Use of biologically inspired algorithms to achieve cooperation between agents

Dávalos Segura, S.

Award date:
2016

Use of Biologically Inspired Algorithms to achieve cooperation between agents

By

STEPHANIE DÁVALOS SEGURA

dr. ir. Cuijpers, R. H.
dr. Guerreiro Tomé Antunes, D. J.

MASTER OF SCIENCES
Human-Technology Interaction

Department of Industrial Engineering & Innovation Sciences
EINDHOVEN UNIVERSITY OF TECHNOLOGY

OCTOBER 2016

Abstract

Drones industry has been growing and will continue to grow in the next few years; this implies an increase in accessibility, allowing development of multiple applications in industries. Nevertheless this accessibility will also create opportunities for criminal purposes, mainly in exposed airspace where there is no longer possible to have fences, video cameras or security guards to protect the space. Therefore a technology able to protect restricted areas is needed.

Until now different ways to stop drones to enter a restricted area has been considered, most of them requiring a human operator. Nevertheless, the ability to cooperate and learn has been found extremely important and the opportunity to use autonomous multi-agent systems is expected to be suitable to guarantee a good performance on this task. In order to test this an environment that simulate an intruder trying to enter some specific area with two defenders to protect it is created on a computer simulator.

The objective of the present study is to find a policy using information from human behavior to obtain a biologically inspired algorithm (BIA) able to achieve cooperation between agents and react in real time. For this purpose an evaluation of the reaction time of BIA compared with the Minmax algorithm was made, finding that BIA reacts 2000 times faster than Minmax.

Furthermore, BIA was also compared against the performance of human players showing evidence that BIA can perform comparable to an average team. Therefore, it is showed how this method can be applied on situations where reactions in real time are needed with the confidence that it will react as good as most of the people.

Table of Contents

List of Tables

Table **Page**

List of Figures

CHAPTER 1

INTRODUCTION

Unmanned aerial vehicles (UAV's), better known as drones, are one of the fastest-growing industries in the world. According to Teal Group Corp. () (Group,) and Radiant Insights RadiantInsights (Insights,) the value of drone market in 2014 was USD $609 million and is projected to reach USD $4.8 billions by 2021. Moreover the market for commercial/civilian drones is predicted to grow at a compound annual growth rate of 19% between 2015 and 2020.

1.1 Motivation

Nowadays the global commercial drone market is developing applications in industries such as: agriculture, energy, utilities, mining, construction, real estate, news media and film production. However, the increase in accessibility, cost, efficiency and technology of drones also means more opportunities to misuse them for criminal purposes. This mainly happens in an exposed airspace, where there is not longer possible to have fences, video cameras or security guards to protect the space.

Some of the problems that have been encountered in the past 3 years are:

- A drone flying over a stadium on the soccer game between Serbia and Albania 2014 (Journal,). In the 40th minute of this match, a drone was seen hovering above the pitch carrying a flag with the 'Greater Albania' insignia. When one of the Serbian defender's pull down the flag a dozen fans ran into the field to confront the Albanian players and the police needed to intervene. This shows that a drone not necessarily need to drop a harmful substance or shut a gun to disturb an event.

- Drone inside the airport (Times,). On April 4, 2016 a drone was seen flying within meters of three landing planes at Schiphol airport. All the planes landed safely. However the airport decided to close the landing strip after the incident, which means that all the other flights were diverted to other airport. This can be seen as a threat for the plane in case that the drone crashed with the turbine but also as a loss of money for the airport.

- Drone "attack" on German Chancellor Angela Merkel (One,). On September 2014, during a campaign, a Parrot AR drone crashed in front of the German chancellor Angela Merkel. No one was harmed. However from this incident one can think of a similar situation with drones carrying weapons.

- Drone caught carrying drugs near the border (News,). On January 20, 2015 a drone carrying methamphetamine crashed near to the US border. The police said that now the smugglers try to do business by air apart from the tunnels that they are already using.

These are a few examples that summarize the most disturbing incidents. However, there have been more small incidents with drones where people have been injured, followed and streamed. It is common to see drones trespassing restricted areas. It is often debatable whether the operator of a drone is ill-informed rather than ill-intentioned. However being one or the other, it seems that build a technology to protect certain areas is needed. This can also be seen in the increment of new startups, universities and defense companies that are working on detecting and stopping unwanted drones.

One can observe several solutions to counteract or mitigate these threats. The Japanese police had began an "anti-drone squad", where one police officer operates a drone that flies above its target carrying a net. The intruder gets stuck in the net being unable to fly and making it possible to carry it to a desire place. Dutch police is training hawks to grab drones. In this case the hawk intercept and grab drones in midair. The British firm Openworks Engineering has developed a projectile, which can fire a net attached to a parachute to capture the drone when it is flying. Moreover a professor at Michigan Tech is developing a drone that can shoot a net to capture other drones in the air. Nevertheless, all these ideas have something in common there is always a need of an operator. (Figure 1.1)

Hence, the motivation of this work is to find an autonomous solution to avoid drones threats, improving the strategy used by Coluccini (Coluccini,). Where the goal is to secure an area from an external intruder using Artificial Intelligence. Coluccini defined the protected area as a circle, where the strategy used is based on a worst-case scenario minimax algorithm for a zero-sum game. The movements of the defenders are restricted to clockwise or counterclockwise but the movements of the attacker are considered to be unknown. All the agents are consider to have full information of the environment. It is shown that the defenders achieve some level of intelligence

FIGURE 1.1. (a) Japanese police. Drone catching another drone using a net.(b) Dutch police. A hawk catching a Drone. (c) Openworks Engineering. A gun to shoot a net (d) Michigan Tech. Combination of a net gun and a drone

and are able to secure the area. Nevertheless, Coluccini encouraged the exploration of different algorithms for smarter and faster search strategies.

1.2 Objective

As previously mention, different incidents related with drones have occurred in recent years; therefore, it is necessary to provide technological solutions to protect certain areas. Such a solu-

tion may include one or several UAV agents. The ability to cooperate and learn in multi-agent autonomous systems is however expected to be critically important to guarantee a suitable performance on problem-solving. Following the definition of Smith (E,), cooperative learning is a process which depends on socially structured exchange of information between the agents involved while each agent is responsible for its own learning. Moreover according to Panait and Luke (Panait,), faster learning and higher efficiency can be achieved for multi-agent cooperative learning compared to individual learning.

Different ways of implementing cooperation have been studied over the years; some of this rely on sharing sensations (communicate instantaneous information), episodes (sequences of sensations) or learning policies. Ming Tan (Tang,), showed that sharing policies by cooperative agents can lead to a faster learning compared to only share sensation or episodes. Earl and Andrea (Rafaello,) showed that by decomposing a problem, assigning individual tasks to each agent can lead to a near-optimal cooperative policy, allowing to solve the individual task on a model predictive framework. Riccardo Coluccini (Coluccini,) considered a defense task for a multi-agent autonomous system and formulated it as a zero-sum game. A Minmax algorithm was proposed to determine the outcome for the agents. Due to computation restrictions a heuristic at given scenarios was used. However, there are two main challenges that the author describe as requiring further research. The first one is related to the definition of the heuristic, which in some cases may not yield acceptable results and the second one is the required computational time for decision making.

The present research will be focused on learning a biologically inspired policy for two agents. The goal of the agents is to protect a specific area from an intruder. To achieve this, a supervised learning algorithm will be implemented, using information obtained from players in a computer game which emulates the position of two agents and the area to protect. The computer game will provide visual information to the participants with respect to the positions of both agents, the intruder and the area that needs to be protected. Each experiment will require three persons, two manipulating the agents and one manipulating the intruder, and the experiment will be composed by a predefined number of trials. The combined information related to the movement of the agents and the intruder will be stored in a csv file for further analysis.

1.3 Outline

This thesis is organized as follows:

Chapter 2: Machine Learning
This chapter is used to introduce the reader topics related to machine learning that are important

for this project. It starts by defining the differences between learning methods encountered in the field of machine learning, then it focuses on the description of one specific approach called supervised learning, followed by the definition of different algorithms that are relevant to the task of classification.

Chapter 3: Research Methodology

The objective of this chapter is to discuss the methodology used to obtain information from humans cooperating in order to later create a biologically inspired policy. To do so, a real world scenario is represented in a computer simulator where the description of the scenario is given, followed by the problem formulation, where a clear math representation of the variables is made, allowing the development of the simulator in Python. Finally the experimental design is described.

