the goal when the Original, Individual and General Policies are used.

Table 5.18: *Effectiveness and number of steps needed to reach the goal with three agents and both initial distributions.*

(a) *Effectiveness reached using three agents and both initial distributions.*

	Effectiveness		
	Original	**Individual**	**General**
D_1	88.20	93.80	94.60
D_2	87.80	91.20	93.00

(b) *Number of steps needed to reach the goal.*

	Number of steps					
	Original		**Individual**		**General**	
	Avg	**Std**	**Avg**	**Std**	**Avg**	**Std**
D_1	8.00	6.81	7.70	5.37	7.97	5.81
D_2	5.94	4.28	6.85	4.87	6.87	4.91

Table 5.19 displays effectiveness and number of steps needed to reach the goal when ten and fifteen agents are used in a 7×7 discrete toroid, and the same in a 8×8 discrete toroid.

It can be seen in Table 5.20 that, using a Random Policy instead of the learned ones decreases the effectiveness considerably, with an important increase in the number of steps needed to reach the goal.

Table 5.19: *Effectiveness and number of steps obtained when the General Policy is used with ten and fifteen agents in a 7×7 discrete toroid and 8×8 discrete toroid.*

7×7 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	82.20	28.93	22.29
15	82.60	17.66	12.19

8×8 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	70.60	37.58	32.69
15	68.20	22.30	22.46

Table 5.20: *Effectiveness and number of steps needed to reach the goal with a Random Policy.*

Effectiveness	**No. of steps**	
	Avg	**Std**
21	292.32	168.98

5.1.3.2 Phase 2

Because the initial distributions used in the previous experiments restrict the number of configurations in which the agents can be found during the learning processes, it is interesting to test what will happen if the agents are trained without these initial distributions. Considering that the lower the number of agents, the lower the number of configurations, and this, in turn, influences the subsequent generalization to a greater number of agents, the number of agents must be increased for these experiments. With an even number of agents facing situations are more possible than with an odd number of them Therefore, in this case a Deterministic Policy is sought by training five agents without initial distributions.

Table 5.21 displays the sensing parameter values and Fig 5.10 shows the shape of this sensing area. The number of possible configurations with this parameterization is 512.

Tables 5.22 show the effectiveness and number of steps needed to reach the goal when the Individual and General Policies are used with five agents and initial

Table 5.21: *Parameter values obtained with Algorithm ObSenPar to solve the grouping problem automatically.*

	ρ	θ	μ	β	δ_1	δ_r
Front	3	1	-2	2	2	2
Right	4	1	2	2	1	1
Back	1	0	1	1	1	1
Left	1	0	1	1	1	1

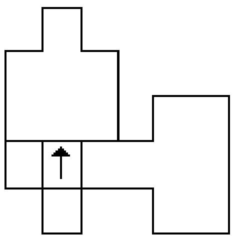


Figure 5.10: *Sensing area using the parameters shown in Table 5.21.*

distributions.

Table 5.22: *Effectiveness and number of steps needed to reach the goal using the corresponding Individual and General Policies and initial distributions.*

(a) *Effectiveness reached using five agents and both initial distributions.*

	Effectiveness	
	Individual	**General**
D_1	88.60	88.50
D_2	70.20	70.25

(b) *Number of steps needed to reach the goal.*

	Number of steps			
	Individual		**General**	
	Avg	**Std**	**Avg**	**Std**
D_1	11.95	9.77	12.06	10.51
D_2	13.55	11.86	12.79	10.85

Table 5.23 displays effectiveness and number of steps needed to reach the goal when ten and fifteen agents are used in a 7×7 discrete toroid, and the same in a 8×8 discrete toroid.

Table 5.23: *Effectiveness and number of steps obtained when the General Policy is used with ten and fifteen agents in a 7×7 discrete toroid and 8×8 discrete toroid.*

7×7 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	92.20	47.86	44.02
15	89.20	19.29	18.71

8×8 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	89.20	75.60	70.66
15	84.40	21.50	22.87

These tables also show that, despite having trained only five agents without any initial distribution, the problem can solved with an effectiveness greater than 80% using the General Policy with ten and fifteen agents with the same sensing capability as the trained agents. Although there is a slightly lower effectiveness when Individual and General Policies are used with the five trained agents than then obtained in Experiment 1b, the effectiveness reached during generalization is considerably higher (see Tables 5.19 and 5.23).

The effectiveness obtained using the General Policy to group 10 agents is substantially greater than that obtained when a Random Policy is used (see Table 5.24).

Table 5.24: *Effectiveness and number of steps needed to reach the goal with a Random Policy using the sensing parameter values displayed in Table 5.21.*

Effectiveness	**No. of steps**	
	Avg	**Std**
1	396.68	231.31

5.1.3.3 Phase 3

To this end, a Stochastic Policy is sought, and only a General Policy will be used. The characteristics of the experiment are the same as in Phase 2.

Table 5.25 displays the sensing parameter values obtained with Algorithm ObSenPar and Fig 5.11 shows the shape of the corresponding sensing area. The number of possible configurations is 768.

Table 5.25: *Parameter values obtained when solving the grouping problem with five agents using a Stochastic Policy.*

	ρ	θ	μ	β	δ_1	δ_r
Front	3	2	3	1	4	1
Right	2	0	2	1	1	1
Back	3	1	-1	2	1	1
Left	4	1	1	1	3	1

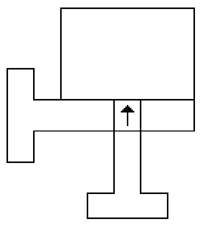


Figure 5.11: *Sensing area using the parameters shown in Table 5.25.*

Because agents are homogeneous it is necessary to obtain an Homogeneous Stochastic Policy with which to test generalization for a greater number of agents. The Homogeneous Stochastic Policy is obtained combining all the Stochastic Policies collected for each agent throughout all training processes. The way to do this is to accumulate the corresponding values of matrix Q for each configuration and each action for all agents throughout all training processes. Although the behaviors are obtained training agents without initial distributions, the policy is tested with them. Table 5.26 shows the effectiveness and number of steps needed to reach the goal.