Chapter 4: Algorithms

This chapter will focus on detailing the two algorithms that we consider for the agents to take decisions: the minimax algorithm that was studied by Riccardo Coluccini (Coluccini,) and a new approach using supervised learning to predict the movement of the agents based on the position of the intruder. In both cases perfect information of the environment is considered.

Chapter 5: Results, Discussion & Conclusion

On this chapter the information related to the simulation will be analyzed. The results of comparing Minmax algorithm and the new biologically inspired algorithm will be presented. Besides the comparison between the biologically inspired algorithm and human players. Finally, an overview on how this can be applied in the real world, accompanied by some possible directions that future research can consider will be given.

CHAPTER 2

MACHINE LEARNING

Machine Learning is a broad notion pertaining to building computational artifacts that learn over time based on experience. Many researches will also refer to machine learning as applied computational statistics. Back in 1959 Arthur Samuel (developer of the world's first self-learning program (Samuel,)) described machine learning as "The field of study that gives computers the ability to learn without being explicitly programmed."

This chapter will start with a short description of the different types of learning that are currently used and the differences between them in Section 2.1, followed by a more detailed introduction of how supervised learning algorithms work in Section 2.2. Finally a description of supervised learning algorithms used for classification will be introduced in Section 2.3.

2.1 Types of Learning

The concept of learning is defined as the process of predicting an output from a finite set of data, and there are three main learning options that Machine Learning can follow to model a problem based on its interaction with the environment (Murphy,):

- **Supervised Learning** can be defined as an induction of an approximate function, where a model is created based on the data mapped from an input space X to an output space Y. In this case it is necessary to have a data set D_N, called the training data set, consisting of N input-output pairs $\{x_i, y_i\}_{i=1}^{N}$. Each training input x_i is a D-dimensional vector of numbers representing the features or attributes of the sample. The output variable belongs to a finite set $y_i \in \{1, ..., C\}$ where in principle can be anything but most methods assume it categorical or nominal. This type of learning is subdivided in two categories:

  - **Regression:** The goal of this method is to learn a predictor f using a training set, where a function that maps inputs to desired outputs is generated. In this case y_i is real-valued.
  - **Classification:** The goal of this method is to assign a new observation to a predefined set of categories. In this case y_i is categorical.

- **Unsupervised Learning** can be defined as the ability to derive a structure by looking to the relations to the data, when only inputs are given, $D = \{x_i\}_{i=1}^{N}$. The objective is to model hidden structures in the data in order to know more or understand better the data. Mostly done in two different ways: Clustering (which use inherent grouping) or Association (which find rules that describe different portions of the data).

- **Reinforcement Learning** can be defined as the act of adapting an improving behavior of an agent by interacting with the environment via trial and error. This kind of algorithms allow agents to automatically determine the optimal behavior within an specific context, in order to maximize its performance via a reward feedback. (Sutton & Barto,)

2.2 How supervised learning algorithms work

Given a training data set D_N, consisting of N input-output pairs $\{x_i, y_i\}_{i=1}^{N}$ infer a function $f : X \rightarrow Y$, where X represents the input space and Y the output space and where the function f represents an element of the space of possible functions F. The aim of this kind of algorithms is to minimize the expected number of wrong prediction when the mapping function is being learned ($f : X \rightarrow Y$) considering the mapping function as non deterministic (there is not one Y for each X) and the training data set incomplete (not covering the cover the complete space X and Y). Taking into account these factors it is possible to represent f using a scoring function $g : X$ x Y $\rightarrow \mathbb{R}$ where g returns the y value that gives the highest score:

$$f(x) = \underset{y}{\operatorname{argmax}}\, g(x, y) \tag{2.1}$$

where x and y are consider as any valid input and output sequences, $f(x)$ considered the predicted sequence with the highest value and G being the space of scoring functions .

Despite the fact that G and F can be any space functions, most learning algorithms are probabilistic models where f takes the form of a conditional probability model $f(x) = P(y|x)$ or g takes the forms of a joint probability model $g(x, y) = P(x, y)$. In order to choose between f or g two approaches are available: empirical risk minimization which seeks the function that best fits the training data set and structural risk minimization which provides a formal mechanism for choosing an optimal model complexity for a finite sample using a penalty function that provides a trade off between hypotheses space complexity (the VC-dimension of approximating functions)

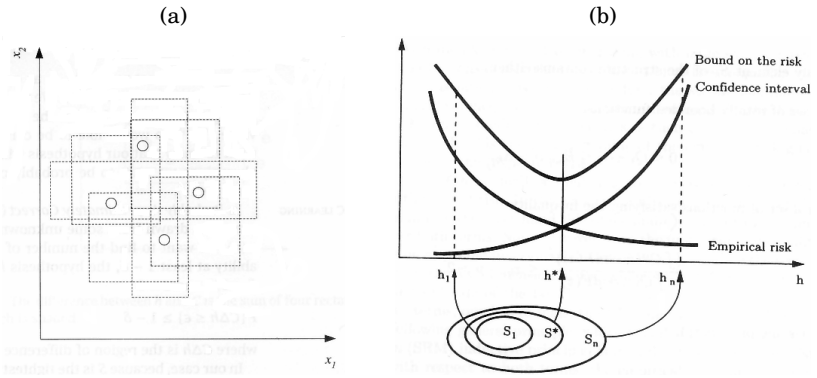

FIGURE 2.1. (a) Capacity of hypothesis space. Measure of complexity, expressive power, richness or flexibility of a set of functions by assessing how wiggly its members can be. (b) Structural Risk Minimization. Inductive principle used to minimize the risk functional with respect to the empirical risk and confidence intervals. Figure taken from (Org,)

and the quality of fitting the training data set (empirical error) (Vapnik,), Figure 2.1, both consider the training data set as independent and identically distributed.

To measure how well a function fits the training data set a loss function is defined $L : X$ x Y x $\mathbb{R} \rightarrow [0, \infty)$ where the term "loss" indicates our interest in small values of L. Considering a training example (x_i, y_i), the loss of predicting the value $\hat{y}$ is given by $L(y_i, \hat{y}) = -logP(y|x)$.

The risk $R(f)$ of the function f is defined as the expected loss of f which can be estimated by the training data set as $R_{emp}(f) = \frac{1}{N} \sum_i L(y_i, f(x_i))$.

2.3 Supervised Learning algorithms

Considering that we need to predict the movement of two agents that are attempting to protect a specific area and the research made by Steinberg et. al. (Xindong Wu & Steinberg,), Caruana & Mizil (Caruana & Niculescu-Mizil,) and Stobiecki and Sniezynski (Stobiecki & Sniezynski,), some supervised learning algorithms relevant to the task were selected for the description of this section.

2.3.1 Decision Trees

Decision tree learning is a method used for inductive inference, which goal is to approximate discrete-valued target functions. This learning method classify instances by sorting them down using a tree form, starting from the root and descending into a leaf node, providing classification of the instances on each branch (Harrington,).

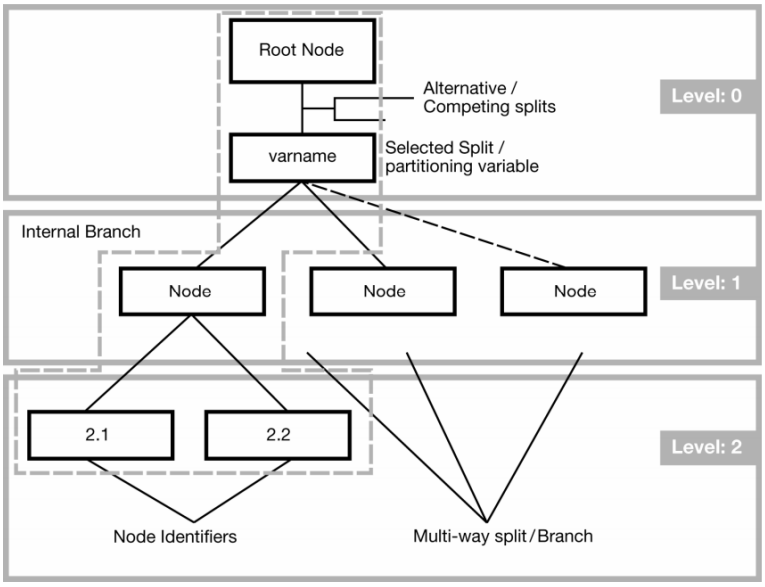

FIGURE 2.2. Illustration of a decision tree nomenclature. Figure taken from (Jensen,)

A nested hierarchy of branches is called a decision tree and each branch is a node, the terminal nodes of the tree are called leaves. For each terminal node there is a unique path from the root to the leaf. All nodes have mutually exclusive assignment rules which allow to predict new node values based on new or unseen data, where the decision rule yields the predicted value, Figure 2.2.