Table 5.26: *Effectiveness and number of steps needed to reach the goal using the Homogeneous Stochastic Policy and initial distributions.*

	Effectiveness	**No. of steps**	
		Avg	**Std**
D_1	99.99	26.60	20.90
D_2	99.99	38.12	31.69

To test generalization ten and fifteen agents are used in a 7×7 and 8×8 toroid. Table 5.27 shows the effectiveness and the number of steps needed to reach the goal in those cases.

Table 5.27: *Effectiveness and number of steps obtained when the General Policy is used with ten and fifteen agents in a* 7×7 *discrete toroid and* 8×8 *discrete toroid.*

7 × 7 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	99.99	79.82	73.83
15	99.99	39.14	35.83

8 × 8 toroid			
No. of Agents	**Effectiveness**	**No. of steps**	
		Avg	**Std**
10	88.60	229.28	185.93
15	98.40	142.94	140.74

The next step is to use a Random Policy with ten agents to compare with the effectiveness obtained with the learned policy. Table 5.28 displays the effectiveness and the number of steps needed to reach the goal. As in all experiments so far, the Random Policy has a very low effectiveness regarding the effectiveness achieved when the policy obtained with the Algorithm ObIndBe is used.

Table 5.28: *Effectiveness and number of steps needed to reach the goal for ten and fifteen agents using a Random Policy.*

Effectiveness	**No. of steps**	
	Avg	**Std**
21.00	396.50	230.47

5.1.4 Results

Previous experiments show that the Algorithm ObIndBe is able to find deterministic behavior for a small group of agents that allow grouping them anywhere within a discrete toroid, replicating this knowledge to the remaining untrained agents that are needed to reach the goal, with a good effectiveness.

These experiments show that for homogeneous behavior, it is possible to find a sensing capability and deterministic behaviors using Algorithms ObSenPar and

ObBinDe proposed in this thesis.

With respect to the Stochastic Policy, there are two aspects to consider when five agents are trained:

1. The effectiveness reached is better than in other kinds of policies, deterministic or random.

2. The number of steps needed to reach the goal increases with respect to a Deterministic Policy, but it is lower compare to a Random Policy.

Regarding fault tolerance, the testing showed that agents can be added without any need for retraining. Therefore, if some agents fail in solving the problem, it is only necessary to replace them with other agents.

5.2 Experiments with square shape

In these experiments Algorithms ObIndBe and ObSenPar are used to find individual behaviors and sensing capabilities that allow forming a square shape as shown in Fig. 5.12. In this particular shape, position numbers and agent numbers are equivalent and are used interchangeably throughout these experiments. Taking into account the position they have in the goal shape, it is possible to distinguish two classes of agents:

- The four corners (positions 0, 2, 6 and 8), named *set of even agents.*

- The four remaining places (positions 1, 3, 5 and 7), named *set of odd agents.*

Agent 4 is used as a seed to determine where the desired shape will be formed. For this reason, this agent is neither trained nor tested.

Intuitively, agents that must occupy the corners, members of the set of even agents, should have the same capabilities; in other words, they can used the same sensing capability and behavior. The same would seem to happen with agents in the odd class, but not among agents in different classes. That means that at least one agent of each class should be trained and their sensing capabilities must be found. These experiments prove that differentiation of behaviors and different sensing capability are needed, and the proposed methods find them.

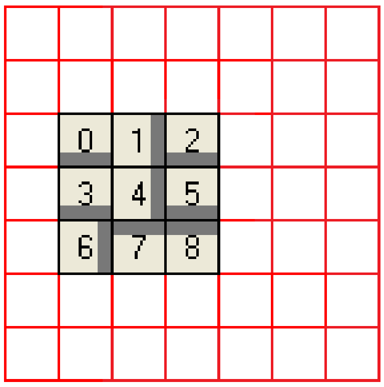


Figure 5.12: *Shape to be reached in Experiment 2. Each position in the objective shape has been enumerated from 0 to 8 in consecutive order. Position 4 determines where the shape has to be formed in the toroid.*

5.2.1 Experiment 2a

In this part of the experiment three out of six parameters are obtained with the Algorithm ObSenPar. The experiment is divided into two phases:

Phase 1: Obtaining the sensing capabilities and behaviors for the Agents 0, 2, 5 and 7. Notice that two of each class of agents are used. Only three parameters will be learned and, therefore, the chromosome used will be compose of 48 genes (three parameters, four sensors, four agents). Parameters ρ, θ and ν are fixed at the beginning of the experiments; Table 5.29 displays the values of these parameters for each sensor, and Fig. 5.13 shows the sensing area obtained with these values.

Phase 2: Obtaining the sensing capabilities and behaviors for Agents 0 and 7, one of each class of agents. These agents are chosen analyzing the results of experiments in Phase 1 in Section 5.2.1.1. In this case the chromosome is compose of 48 genes (six parameters times four sensors times two agents).

Here, it is not possible to use a Deterministic Policy because the number of possible final states, with distinguishable agents, is only one. So, if this is the case, and considering a minimal sensing capability, the probability of conflictive perceptual aliasing is too high. A Deterministic Policy that takes bad decisions in an intermediate state, can leave the system in an unsolved situation, as mention in Section 4.1. For those reasons, a Stochastic Policy will be used (see Section 3.1.2).