Decision trees have the ability to analyze multiple variables at the same time facilitating the description of multiple influences at once. The advantage of this methods relies in their ease of use and interpretation, relative power and robustness to deal with a variety of data. There are presented as a combined set of multiple influences (one cause-one effect relationships) in a recursive form, it can be seen as a set of if-else statements, which attempt to find a strong relationship between the input-output values of a group of observations (training data set) (Jensen,).

All trees start from a root node from which testing a previously specified attribute decide the following branch until the leaf node is reached. The main difference between trees relies on the way in which each branch is chosen, the splitting criteria, where most of the decisions are based on a single attribute.

Considering a training data set $S = s_1, s_2, ..., s_n$ where each sample s_i is a p-dimensional vector $(x_{1,i}, x_{2,i}, ..., x_{p,i})$ where x_j represents the features of the sample the two most used algorithms used for Decision Trees will be described:

- **C4.5** uses the concept of information entropy for its impurity function, referring to the

split made on each node, where the objective is to obtain a node composed by data points belonging to the same class (a pure node).

The impurity function for each ordered variable x_j is defined by the question $\{Is \quad x_j \leq c?\}$ where c is real-valued and the information gain is defined as reduction in entropy.

$$Entropy(i) = -\sum x(j,i) log_2 x(j,i) \tag{2.2}$$

where, $x(j,i)$ denote the fraction of records that belong to class j at a given node i. Hence, the entropy of an event $c(x)$ will become smaller as we learn about the value of some feature of x. The advantage of this algorithm is the fact that the tree can be pruned, decreasing the error rates on unseeing data (Singh,).

- **CART** uses the concept of a Gini Impurity function, which evaluates the divergence between the probability distributions of the target attribute values.

$$GiniIndex = 1 - \sum |\, x(j,i)\,|^2 \tag{2.3}$$

One of the advantages of CART is its ability to generate regression trees, allowing the leaf nodes to make predictions. This algorithm is able to identify by itself the most significant and non-significant variables to use it or eliminate it.

2.3.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a classification technique able to map nonlinear separable vectors into a high (or infinite) dimensional space and separate them using linear classifiers called hyperplanes; allowing it to make good prediction of data points outside the training data set. The predictions are best when the hyperplane has the largest distance to the nearest data point of any class, this is called functional margin. An hyperplane is represented by the equation (Kulkarni,):

$$aX + bY = C \tag{2.4}$$

In order to understand how a SVM works a training data set composed by vectors $\{x_i, y_i\}$, $i = 1,...,l$, $y_i \in \{-1,1\}$ in some space $X \in \mathbb{R}^d$ is defined. And it is assume that there is an hyperplane able to separate the two classes (positive and negative). The vectors that are closer to the hyperplane are called support vectors which have direct bearing on the optimum location of the decision surface. To define the hyperplane the training data in space X is projected to a higher dimensional feature space F ($\phi : X \mapsto F$) where it is no need to know explicitly ϕ only needed to know $\langle \phi(x_i), \phi(x_j) \rangle$ the dot product of the transformed points. Consequently it is possible to work with $K : XxX \mapsto \mathbb{R}$, where K takes two points as input and return a real value. The existence of ϕ can only be guaranteed when K is a convex function (positive definite), this condition is

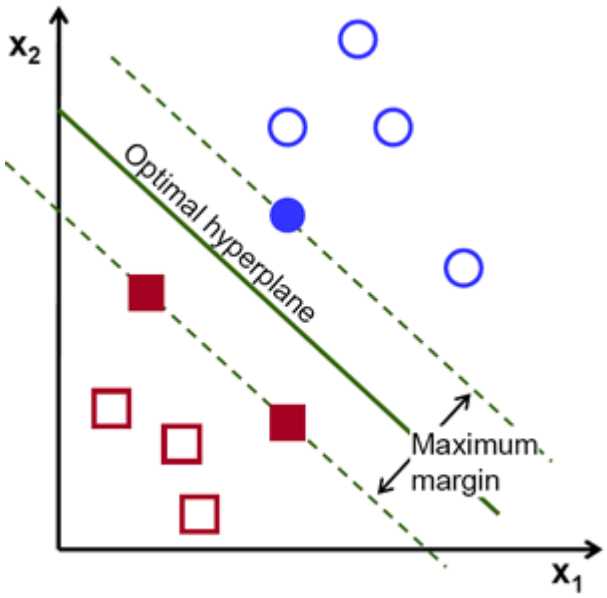

FIGURE 2.3. Optimal hyperplane separating features. Figure taken from (OpenCV,)

called a Mercer's kernel condition and guarantees the existence of an underlying map ϕ such that $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ (Cortes,). For this the set of classifiers is considered in the form:

$$f(x) = \left(\sum_{i=1}^{n} \alpha_i K(x_i, x) \right) \tag{2.5}$$

and assuming that Mercer's condition is satisfied it is possible to rewrite f as:

$$f(x) = w \cdot \phi(x), \qquad where \quad w = \sum_{i=1}^{n} \alpha_i \phi(x_i) \tag{2.6}$$

where SVM obtain the maximal margin hyperplane in F making use of the $\alpha_i s$. Figure 2.3 shows an optimal hyperplane with maximum margins with respect to the support vectors.

2.3.3 K-Nearest Neighbor (k-NN)

K-Nearest Neighbor is defined as a type of instance-based learning algorithm when is used for classification. It stores the training data and classifies new instances based on the highest number of neighbors shared, the number of neighbors (k) to be analyzed around the new instance is a pre-specified variable and the goal of this algorithm is to measure the differences or similarities between two or more instances.

In order to make a prediction with k-NN a metric for measure the distance between instances

should be defined, the most common ones are:

$$D(x,y) = \begin{cases} \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} & Euclidean \\ \sum_{i=1}^{k} \mid x_i - y_i \mid & Manhattan \\ \left(\sum_{i=1}^{k}(\mid x_i - y_i \mid)^q\right)^{1/q} & Minkowski \end{cases} \tag{2.7}$$

the standard distance function is the Euclidean distance $d(x,y)$.

A training data set composed by $\{(x_1,y_1),...,(x_n,y_n)\}$, where $X \in \mathbb{R}$, x_i represents features and y_i is consider to be the index of a category to which the ith instance belongs on the set $\{1,2,...,M\}$. Considering a new instance is given (x,y) where only information related to x is available and it is desired to estimate y using the information contained on the training data set and only one neighbor is considered.

The new instance is defined to belong to a specific set if:

$$min \quad d(x_i,x) = d(x_p,x) \qquad i = 1,2,...,n \tag{2.8}$$

where

$$x_p \in \{x_1,x_2,...,x_n\} \tag{2.9}$$

Therefore the new instance will be classifies as being part of the category y_p considering their nearest neighbors x_p (Cover,).

For more than one neighbor ($k = n$) the algorithm classifies according to the maximum number of neighbors shared by one class. In Figure 2.4 the decision of the class is made based on 3-nearest neighbors, in this case it is possible to see that there are 2-nearest points belonging to the category of B and 1-nearest point belonging to the category of A, from this information the algorithm will decide to classify the new point as belonging to the category of B.

2.3.4 Naive Bayes

Naive Bayes is a classification technique based on Bayes' Theorem. The main characteristic of this classifier is that it consider all the features to be independent for the given value of the class variable, this is called conditional independence.

In order to understand better how this classification algorithm works it will be considered a training data set composed by $\{(x_1,y_1),...,(x_n,y_n)\}$, where x_i is a vector that represents features and y_i indicates the category to which x belongs. In this example we will assume only two classes $y \in \{-1,1\}$. According to Bayes' Theorem the probability of an example $E = \{x_{n1},x_{n2},...,x_{nn}\}$ to

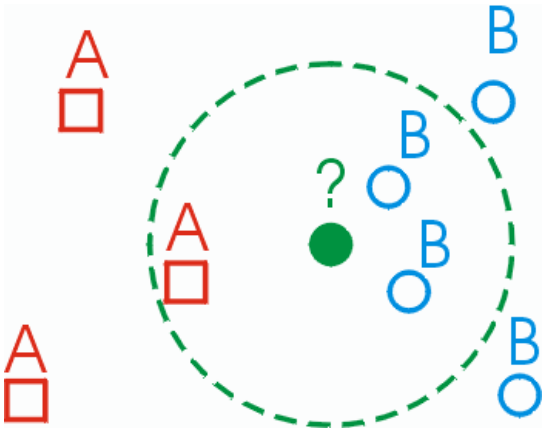

FIGURE 2.4. Decision based on 3-nearest neighbors. Figure taken from (Perrier,)

belong to class y is given by (Zhang,)

$$p(y|E) = \frac{p(E|y)p(y)}{p(E)} \tag{2.10}$$

where $p(y|E)$ is a conditional probability, probability to observe y given that E is true. E is classified as belong to class $y = 1$ if and only if

$$f_b(x) = \frac{p(y=1|E)}{p(y=-1|E)} \geq 1 \tag{2.11}$$

where $f_b(E)$ is called a Bayesian classifier. Now, assuming that all the features of the example are independent to each other given the value of the class it is possible to represent it as

$$p(E|y) = p(x_1, x_2, ..., x_n|y) = \prod_{i=1}^{n} p(x_i|y) \tag{2.12}$$

giving as a result a new classifier called Naive Bayes classifier ($f_{nb}(E)$)

$$f_{nb}(E) = \frac{p(y=1)}{p(y=-1)} \prod_{i=1}^{n} \frac{p(x_i|y=1)}{p(x_i|y=-1)} \tag{2.13}$$

One of the advantages of the Naive Bayes classifier is that it do not require large amount of training data to provide a good classification of new data; because the interaction between features are ignored when the model is built. Figure 2.5 shows a representation of independence between variables given a class.

2.3.5 Neuronal Networks

Neural Network, mostly found in the literature as artificial neural networks (ANNs), can be thought as a fitting function (or approximation function). In order to understand the functioning of ANN, the way in which a *perceptron* (artificial neuron) works should be understood.

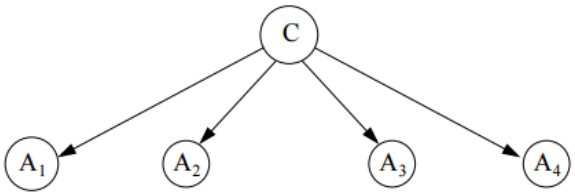

FIGURE 2.5. Representation of independent variable for Naive Bayes classifier. Figure taken from (Zhang,)

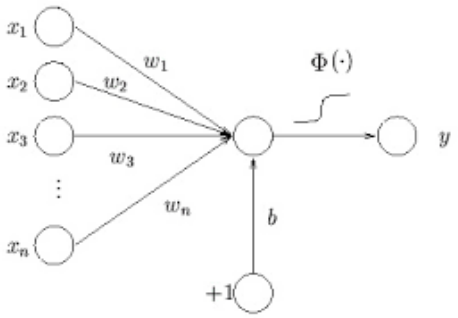

FIGURE 2.6. Representation of a single perceptron. Figure taken from (Honkela,)

A perceptron takes several inputs $x_1, x_2, ..., x_n$, weight them according to pre-specified values, which indicate the importance of the input $w_1, w_2, ..., w_n$ and produce a binary output $(0, 1)$ that is determined by whether the sum $\sum_j w_j x_j$ is greater or smaller than some threshold value.

$$output = \begin{cases} 0 & if \quad \sum_j w_j x_j \leq \quad threshold \\ 1 & if \quad \sum_j w_j x_j > \quad threshold \end{cases} \tag{2.14}$$

the equation of a perceptron can be defined as

$$y = \phi\left(\sum_{j=1}^{n} w_j x_j + b\right) \tag{2.15}$$

where y represents the output, ϕ the activation function, n the number of connections to the perceptron, w_j the weight associated to each connection, x_j the value of each input and b the threshold (Honkela,). It is important to notice that varying the weights and the threshold different models can be obtained, however the problem here is that any small change on the weight or bias of a single perceptron can lead a drastic change in the output.

In order to deal with this, a different neuron is used, called sigmoid neuron. It works in a similar way of perceptrons with the advantage that it can overcome small changes. The main difference between a perceptron and sigmoid neurons is the output

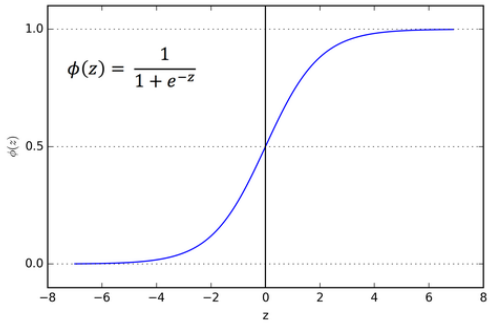

FIGURE 2.7. Representation of a sigmoid function. Figure taken from (Raschka,)

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+exp^{-\sum_j w_j x_j - b}} \tag{2.16}$$

where $z \equiv w \cdot x + b$ is consider a large positive number, hence $e^{-z} \approx 0$ and so $\sigma(z) \approx 1$ or in other words when $e^{-z} \to \infty$, $\sigma(z) \approx 0$. This function is consider as a smooth perceptron and can be seen in Figure 2.7.

2.3.6 Ensemble Methods

The main idea of Ensemble learning is to use multiple learners and combine their predictions. First, a set of classifiers is constructed $f_1, f_2, f_3, ..., f_M$ and when a new data point needs to be classified the algorithm "average" their (weighted) votes of the predictions and reduce the risk of choosing the wrong classifier (Sewell,). Consider a binary output ($1 or -1$) and a training test $\hat{x}$, then compute $\hat{y}_1 = f_1(\hat{x}), ..., \hat{y}_M = f_M(\hat{x})$. If there are more +1 in the list $(y_1, ..., y_M)$ then you predict +1 otherwise you predict -1, Figure 2.8.

This learning method is good when the size of the hypothesis space is a lot bigger than the amount of training data available. The main problem is that without enough data, the learning algorithm can find many different hypothesis with the same accuracy, leading to a misclassification. One of the advantages of ensemble algorithms is the fact that an average from the (weighted) output of all the accurate classifiers will be consider, which reduce the risk of choosing a wrong classifier (Figure 2.9(a)). Another reason is the fact that the algorithm starts from many different points, which helps to avoid getting stuck in local optima. The third advantage of ensemble methods is the possibility of expanding the space of representable functions, where the function f cannot be represented by any of the hypothesis given by the training data set. (Dietterich,)

The most used ensemble methods are:

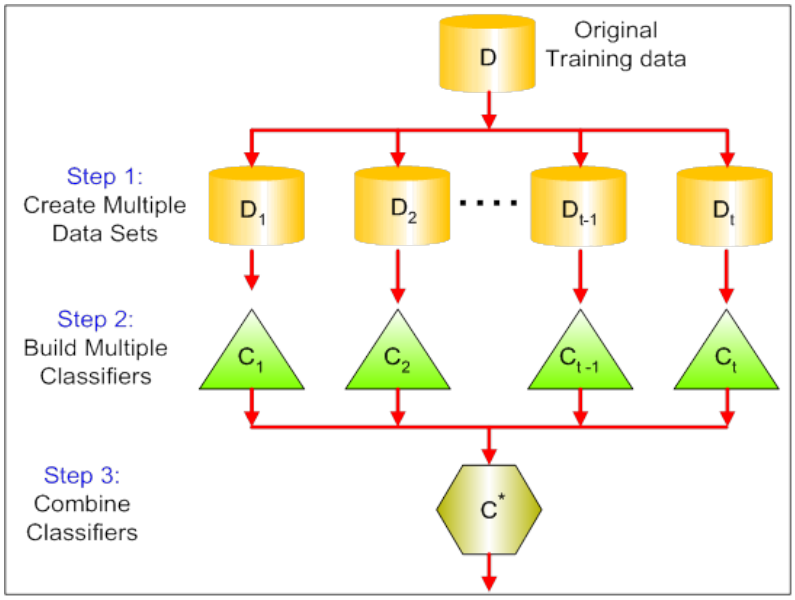

FIGURE 2.8. General idea of Ensemble Methods. Figure taken from (Vidhya,)