Table 5.29: *Parameters ρ, θ and ν for Experiment 5.2.1.*

	ρ	θ	ν
Front	4	1	-1
Right	2	1	1
Back	2	1	2
Left	2	1	1

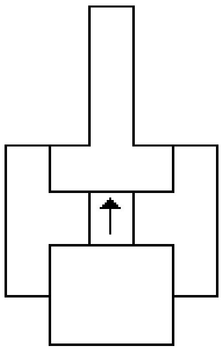


Figure 5.13: *Sensing area using the parameters displayed in Table 5.29.*

5.2.1.1 Phase 1

Tables 5.30 displays the sensing parameter values obtained with the Algorithm ObSenPar. Table 5.31 displays the number of possible configurations for each parameterization, according to Eq. 4.5. With these parameterizations the effectiveness obtained is 6%. Naturally, a question arises when such low effectiveness is obtained: which are the agents that contributed to such low value?

To answer it, individual agents are tested. That is, nine agents are located in the toroid in such a way that they form an incomplete square (for example, to test Agent 0, the nine agents form a square with an empty Position 0). Table 5.32 displays the effectiveness for each test. It can be seen that, the values obtained by Agents 2 (63%) and 5 (64%) are lower than those obtained by Agents 0 (97%) and 7 (92%). This means that Agents 2 and 5 did not learn properly and disturb the total system performance.

In the goal shape, Agents 0 and 2 are members of the set of even agents. Because Agent 0 obtained better effectiveness than Agent 2, the next step is to replicate the knowledge learned by Agent 0 (sensing capability and behavior) in Agent 2. The same is made with Agent 5 and 7, both members of the set of odd agents, so, Agent 5, in this step, uses the knowledge for Agent 7. Table 5.33

Table 5.30: *Parameter values for Agent 0, 2, 5 and 7, obtained with the proposed methods.*

(a) *Parameter values for Agent 0 obtained with the proposed methods.*

Agent 0	β	δ_1	δ_r
Front	2	2	1
Right	2	1	1
Back	2	3	1
Left	1	2	1

(b) *Parameter values for Agent 2 obtained with the proposed methods.*

Agent 2	β	δ_1	δ_r
Front	4	2	1
Right	2	1	1
Back	1	4	1
Left	2	1	1

(c) *Parameter values for Agent 5 obtained with the proposed methods.*

Agent 5	β	δ_1	δ_r
Front	2	2	1
Right	1	2	1
Back	2	1	2
Left	2	1	1

(d) *Parameter values for Agent 7 obtained with the proposed methods.*

Agent 7	β	δ_1	δ_r
Front	4	3	1
Right	2	1	2
Back	1	4	1
Left	1	2	1

Table 5.31: *Number of possible configurations for the sensing parameter values displayed in Table 5.30.*

Agent	**0**	**2**	**5**	**7**
Number of Configurations	3072	8192	2048	24576

Table 5.32: *Effectiveness for policies and sensing capabilities obtained with the proposed methods.*

Agent	**0**	**2**	**5**	*7*
Effectiveness	97	63	64	92

displays the effectiveness obtained when the knowledge for Agent 0 and Agent 7 is replicated in Agent 2 and Agent 5, respectively. The values obtained by Agents 2 (95%) and 5 (86%) increased with respect to the values obtained when their own knowledge was used (see Table 5.32).

To complete the experiments, the remaining agents - those who were not trained - are tested using the knowledge of the trained agent from its own class (i.e., Agents 6 and 8 use the knowledge learned by Agent 0 and Agents 1 and 3 use the knowledge learned by Agent 7.). Table 5.34 displays the effectiveness for

Table 5.33: *Effectiveness for Agents 2 and 5 using knowledge learned by Agents 0 and 7, respectively.*

Agent	**2**	**5**
Effectiveness	95	86

Agents 1, 3, 6 and 8.

Table 5.34: *Effectiveness obtained for non trained agents using the knowledge of the trained agent in its own class.*

Agent	**1**	**3**	**6**	**8**
Effectiveness	90	89	95	94

It is worth mentioning that, in this particular experiment, the underlying idea is to prove that differentiation of behaviors is necessary and, therefore, the proposed methods in this thesis allow reaching this goal. A new test is made to verify differentiation: Agent 0 must go to Position 0 in the square shape but using the knowledge learned by Agent 7 and Agent 7 must go to Position 7 but using the knowledge learned by Agent 0. Table 5.35 displays the effectiveness obtained in this case. It can be observed that both effectiveness decreased compared with the values obtained using their own knowledge (see Table 5.32). However, the worst value is that obtained by Agent 7, 4.00%. It is possible to explain such low value by paying attention to the sensing capabilities and number of possible configurations for both agents. The number of possible configurations for Agent 0 (see Table 5.31) is smaller than those for Agent 7. This results in decreasing the ability to distinguish situations that Agent 7 could differentiate when he used his own knowledge.

Table 5.35: *Effectiveness obtained when testing differentiation. Column 0 shows the effectiveness for Agent 0 going to Position 0 but using the knowledge learned by Agent 7. Column 7 shows the effectiveness for Agent 7 going to Position 7 but using the knowledge learned by Agent 0.*

Agent	**0**	**7**
Effectiveness	82	4

The reason that even agents had better performance than odd ones can be understood by analyzing the sensing parameters in Tables 5.30. It is necessary to remember that, the greater the number of possible configurations, the more likely the sensing capability can be successfully used in any position of the target shape. The problem is that the greater the number of configurations, the larger the state space, with its corresponding learning problems. (see [30]).

Table 5.36 shows the effectiveness obtained when agents are tested individually. To do that, the knowledge learned by Agent 0 is used for even agents, and the knowledge learned by Agent 7 is used for odd agents.

Table 5.36: *Effectiveness for each Agent using information from Agent 0 and 7.*

Agent	0	**1**	**2**	**3**	**5**	**6**	**7**	**8**
Effectiveness	97.00	89.00	95.00	89.0	86.0	95.00	92.00	94.00

All nine agents are tested leaving Agent 4 docked as a seed using knowledge for Agent 0 in the even agents, and knowledge for Agent 7 in odd agents. The effectiveness obtained is 52.00% grater than the obtained when four agents were trained.

Taking these values into consideration, it is possible to conclude that it is enough to train as many agents as distinguishable positions there can be found in the goal shape.