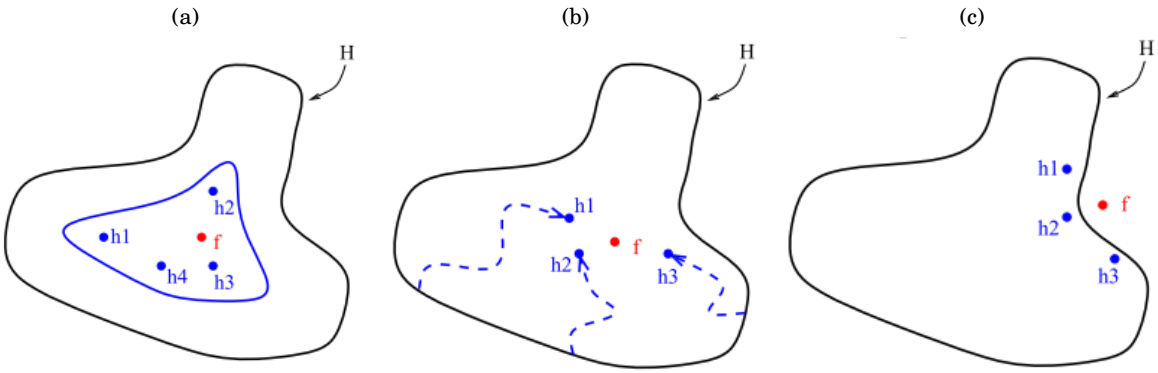

FIGURE 2.9. (a) Average of outputs to avoid misclassification. (b) Start from different points. (c) Account for hypothesis outside the space of representable functions. Figure taken from (Dietterich,)

- **Bagging**, a name derived from bootstrap aggregation, where a classifier is built on each bootstrap sample (sample with replacement), each sample has a probability $1-(1-1/n)^n$ of being selected

- **Boosting**, is an iterative procedure to adaptively change distributions of training data by focusing more on previous misclassified outputs. If an output is classified wrongly, their weight will increased, otherwise their weights will decreased.

  - **AdaBoost** is an algorithm for constructing a "strong" classifier as linear combination

of simple "weak" classifiers $h_t(x)$

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{2.17}$$

where h_t is consider as the week hypothesis, α_t is the assigned weight and the function is computed as a weight majority vote of the weak hypothesis. (Schapire,)

– **Random Forest** is a combination of tree predictors $h(x, \Theta_k)$ where each tree depends on the values of a random vector Θ_k sampled independently with the same distribution for all trees in the forest and each tree gives a unit vote for the most popular class at input x. (Breiman,)

CHAPTER 3

RESEARCH METHODOLOGY

The task of preventing drones from entering into a restricted area, until now is done by different means. However most of them require a human operator and it is mostly achieved by only one agent (drone/gun); this restrict the protection task to the periphery around the human or around the visible area. This makes it difficult to protect big areas as stadiums, prisons or important monuments. In order to achieve protection for larger places it is necessary to control more than one agent, which requires cooperation among the agents to find an optimal solution.

In order to achieve this goal it was necessary to create and environment that simulate a situation that can be encounter in real life. For this, it was decided to design a computer simulator, where two defending agents, one intruder and a protected area can be visualized.

The objective of this chapter is to discuss the methodology used to obtain information from humans cooperating. This information will be used later to learn a policy for the two defending agents to protect a given area. First, a description of the scenario will be given in Section 3.1, followed by the problem formulation in Section 3.2 and the description of the experimental design on Section 3.3.

3.1 Description of the scenario

A multi player game on a 2-D grid is considered, in which two agents try to protect a specific area from another agent referred to as intruder. The agents objective is to protect a circular perimeter and for this purpose some assumptions are made: both agents are placed on the periphery and

their movements are restricted to be -move clockwise along the periphery, move counterclockwise or stand still-. The total number of actions that the agents can produce correspond to a permutation with repetition that produces 9 different possibilities. The intruder has the ability to move on 9 different directions, the four cardinal points and the four intercardinal directions and stand still. The only restriction for the intruder is that it can not move out from the game area.

Once the game starts the two agents are positioned randomly over the periphery of the area to be protected, after 2 seconds the intruder appears in a random position in the world. The game is considered to be completed, when the distance of at least one of the agents to the intruder is smaller than a certain region around the agents (ϵ) or when the intruder reach the protected area.

3.2 Problem Formulation

The world W is defined in an Euclidean space in which $W = N\text{x}M \in \mathbb{R}^2$ and the agents are defined in the world as agent one (A^1), agent two (A^2) and intruder (A^3). A goal region is defined as the area that needs to be protected $G \subset W$ and it is assumed to be stationary at all times. The space of all possible configurations for one agent in the world W is defined as the interaction space I. The position of the intruder (A^3) in the world will be denoted by $q \in I$ where $q^3 = (x, y)$ and for the defenders (A^1 & A^2) by $q^i = \theta_i$ where $i = 1, 2$ for $W = \mathbb{R}^2$. The complete state space is defined as the Cartesian product of all the interactions spaces of the agents A^i for $i = 1, ..., 3$

$$X = \theta_1 * \theta_2 * (x, y) \tag{3.1}$$

where the states are defined as $x = (q^1, q^2, q^3)$.

As the goal is stationary, the agents will be restricted to move only over the periphery and it is considered that the game is completed when the intruder reach the goal space. The goal region is defined in the completed state space as

$$X_{goal} = \{q^i \in X | A^i(q) \cap G \neq 0\} \tag{3.2}$$

considering that within the periphery no agent can be positioned, the leftover space $X - X_{goal}$ contains the remaining configurations and it is called the free state space X_{free}.

To be able to obtain the information from the game, the world is represented by a finite number of states x. Any action of an agent, u, when applied from the current state x to a new state x' is represented by the transition function

$$x' = f(x, u) \tag{3.3}$$

with $U(x)$ acting as the action space for each state x. Defining the set of all possible actions over all the states (LaValle,)

$$U = \bigcup_{x \in X} U(x) \tag{3.4}$$

The movements of every agent are represented as a point in the grid with coordinates of the form (i, j) and the transition from one point to another is represented as $f(x,u) = x + u$. A state of the intruder has a maximum of 9 possible options, reachable with the set of actions $U^3 = \{(x,0),(x,y),(0,y),(-x,y),(-x,0),(-x,-y),(0,-y),(x,-y),(0,0)\}$. The state of any of the agents will be different due to limited movement over the periphery, so it will be considered in terms of radians where the movement counterclockwise is defined as $\theta = \theta - \frac{\pi}{36}$, clockwise is $\theta = \theta + \frac{\pi}{36}$ and stand still is $\theta = \theta$. This movements will be linked to the Cartesian coordinates using the following equations

$$\begin{aligned} x' &= x + r cos\theta \\ y' &= y + r sin\theta \end{aligned} \tag{3.5}$$

where r represent the radius of the circumference for the protected area. The reachable set of actions for the agents is defined as $U^{1,2} = \{(x,0),(0,y),(0,0)\}$. And the set of possible actions for the three agents defined as $U = \{\theta_1, \theta_2, (x^3, y^3)\}$ or in Cartesian coordinates as $U = \{(x^1, y^1),(x^2, y^2),(x^3, y^3)\}$. It is important to note that: the world, the position of the agents and the position of the protected area in the world can be extended to $\mathbb{R}^3$.

3.3 Experimental Design

A computer game was developed using Python together with the library Pygame, an open source library focused on game development. The simulation was developed considering a world W = 600x400, where tree players interact, two controlling one agent each and a third one controlling the intruder. The agents are programmed to be controlled using a USB control, one per agent and the intruder is programmed to be controlled using a keyboard, Figure 3.1.

As it was describe before, the objective of the simulation is to create an environment in which two agents (drones) cooperate to protect a specific area. An initial screen was created in order to give the necessary instructions to the participants, Figure 3.2(a). The participants get clear instructions on how to use the controls. In particular is not allowed to speak to each other during the simulation or communicate in any way and the only information that they will have is feedback given by the screen; also they are told that an intruder will appear in a random place in the screen after two seconds. Finally when they feel ready, they can press the button start.

After one of the players press the button start a second screen appears, see Figure 3.2(b). In this screen two circles of different color (magenta & cyan) represent the agents, they are

(a)

(b)

FIGURE 3.1. (a) USB Controls. Used to control the agents (b) Keyboard. Arrow keys used to control the intruder

allowed to move clockwise or counterclockwise over a circumference that represent the area to be protected. After two seconds an intruder appears in the simulation, see Figure 3.2(c). Which is represented by an orange circle. At this point the participants can move agents and intruder in such a way that they can achieve their specific goals.

The game is considered to be over if the agents move in such a way that they prevent the intruder to enter the protected area by positioning themselves a distance ϵ from the intruder, Figure 3.3(a), or if the intruder is able to enter the protect area, Figure 3.3(b).

The simulation is programmed in such a way that 100 trial are done per participants and has an estimated duration of 15 minutes. The amount of trials per simulation was chosen considering that each trial will give an approximate of 170 points of information. During each trial a .csv file is saved containing the (x, y) position of the intruder and the position of the agents in angles. This information is the used later to learn the policy for the agents.

Furthermore, in order to evaluated how good the biologically inspired policy works another experiment is done. Where 6 different participants were asked to play 2 times 20 trials against

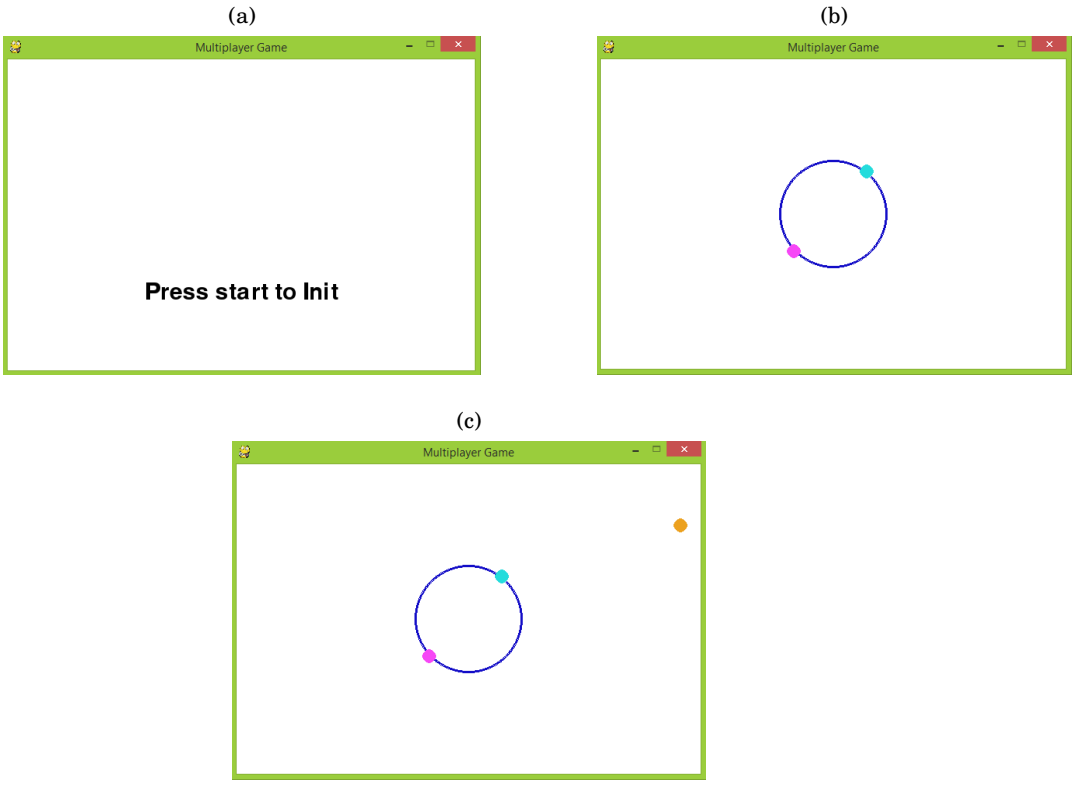

FIGURE 3.2. (a) Initial screen of the simulation. (b) Agents on the simulation. (c) Intruder and agents on the simulation.

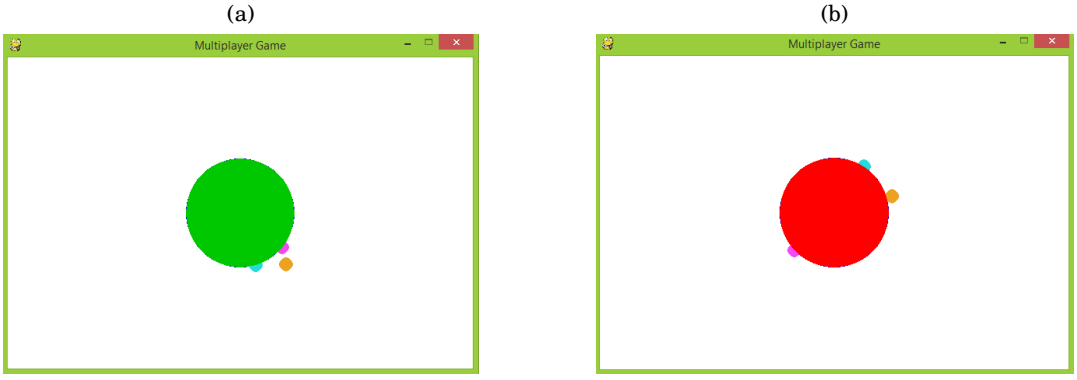

FIGURE 3.3. (a) Agents stop intruder from entering the protected area. (b) Intruder enters the protected area.

the computer programmed with the biologically inspired policy. The participants were instructed to try to win as much as possible. The information collected is again saved in .csv files and later compared against the information collected by the games played by humans.

CHAPTER 4

ALGORITHMS

An algorithm is defined as a set of rules that need to be followed in order to solve a problem or accomplish a task (Press,). The objective of this research is to compare two algorithms that have the goal of protecting an specific area. The first algorithm that will be studied is the well known minimax algorithm, which is formulated for two-players based on zero-sum game theory, this algorithm was previously studied by Coluccini on march 2016 (Coluccini,). The second algorithm is a new approach using supervised learning to find a biologically inspired policy.

In this chapter the minimax algorithm will be described including all the assumptions made by Coluccini in section 4.1, followed by the description of the algorithm used to classify the information acquired by humans in order to obtain a biologically inspired policy in section 4.2.

4.1 Minimax

The traditional minimax algorithm (Russell,) constructs a search tree as a way to represent points of decision for two players agents (considered as a single player) and intruder. The search starts at the current state and decisions are generated based on possible movements for both players until no more decisions are available or a specified depth is achieved. The number of leafs that can be generated is related to the number of available states that can be achieved, this can be previously specified selecting the depth of the tree N.

A utility is computed using a heuristic and given to leaf nodes of the tree and is propagated back. One player is called maximizer and its function is to pick the decision that leads to the best possible outcome considering the moves that the opponent can make. The other player is called minimizer and as the name implies its purpose is to find the decision that minimize its

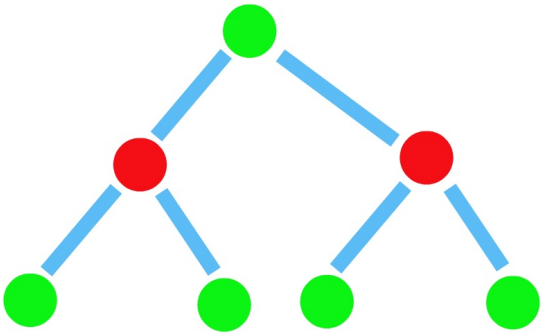

FIGURE 4.1. Minimax search tree, the player who is maximizing the utility is represented by the green dots and the minimizer is represented by the red dot

utility. Once defined the players, the tree is started to be built taking the first decision for the maximizer and then for the minimizer, repeating the process at next iteration. This provide a real-time way to obtain agents decisions. Note that the actions of players are taken sequentially, which approximates the real scenario where decisions are taken simultaneously. Figure 4.1 shows a structure of a decision tree for a minimax algorithm where the green dots represent the maximizer and red dots the minimizer.

In this research the agents will be considered as maximizer and the intruder as minimizer, following the structure proposed by Coluccini (Coluccini,). The state of the intruder as was previously mentioned had 9 possible options, the four cardinal points, four intercardinal point and the possibility to stand still. The agents have 9 different possibilities of actions combined, corresponding to individual actions of moving clockwise, counterclockwise and standing still.

The heuristic used is the one described by Coluccini (Coluccini,) where utility function is defined by the difference between the distance of intruder and agents with respect to each point of the protected area.

$$utility = \| dist_i - dist_a \|_2 = \sqrt{dist_i^2 - dist_a^2} \tag{4.1}$$

where $dist_i$ refers to the distance of the intruder and $dist_a$ to the minimum distance of the agents.