5.2.1.2 Phase 2

For the reason explained in Phase 1 in Section 5.2.1.1 Agent 0 and 7 are selected to be trained using the proposed methods. The methods give two different sensing capabilities, one for Agent 0 and a distinct one for Agent 7, both are displayed in Table 5.37.

Table 5.37: *Parameter values obtained for Agents 0 and 7 with the proposed methods.*

(a) *Parameter values obtained for Agent 0.*

Agent 0	β	δ_1	δ_r
Front	4	2	1
Right	1	1	1
Back	2	2	1
Left	1	1	2

(b) *Parameter values obtained for Agent 7.*

Agent 7	β	δ_1	δ_r
Front	4	3	1
Right	2	1	2
Back	1	4	1
Left	1	2	1

Table 5.38 displays the number of possible configurations for each sensing parameter values in Table 5.37. Although the sensing area is the same for both agents, the number of possible configurations is different because the number of layers and densities varies among them.

Table 5.38: *Number of possible configurations for each sensing parameter values shown in Table 5.37.*

Agent	**0**	**7**
Number of Configurations	5096	30576

The effectiveness obtained after training both agents with these sensing parameter values using Algorithm ObIndBe, is 82%.

All the tests carried out in Phase 1 in Section 5.2.1.1 are repeated in this phase using the knowledge gathered for both agents, keeping in mind that only Agent 0 and 7 are trained. Table 5.39 displays the effectiveness obtained using the policy and sensing capability from Agent 0 for the even positions and the policy and sensing capability for Agent 7 for odd positions, when agents are tested.

Table 5.39: *Effectiveness for each Agent. Agents belonging to the set of even agents are tested using knowledge learned by Agent 0. Agents belonging to the set of odd agents are tested using knowledge learned by Agent 7.*

Agent	**0**	**1**	**2**	**3**	**5**	**6**	**7**	**8**
Effectiveness	99.00	92.00	100.00	86.00	88.00	98.00	87.00	99.00

When all agents are put to work together, with the criteria explained in the previous paragraph, the effectiveness obtained is 56%, that is greater than the effectiveness obtained with the information gathered in Phase 1 in Section5.2.1.1, that is 52.00%.

5.2.2 Experiment 2b

As in Experiment 2 Phase 2 in Section 5.2.1.2, Agents 0 and 7 are used in these experiments, but all six parameter values are learned in this case. For this reason, the chromosome length is 48 (six parameters, four sensors, two agents).

Table 5.40 displays the sensing capabilities resulting from the Algorithm ObSenPar for both agents. With these sensing capability parameter values, the number of possible configurations is 3072 for Agent 0 and 4608 for Agent 7, as displayed in Table 5.41.

Table 5.40: *Parameter values obtained for Agent 0 and 7 in Experiment 2b.*

Agent 0	ρ	θ	μ	β	δ_1	δ_r
Front	1	1	1	1	2	1
Right	4	2	2	1	5	1
Back	2	1	1	2	1	1
Left	2	0	2	1	1	1

Agent 7	ρ	θ	μ	β	δ_1	δ_r
Front	3	1	2	2	1	1
Right	1	1	1	1	3	1
Back	2	0	2	1	1	1
Left	1	0	1	1	1	1

Table 5.41: *Number of possible configurations for each sensing parameter values shown in Table 5.40.*

Agent	**0**	**7**
Number of Configurations	1280	192

Table 5.42 displays the effectiveness and number of steps needed to reach the goal when Agents 0 and 7 are tested with the seven remaining agents fixed as in Fig 5.14.

Table 5.42: *Effectiveness and number of steps needed to reach the goal when Agents 0 and 7 are tested with the seven remaining agents fixed.*

Effectiveness	**No. of steps**	
	Avg	**Std**
96.80	108.70	73.35

The next test is carried out as follows: the behavior for Agent 0 is used by Agents 2, 4 and 8, and the behavior of Agent 7 is used by Agents 1, 3 and 5. The nine agents are tested in a 7×7 and 8×8 toroide. Table 5.43 displays the effectiveness and number of steps in these cases.

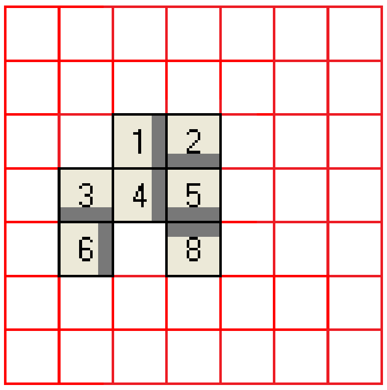


Figure 5.14: *Seven agents fixed.*

Table 5.43: *Effectiveness and number of steps needed to reach the goal when nine agents are tested in a 7×7 and 8×8 toroid.*

Toroid dimension	**Effectiveness**	**No. of steps**	
		Avg	**Std**
7×7	90.24	253.78	92.65
8×8	76.88	290.51	98.16

It can be seen that the values have improved compared with those obtained in Experiment 2a.

Table 5.44 displays the effectiveness and number of steps when a Random Policy is used.

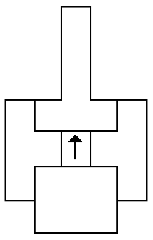
Table 5.44: *Effectiveness and number of steps needed to reach the goal when a Random Policy is used.*

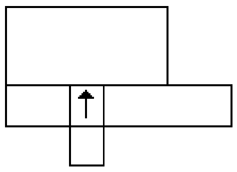
Effectiveness	**No. of steps**	
	Avg	**Std**
6	427.24	62.27

5.2.3 Results

The effectiveness obtained in Experiment 2b has a remarkable improvement with respect to those obtained in Experiment 2a where four agents are used. The number of configurations for Agents 0 and 7 are higher than that which are obtain in Experiment 2a. It can be explained considering that Agent 0 must taking into account configurations that Agent 2 considered and the same happens with Agents 7 and 5.

Then, it is possible to compare the sensing area for the sensing capability for each trained agent in both experiments. It is important to point out that, for Experiment 2a the area is set manually, being the same for all agents, and it is shown in Fig. 5.15a; whereas, in Experiment 2b the sensing area for the sensing capability for each trained agent is learned using the proposed methods and they are shown in Figs. 5.15b and 5.15c.


(a) *Sensing area for agents in Experiment 2a.*


(b) *Sensing area for Agent 0 in Experiment 2b.*

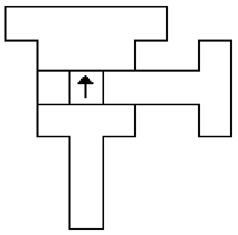

(c) *Sensing area for Agent 7 in Experiment 2b.*

Figure 5.15: *Sensing area for agents in Experiments 2a and 2b.*

Another remarkable point to taking into account is that when all parameter values are learned by Algorithm ObSenPar, the number of possible configurations decrease considerably. Even so, the effectiveness increases.

5.3 Testing proposed methods

In this section two shapes are used to test the proposed methods. All the six parameters are obtained using Algorithm ObSenPar. Different classes of agents are detected and an example of each is used to be trained using Algorithm ObIndBe. The first is a triangular shape and the second is a diagonal shape.

5.3.1 Triangular shape

In this experiment the proposed methods are applied to find the triangular formation shown in Fig. 5.16. As in the previous experiments, a Stochastic Policy is used. The reasons are the same as those explained in Section 5.2, for the square shape.

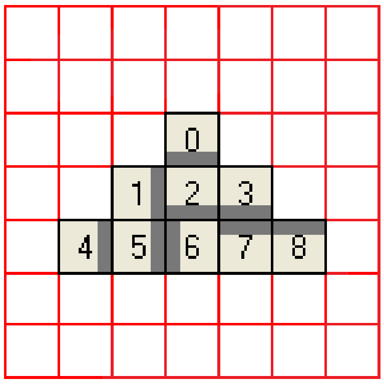


Figure 5.16: *Triangular shape in an* 7 × 7 *toroid whit nine agents.*

In this particular shape it is possible to distinguish 4 classes of agents:

- The first class formed by Agents 0, 4 and 8, named *corner agents.*
- The second class formed by Agents 1 and 3, named *side agents.*
- The third class formed by Agents 5 and 7, named *base agents.*
- The fourth class formed by Agent 6, named *middle agent.*

One agent from each class is used to be trained, in particular, Agents 0, 1, 6 and 7, are chosen. In this experiment the chromosome length is 96 (six parameters, four sensors, four agents). The sensing capability parameter values found with ObSenPar are displayed in Table 5.45. Table 5.46 displays the number of possible configurations for the obtained sensing capabilities for each trained agent.

Table 5.47 displays effectiveness and number of steps needed to reach the goal when Agents 0, 1, 6 and 7 are testing (the five remaining agents are fixed as shown in Fig 5.17).

The next test is carried out by replicating the knowledge obtained during training into the remaining agents in the corresponding class. Table 5.48 displays the effectiveness and number of steps in these cases.

Table 5.45: *Parameter values obtained for Agent 0, 1, 6 and 7 in Experiment 5.3.1.*

Agent 0	ρ	θ	μ	β	δ_1	δ_r
Front	3	1	1	2	1	1
Right	3	2	2	2	1	1
Back	3	0	3	3	1	1
Left	1	1	1	1	1	1

Agent 1	ρ	θ	μ	β	δ_1	δ_r
Front	4	1	1	3	1	1
Right	1	1	1	1	1	1
Back	1	0	1	1	1	1
Left	4	2	-1	4	2	1

Agent 6	ρ	θ	μ	β	δ_1	δ_r
Front	3	0	3	3	1	1
Right	2	1	2	1	1	1
Back	2	0	2	1	1	1
Left	3	1	2	1	2	1

Agent 7	ρ	θ	μ	β	δ_1	δ_r
Front	5	0	5	5	1	1
Right	2	1	1	1	1	1
Back	1	0	1	1	1	1
Left	4	3	2	1	9	1

Table 5.46: *Number of possible configurations for each agent for the obtained sensing capability in Experiment 5.3.1.*

Agent	**0**	**1**	**6**	**7**
Number of Configurations	512	2048	256	4608

Table 5.47: *Effectiveness and number of steps needed to reach the goal when Agents 0, 1, 6 and 7 are tested with the five remaining agents fixed as shown in Fig5.17.*

Effectiveness	**No. of steps**	
	Avg	**Std**
81.30	201.28	90.14

Table 5.49 displays the effectiveness and number of steps when a Random Policy is used.

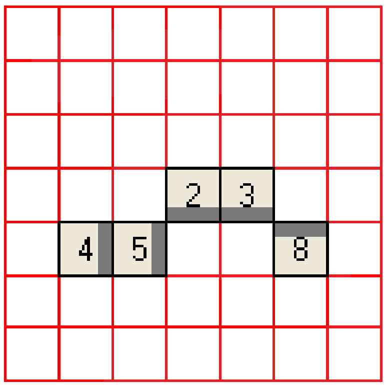


Figure 5.17: *Five agents fixed.*

Table 5.48: *Effectiveness and number of steps needed to reach the goal when nine agents are tested in a* 7×7 *and* 8×8 *toroid.*

Toroid dimension	Effectiveness	No. of steps	
		Avg	Std
7×7	84.60	368.44	157.16
8×8	63.50	453.06	167.56

Table 5.49: *Effectiveness and number of steps needed to reach the goal when a Random Policy is used.*

Effectiveness	No. of steps	
	Avg	Std
12.00	397.61	75.89

5.3.2 Diagonal shape

In this experiment the proposed methods are applied to find the diagonal formation shown in Fig. 5.18. As in the previous experiments, a Stochastic Policy is used. The reasons are the same as those explained in Section 5.2, for the square shape.