The pseudocode for this algorithm is shown below and to implement it one needs the initial position of the players (agents & intruder), the player in turn, in our case the maximizer (defenders), and the depth of the algorithm. As a result, the algorithm will return the action that the agents should take in order to prevent the intruder to enter the protected area.

input : positions of the players, turn of the player, depth
output: decision of maximizer

```
Minmax(positions, player, depth);
if node is a terminal node or depth is reached then
    score the node using the heuristic;
    return score
end
bestScoreMax = −∞;
bestScoreMin = ∞;
compute the children of the node;
for each child do
    if maximizer turn then
        score = minimizing(child, depth-1);
        if score > bestScoreMax then
            bestScore = score;
            position = child;
        end
    else
        score = maximizing(child, depth-1);
        if score < bestScoreMin then
            bestScore = score;
            position = child;
        end
    end
end
return position of the players based on the decision of maximizer
```

Algorithm 1: Minimax algorithm

The algorithm is called every time that the agents needs to take an action. The positions of agents (θ_1, θ_2, (x, y)), the player in turn (*maximizer*) and the depth is given, in this case lets consider a depth of three. Then the algorithm check if the depth is reached or if the game is over, either the defenders win or the intruder win. If non of this happens then the algorithm computes all the possible children for the defenders, which in this case are 9 and are called the children of the node.

For each possible movement of the defender (child) the algorithm needs to know based on the heuristic (which give a score) which one is the best. Therefore it calls an auxiliary function called minimizer sending the information of the child in turn and reducing in one the depth. Besides the player in turn change allowing to evaluate the movements of the intruder.

The minimizer function will check again if the pre-established depth is reached or if the game is over, the same processes will happen with the difference that now the children are related to the intruder. In order to evaluate them the maximizing function is called using the information of the intruder and reducing the depth.

This recursion will happen over and over alternating turns until the depth is reached ($depth = 0$) or the game is over. Once the depth is reached it will use the heuristic to give a score for the position and will return this value. Depending on the turn the algorithm will evaluate the score to find the maximum or the minimum score over all the possible movements. Consequently the position that maximizes the score will be chosen and returned.

```
minimizing(positions, depth);
if node is a terminal node or depth is reached then
    score the node using the heuristic;
    return score
end
bestScore = ∞;
compute the childrens of the node;
for each child do
    score = maximizing(child, depth-1);
    if score < bestScore then
        bestScore = score;
    end
end
return bestScore
```

```
maximizing(positions, depth);
if node is a terminal node or depth is reached then
    score the node using the heuristic;
    return score
end
bestScore = −∞;
compute the childrens of the node;
for each child do
    score = minimizing(child, depth-1);
    if score > bestScore then
        bestScore = score;
    end
end
return bestScore
```

Algorithm 2: Auxiliary functions for Minmax algorithm

4.2 Biologically inspired policy

In order to program a biologically inspired algorithm the data obtained from the participants was used. The simulation was played 21 times and each one was constituted by 100 trials giving in total 2100 games played. During the simulations was found that some participants made more mistakes than others and only 1287 times of 2100 the participants were able to protect the area, giving on average a rate of successful protection 64 times of 100 games played. Based on this information it was decided to separate the winning games from the loosing ones. A game is consider to be part of the group "*winning games*" when the agents manage to stop the intruder and protect the area.

Once all the games were appended into one file, the following information was found:

Variable	Observations	Min	Max
x	254,257	9	591
y	254,257	9	391
θ_1	254,257	0	360
θ_2	254,257	0	360
class	254,257	1	9

TABLE 4.1. Summary of the information acquired by the players, where Min and Max refer to the minimum and maximum position on the world for intruder (x, y) and agents (θ_1 & θ_2)

meaning that from the state of all possible actions conformed by all interactions allowed, equation 3.1, and considering that each defender make steps of $\frac{\pi}{36}$ over the circumference and the size of the world in which the intruder can move is 400x600. The state space is:

$$X = \theta_1 * \theta_2 * N * M = 72x72x600x400 = 1244160000 = 1.1x10^9 \tag{4.2}$$

Besides the information obtain from the participants is $2.5x10^5$. Hence dividing the state space over the available information ($1.1x10^9/2.5x10^5$), it is found that only 1 of every 4400 possible states can be found on the information obtained from the players.

This was not sufficient to make proper predictions and learn the policy for the defenders. In order to increase the amount of information, the similarity of the world was used to mirror the positions of the agents and intruders over the vertical and the horizontal axes and the positions of θ_1 and θ_2 where exchanged over all the points which allow increase the information on a factor of eight, giving on average 15 information points per state. The algorithm for this implementation can be seen below in Algorithm 3. To make it more visual, Figure 4.2 shows the representation of one data point mirror over the horizontal and vertical axes.

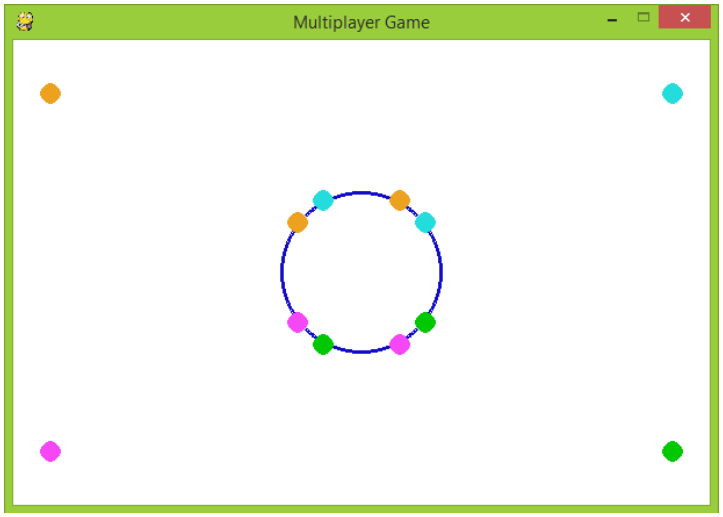

FIGURE 4.2. Increasing the amount of information by mirror the positions of agents and intruder

	Before		After	
Class	Freq.	Percent	Freq.	Percent
AA	22,846	8.99	182,332	8.96
AC	18,985	7.47	151,232	7.43
AS	32,165	12.65	213,162	10.48
CA	18,823	7.40	151,232	7.43
CC	22,737	8.94	182,332	8.96
CS	31,461	12.37	213,162	10.48
SA	21,476	8.45	213,162	10.48
SC	21,479	8.45	213,162	10.48
SS	64,285	25.28	514,280	25.28
Total	254,257	100.00	2,034,056	100.00

TABLE 4.2. Summary of the classes acquired before and after increase the information obtained

Table 4.2, shows the percentage and frequency of each class before and after increase the information where "A" is used to represent for counterclockwise, "C" clockwise and "S" stand still. From this table it is possible to see that the positions stand still is one of the most used with 25% of recurrences over the game. However it was noticed that this position mostly occurs at the beginning of each game (before the intruder appears). It was also found that no information exist of two agents completely opposite to the position of the intruder, Figure 4.3, and little or null information exist when the intruder is near to the periphery.

Based on this and considering that the information acquired was still not enough to create a

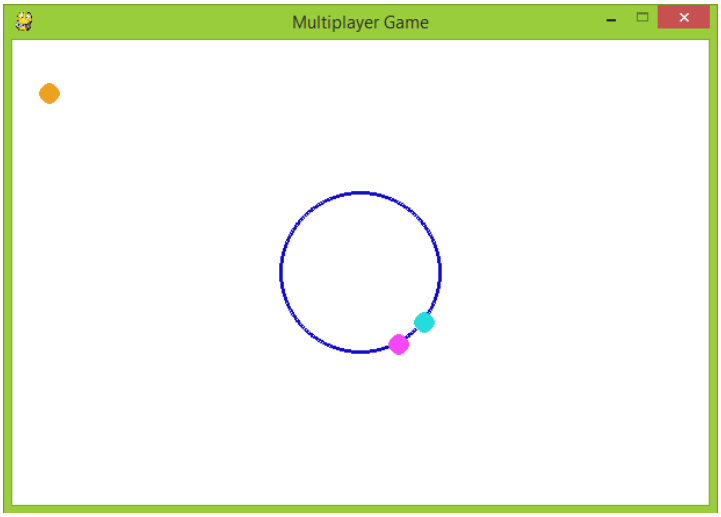

FIGURE 4.3. Positions with no information available

Class	Freq.	Percent
without	87,552	4.22
AA	293,608	14.16
AC	375,932	18.13
AS	179,869	8.67
CA	372,358	17.96
CC	339,958	16.39
CS	194,427	9.38
SA	109,034	5.26
SC	117,904	5.69
SS	2,958	0.14
Total	2,073,600	100.00

TABLE 4.3. Summary of the classes acquired by reducing the state and merged the information

functional heuristic, since only 1 of every 550 states ($1.1x10^9/2x10^6$) can be found. It was decided to reduce the state space dividing it on 15 and 10 units respectively for the Cartesian coordinates and 10 units for the angles. Allowing to reduce the state space to $X = 36x36x40x40 = 2.1x10^6$ and increasing the average amount of information per state to one. Furthermore algorithm 4 was programed to create a new matrix to store all the classes that can be encounter on each chunk. All the information is sorted and positioned in the corresponding division, and for every chunk of information the classes found are stored in the new matrix.