In this particular shape it is possible to distinguish 4 classes of agents:

- The first class formed by Agents 0, named *top agent.*
- The second class formed by Agents 1, 3, 5 and 7, named *right agents.*
- The third class formed by Agents 2, 6 and 8, named *left agents.*
- The fourth class formed by Agent 9, named *bottom agent.*

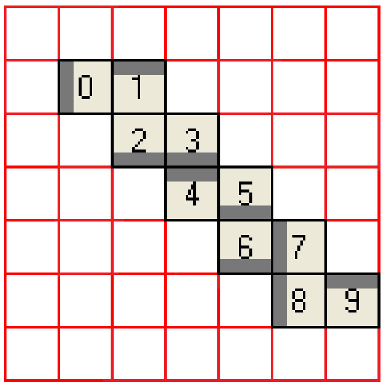


Figure 5.18: *Diagonal shape in an 7×7 toroid whit ten agents.*

Agent 4 is used as a seed to determine where the desired shape will be formed. For this reason, this agent is neither trained nor tested. One agent from each class is used to be trained, in particular, Agents 0, 3, 6 and 9, are chosen. In this particular case the chromosome has 96 genes (six parameters, four sensors, four agents).

The sensing capability parameter values found with ObSenPar are displayed for each agent in Table 5.50. Table 5.51 displays the number of possible configurations for the obtained sensing capabilities for each trained agent.

Table 5.52 displays effectiveness and number of steps needed to reach the goal when Agents 0, 3, 6 and 9 are testing (the six remaining agents are fixed as shown in Fig 5.19).

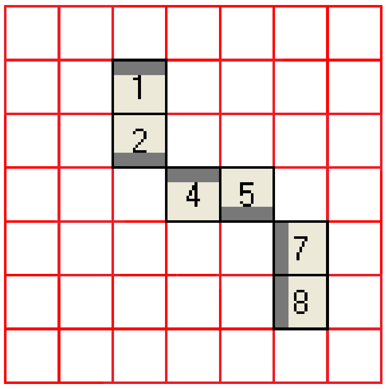


Figure 5.19: *Six agents fixed.*

The next test is carried out by replicating the knowledge obtained during training into the remaining agents in the corresponding class. Table 5.53 displays the effectiveness and number of steps in these cases.

Table 5.50: *Parameter values obtained for Agent 0, 3, 6 and 9 in Experiment 5.3.2.*

Agent 0	ρ	θ	μ	β	δ_1	δ_r
Front	4	0	4	3	1	1
Right	2	0	2	2	1	1
Back	2	0	2	1	1	1
Left	1	1	1	1	1	1

Agent 3	ρ	θ	μ	β	δ_1	δ_r
Front	1	0	1	1	1	1
Right	2	0	2	1	1	1
Back	5	0	5	2	1	1
Left	1	0	1	1	1	1

Agent 6	ρ	θ	μ	β	δ_1	δ_r
Front	1	0	1	1	1	1
Right	3	1	2	2	1	2
Back	4	0	4	3	1	1
Left	1	0	1	1	1	1

Agent 9	ρ	θ	μ	β	δ_1	δ_r
Front	2	0	2	1	1	1
Right	1	0	1	1	1	1
Back	1	0	1	1	1	1
Left	2	0	2	2	1	1

Table 5.51: *Number of possible configurations for each agent for the obtained sensing capability in Experiment 5.3.2.*

Agent	**0**	**3**	**6**	**9**
Number of Configurations	256	64	512	64

Table 5.52: *Effectiveness and number of steps needed to reach the goal when Agents 0, 3, 6 and 9 are tested with the six remaining agents fixed as shown in Fig5.19.*

Effectiveness	**No. of steps**	
	Avg	**Std**
97.40	215.01	117.81

Table 5.54 displays the effectiveness and number of steps when a Random Policy is used.

Table 5.53: *Effectiveness and number of steps needed to reach the goal when nine agents are tested in a* 7×7 *and* 8×8 *toroid.*

Toroid dimension	**Effectiveness**	**No. of steps**	
		Avg	**Std**
$\mathbf{7 \times 7}$	91.60	310.83	115.93
$\mathbf{8 \times 8}$	78.80	356.35	119.62

Table 5.54: *Effectiveness and number of steps needed to reach the goal when a Random Policy is used.*

Effectiveness	**No. of steps**	
	Avg	**Std**
13.00	396.85	70.70

5.3.3 Results

Experiments in this section show that the proposed methods allow dealing with different emergent behaviors. In other words, they allow learning individual behaviors and sensing capabilities even when differentiation of behaviors is required.

Chapter 6

Conclusions and Future Work

This work has introduced the methods ObIndBe and ObSenPar which efficiently allow finding individual behaviors and the necessary sensing capability for a group of agents in a multi-agent system, solving problems related to pattern formation, seen as emergent behaviors of that system. Additionally, the Algorithm ExpDir, which is embedded in ObIndBe, has been presented. This last algorithm allows the agents to explore the environment (even at an advanced stage of the learning processes) taking into account two problematic characteristics to deal with: perceptual aliasing and non-determinism.

It was experimentally demonstrated that the methods (ObIndBe and ObSenPar), working together, could learn homogeneous and heterogeneous behaviors with their corresponding sensing capability. An aspect to point out, is that it was only necessary to find behaviors for a few agents (one representative per class of agents) and then use these behaviors with the remaining agents to complete the task successfully.

It is important to notice that, although all agents in all experiments were trained using Algorithm ObIndBe, performances differed among experiments of the same type (grouping, square shape, triangular shape, diagonal shape). What was different among them, was their sensing capabilities. It was necessary to develop a method that allows finding certain sensing capabilities to solve the proposed problems automatically. The Algorithm ObSenPar introduced here, allowed this, improving not only the performance, but also minimizing the number of possible configurations. This happened both for homogeneous agents (grouping) as for heterogeneous agents (square, triangular or diagonal shapes).

It is well known that, when perceptual aliasing is present the best behavior to use is a Stochastic Policy. In spite of that, in some particular cases a Deterministic and a Stochastic Policy were found. When agents are homogeneous and indistinguishable, if there is no requirement on a particular geometric shape (any shape with agents touching side by side among them is a goal), the number of possible solutions is very high. Even if conflictive perceptual aliasing is present (given by the minimum sensing capability imposed on the agents) it appears that, Deterministic Policy decisions can also lead to a state solution. When agents are heterogeneous and distinguishable, requiring a particular geometric shape, a single goal state exists. Furthermore, if the sensing capability presents conflictive perceptual aliasing, a bad decision performed by a Deterministic Policy can lead the agents into unsolvable situations making it impossible to reach a desired solution. In these cases, a Stochastic Policy must be used even when the number of steps needed to reach the goal increases. This kinds of policies ensure that agents, at some point in their evolution, reach the required pattern. That is to say that, the probability to get a sensing capability with a minimal number of possible configurations without conflictive perceptual aliasing is very low. Therefore, it is highly unlikely that a Deterministic Policy can solve the problem efficiently. In other words, when only one goal state exists, the system is more sensitive to conflictive perceptual aliasing. For that reason, a Deterministic Policy could never find that solution. However, in the case where many goal states are possible, being the system less sensitive to bad sensing capabilities, a Deterministic Policy can work properly even if bad decisions were taken in early states, because another final state can be reached. In the clustering problem, it could be seen that the policies learned by each agent, using Algorithm ObIndBe, could not be rejected, neither by Deterministic nor Stochastic Policies. Although it was empirically demonstrated that a Deterministic Policy can be found when there are many possible solutions to the problem, and in particular when the agents involved in the problem are homogeneous, a Stochastic Policy was obtained under the same conditions giving a better effectiveness than the Deterministic Policy (see Experiment 1c). This justified the need for policies which cover everything learned by each agent in each training process, resulting in the creation of Individual and General policies in the deterministic case, and Homogeneous Stochastic Policies in the stochastic case.

The experiments showed that the Algorithm ObIndBe was able to find deterministic behavior for a small group of agents that allows grouping anywhere within a discrete toroid. To reach the goal it was only necessary to replicate this knowledge in the remaining untrained agents.

At the end of these series of experiments involving grouping agents, it could be seen that it was not necessary to train agents starting with initial distributions. Five of them were trained to obtain sensing capability and behaviors with total success in the effectiveness. It is worth mentioning that, in no case was there any need to train all the agents involved in the solution of the problem. Moreover, less than a half of the agents were trained to obtain behaviors and sensing capability.

When a square or a triangular shape is sought, not only were different behaviors found but different sensing capabilities were obtained. It was only necessary to detect classes of agents, depending on its position in the shape, and train only one example of each class. In these experiments a remarkable decrease in the number of possible configurations could be noticed.

The parameter values found in the experiments where agents had to group in a discrete toroid, showed that the sensing capabilities obtained were minimal, considering the number of possible configurations. Furthermore, the method Obsenpar has proved to be more effective to define the sensing parameter values, than the definition made by hand by an expert in the problem.

It was empirically proven that the policies, both stochastic and deterministic, worked much better than a purely Random one.

To sum up, the three questions raised in the introduction have been answered empirically:

1. Can agents act in an autonomous way or do they need to be coordinated in a central way?

   Agents did not need a central coordination to achieve the goal. Moreover, they could learn not only their behavior but also, their sensing capabilities in an independent way. The only influence among agents was the change in the environment, used as a kind of communication.

2. Must all agents have the same capabilities or can different types of agents be used to solve the problem?

   It was shown that, both kinds of behaviors, homogeneous and heterogeneous, were found using the proposed methods. Furthermore, in the case of clustering problem, where a homogeneous and deterministic behavior is needed, a Stochastic Policy could be used too. This Stochastic Policy was learned using the proposed methods in this thesis.

3. Must each agent know how to solve the whole problem or may the problem be solved by the interaction of many or all of the agents in the swarm?

   The interaction of all agents in the swarm was necessary to solve the problems presented here, and the proposed methods allow learning the required knowledge.

With regard to fault tolerance, and in the case of homogeneous behaviors, the testing processes showed that agents can be added without any need for retraining. Therefore, if some agents fail in solving the problem, it is only necessary to replace them with other agents with similar characteristics. If the agents are heterogeneous and some of them fail, it is only necessary to replace those agents with agents that are member of the same class as the broken ones. These classes must be detected before starting to use the proposed methods.

Future work will be devoted to the characterization of the perceptual aliasing, specifically the detection of the hidden states that cause conflictive perceptual aliasing. As it was stated in [34, 18], a hidden state arises when the mapping between states of the world and what the agent is sensing is not one-to-one". The characterization of the hidden states will allow a premature rejection of configurations coded in the chromosome, pruning the search space considerably.

Bibliography

[1] *Behavior-based robotics*, ch. 9: Social Behavior, MIT Press, 1998.

[2] N. Agmon, S. Kraus, and G. A. Kaminka, *Multi-robot perimeter patrol in adversarial settings*, IEEE International Conference on Robotics and Automation (ICRA 2008) (2008), 2339–2345.

[3] Y. Altshuler, V. Yanovsky, I. A. Wagner, and A. M. Bruckstein, *The co-operative hunters-efficient cooperative search for smart targets using uav swarms*, Proceedings of MARS2005 workshop at ICINCO2005. INSTICC (2005), Barcelona, Spain.

[4] Alexander U. Berezhnoy, *Emergent behavior in multiagent systems*, 3rd Annual Winona Computer Science Undergraduate Research Symposium (2003), Winona.

[5] S. Camazine, J. Deneubour, N.R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau, *Self-organization in biological systems*, Princeton University Press (2003).

[6] Lonnie Chrisman, *Reinforcement learning with perceptual aliasing: The perceptual distinctions approach*, AAAI-92 Proceedings (1992).

[7] D.V.Dimarogonas and K.J.Kyriakopoulos, *Decentralized stabilization and collision avoidance of multiple air vehicles with limited sensing capabilities*, Journal of Intelligent and Robotic Systems (2007), no. 3, 411–433.

[8] Y. Elmaliach, N. Agmon, and G.A. Kaminka, *Multi-robot area patrol under frequency constraints*, IEEE International Conference on Robotics and Automation (ICRA 2007) (2007), 385–390.

[9] Y. Elor and A. M. Bruckstein, *Multi-agent graph patrolling and partitioning*, Tech. report, Technical Report CIS-2009-01, Center for Intelligent Systems, Technion - IIT, Israel, 2009.

[10] David E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, ISBN 0-201-15767-5, Addison Wesley (1989).

[11] Noam Gordon, *Fundamental problems in the theory of multi-agent robotics*, Ph. D. Thesis, Israel Institute of Technology, Technion - Computer Science Department - Israel Institute of Technology (2010).

[12] Noam Gordon, Yotam Elor, and Alfred M. Bruckstein, *Gathering multiple robotic agents with crude distance sensing capabilities*, ANTS 2008, Springer-Verlag Berlin Heidelberg **LNCS 5217** (2008), 72–83.

[13] John H. Holland, *Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control, and artificial intelligence*, Mit Press (1992).

[14] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore, *Reinforcement learning: A survey*, Journal of Artificial Intelligence Research **4** (1996), 237–285.

[15] C. R. Kube and H. Zhang, *The use of perceptual cues in multi-robot boxpushing*, Proceedings of IEEE International Conference on Robotics and Automation **3** (1996), 2085–2090.

[16] Zhengping Li, Cheng Hwee Sim, and Malcolm Yoke Hean Low, *A survey of emergent behavior and its impacts in agent-based systems*, IEEE (2006).

[17] M. J. Matarić, M. Nilsson, and K. T. Simsarin, *Cooperative multi-robot boxpushin*, Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. 'Human Robot Interaction and Cooperative Robots' **3** (1995), 556–561.

[18] Andrew Kachites McCallum, *Instance-based state identification for reinforcement learning*, 1994.

[19] ———, *Reinforcement learning with selective perception and hidden state*, Thesis doctoral from University of Rochester (1996).

[20] R. R. Murphy, *Rescue robotics for homeland security*, Communication of the ACM **47** (2004), no. 3.

[21] Liviu Panait and Karl Tuyls, *Theoretical advantages of lenient q-learners: An evolutionary game theoretic perspective*, The Sixth Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 07) (2007), 180–186.

[22] Cristina Parpaglione and Juan Miguel Santos, *Obtención de comportamientos emergentes en sistemas multiagente mediante aprendizaje por refuerzo*, Argentine Symposium on Artificial Intelligence (ASAI2009), Jornadas Argentinas de Informática (38JAIIO) (2009), Mar del Plata, Argentina.

[23] Sol Pedre, Pablo de Cristóforis, Diego Bendersky, and Juan Miguel Santos, *Reinforcement learning in vision based mobile robots using the hough transform*, Proceedings of the 4th International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2007), Springer Verlag (2007), 161–168.

[24] M. Quinn, L. Smith, G. Mayley, and P. Husbands, *Evolving formation movement for a homogeneous multi-robot system: Teamwork and role-allocation with real robots*, Cognitive Science Research Papers, University of Sussex (2002).

[25] M. Rubenstein, N. Hoff, and R. Hagpal, *Kilobot: A low cost scalable robot system for collective behaviors*, Technical Report TR-06-11, Harvard University, Cambridge, Massachusetts (2011).

[26] Stuart Russell and Peter Norvig, *Artificial intelligence: A modern approach (3rd edition)*, Prentice Hall (2009).

[27] Richard Sutton and Andrew Barto, *Reinforcement learning: An introduction*, MIT Press, Cambridge, Massachusetts, 1998.

[28] Ichiro Suzuki and Masafumi Yamashita, *Distributed anonymous mobile robots: Formation of geometric patterns*, Siam Journal on Computing **28(4)** (1999), 1347–1363.

[29] P. Svenson, C. Martenson, H. Sidenbladh, and M. Malm, *Swarm intelligence for logistics: background*, Swedish Defence Research Agency (2004).

[30] A. J. Villar and J. M. Santos, *Q-function kernel smoother: a new approach for opened issues in huge state*, Anales del X Argentine Symposium on Artificial Intelligence, Mar del Plata, Argentina (2009).

[31] C. J. C. H. Watkins, *Learning from delay rewards*, Ph.D. thesis, Cambridge University, Cambridge, England (1989).

[32] P. R. Wavish, *Exploiting emergent behavior in multi-agent systems*, Decentralized A.I. 3, Proceedings of the 3rd European Workshop on Modeling an Autonomous Agent in a Multi-Agent World, Kaiserlautern, Elsevier Science Publishers B.V. (North Holland) (1991).

[33] A. Weitzenfeld, A. Vallesa, and H. Flores, *A biologically-inspired wolf pack multiple robot hunting model*, IEEE 3rd Latin American Robotics Symposium (LARS '06) (2006), 120–127.

[34] Steven D. Whitehead, *Reinforcement learning for the adaptive control of perception and action*, Thesis doctoral from University of Rochester (1992).

[35] Steven D. Whitehead, Richard S. Sutton, and Dana H. Ballard, *Learning to perceive and act by trial and error*, Machine Learning, 7(1):45-83 (1991).

[36] Huaxing Xu, Haibing Guan, Alei Liang, and Xinan Yan, *A multi-robot pattern formation algorithm based on distributed swarm intelligence*, Proceedings of the 2010 Second International Conference on Computer Engineering and Applications, ICCEA **1** (2010), 71–75.

[37] H. Yamaguchi, *A distributed motion coordination strategy for multiple nonholonomic mobile robots in cooperative hunting operations*, Robotics and Autonomous Systems **43** (2003), no. 4, 257–282.