Considering that every chunk can contain multiple classes and that it is necessary to keep and learn from the information given by the players an adaptation using the knowledge of the

K-Nearest Neighbor classifier was done, where the minimum amount of neighbors of the same class required to made a classification in a chunk was set to five. A distance metric between classes is used to compare the movements made from both agents and the biggest class per agent was taken as the class of the chunk.

---

input : x, y, θ_1, θ_2, class
output: matrix with increased information

Load the information;
Create a new matrix;

foreach *line of the data* **do**
    matrix = Append the information;
end

`// Mirroring over the vertical axes`
foreach *line of the matrix* **do**
    if *x != 300* **then**
        x = 600 - x;
        if $\theta_1 \leq 180$ **then**
            $\theta_1 = 180 - \theta_1$;
        end
        if $\theta_1 > 180$ **then**
            $\theta_1 = 540 - \theta_1$;
        end
        if $\theta_2 \leq 180$ **then**
            $\theta_2 = 180 - \theta_2$;
        end
        if $\theta_2 > 180$ **then**
            $\theta_2 = 540 - \theta_2$;
        end

        Change class;
        'AA' → 'CC';
        'AC' → 'CA';
        'CA' → 'AC';
        'CC' → 'AA';
        'AS' → 'CS';
        'CS' → 'AS';
        'SA' → 'SC';
        'SC' → 'SA';
    end
    matrix = Append information;
end

```
   // Mirroring over the horizontal axes
foreach line of the matrix do
    if y != 200 then
        y = 400 - x;
        θ1 = 360 − θ1;
        θ2 = 360 − θ2;
        Change class;
        'AA' → 'CC';
        'AC' → 'CA';
        'CA' → 'AC';
        'CC' → 'AA';
        'AS' → 'CS';
        'CS' → 'AS';
        'SA' → 'SC';
        'SC' → 'SA';
    end
    matrix = Append information;
end
// Exchange the position of the agents
foreach line of the matrix do
    θ1 = θ2;
    θ2 = θ1;
    Change class;
    'AA' → 'AA';
    'AC' → 'CA';
    'AS' → 'SA';
    'CA' → 'AC';
    'CC' → 'CC';
    'CS' → 'SC';
    'SA' → 'AS';
    'SC' → 'CS';
    matrix = Append information;
end
return matrix
```

Algorithm 3: Increasing the information in a factor of eight

As it was previously stated, this step allow to get on average one point of information for every state on our new world, and considering that not all the states are visited during the game it was consider to make three more divisions of the state space: every time half of the previous size (20x20x18x18, 10x10x9x9 and 5x5x6x6). The previously obtained information was merged with the information obtained by the the second division (20x20x18x18), only considering to

input : x, y, θ_1, θ_2, class
output: matrix whit the number of different classes on each chunk

Load the information;
Create a new matrix filled with zeros;

```
// Defining the fractions
```

fraq = Total amount of fractions of the circumference;
nx = Total amount of fractions of x;
ny = Total amount of fractions of y;

foreach *line of the data* **do**
    n = identify the fraction of y;
    m = identify the fraction of x;
    t_1 = identify the fraction of θ_1;
    t_2 = identify the fraction of θ_2;

```
    // Find the position on the matrix
```

    position = (n-1) x fraq x fraq x nx + (m-1) x fraq x fraq + (t_1-1) x fraq + (t_2-1);

    Find the position in the matrix;
    Identify the class;
    Increase the class by 1 in the matrix;
end
return *matrix*

Algorithm 4: Reducing the state space

assign a new class when no class was found before. This was done with the four divisions starting from the smaller to the biggest one.

Table 4.3 summarize the information of the total amount of states, from which it is possible to see that it was reduced the amount of states with no information to 4.22% and stand still to 0.14%. From all the winning games it was found that in most of the cases the agents stand still just at the beginning of the game (before the intruder appears). It was also found that no information exist of two agents completely opposite to the position of the intruder, Figure 4.3, and little or null information exist when the intruder is near to the periphery.

Therefore a K-Nearest Neighbor classifier was implemented using an open source library called scikit-learn. To decide the amount of neighbors used to predict the actions of the agents, the information was divided in 80% to train the classifier and 20% to test it. Different models where tested and evaluated, from which a final decision of use 15 neighbors was taken. Finally the model was obtain using all the information.

The way in which K-Nearest Neighbor works is looking for similarities on the data. To do so a data point is given and compared with all the information. The distance between the given point and the each information point is calculated using a euclidean distance measure. The information about the distances is saved into a variable and a subset with the k smallest distances values is taken.

Once the neighbors are found, the classes of each neighbor are counted and the class with more predictions is the one which is taken. One of the biggest advantages of K-Nearest Neighbor is that it does not assume anything about the data, except for the distances calculated.

CHAPTER 5

RESULTS, DISCUSSION & CONCLUSIONS

This chapter will start describing the results obtained from the comparison of Minmax and biologically inspired algorithm (BIA), giving a better insight on how the information of players is used to make decisions and how this is used on the simulation. Further a comparison between human players and BIA will be given.

Section 5.2 will summarize the most important findings including the interpretation of the results and the limitations encounter during this research, ending with the conclusion on this research including possible applications of the findings and recommendations for further research.

5.1 Results

As the previous chapter mention a classifier was constructed based on the information obtained from the participants using only the winning games (agents stop the intruder to enter the protected area). In order to use the classifier, the information needs to be loaded beforehand. This action takes 2.9 seconds and the size of the file is 2.1Kb. Once this is done this operation does not need to be repeated. To get a prediction it is necessary to send the position of the intruder and both agents and it will return the action that each agent needs to take using the letters "A" for counterclockwise, "C" for clockwise and "S" for stand still.

Now, one of the concerning mention by Coluccini (Coluccini,) was the computational time required by the Minmax algorithm to react on a new situation. In his thesis it is stated that the algorithm requires 7.575 seconds to react, considering a limited depth of the search tree to four. Therefore a comparison between the reaction times of Minimax algorithm and Biologically

inspired one was made. The same number of random initial positions (one hundred) of the intruder used during the games were considered and the agents were kept on the same position (agent1 with an angle of 33° and agent2 with an angle of 201°). The time was measured only for the algorithms.

	Obs	Mean	Std. Dev.	Min	Max
Minmax	100	6.405401	.4157263	5.44911	7.897384
Biologically Insp. Algm.	100	.0030428	.0032968	.0009303	.0240014

TABLE 5.1. Computational time in seconds of Minmax and Biologically inspired algorithm (BIA)

Table 5.1 shows the average computational time taken by each algorithm. A T-test ($t(99) = 154.0253$, $p = 0.0000$) was conducted to compare the difference of computational time needed between algorithms and it was found that there is a statistically significant difference between the time needed by Minmax ($\mu = 6.4$, $\sigma = 0.42$) to compute which is the best next movement and the biologically inspired algorithm ($\mu = 0.003$, $\sigma = 0.003$).

In order to visualize how the algorithms are reacting to the position of the intruder, the same number of points (one hundred) was used to compare the computational time, using different colors to recognize the movements that are predicted to be done and separating them by classes. The classes are constructed using the predicted movement for agent1 and agent2 respectively. Figure 5.1 shows the predicted next movement for the agents using the biologically inspired algorithm for different positions of the intruder considering the same position for the agents.

Meanwhile, Figure 5.2 shows the predicted next movement of the agents using the Minmax algorithm. The same positions for the intruder and agents where considered in both cases. It is possible to see that the Minmax algorithm most of the time consider to make the same movement when the positions are close together. For example, positions in the left up corner are classified as "AC" which means that agent1 one will move anticlockwise to its position and agent2 clockwise.

In the case of the biologically inspired algorithm we can see that even when the position of the intruder are close together not always the same class is predicted. Leading us to conclude that each person reacts differently on certain situations. Table 5.2, shows the frequency on which each class is chosen for both algorithms, from here we can see that mostly persons do not consider stay on the same position, no matter where the intruder is. And from the simulation of the biologically inspired predictions it is possible to say that humans do not stand still but jump between two

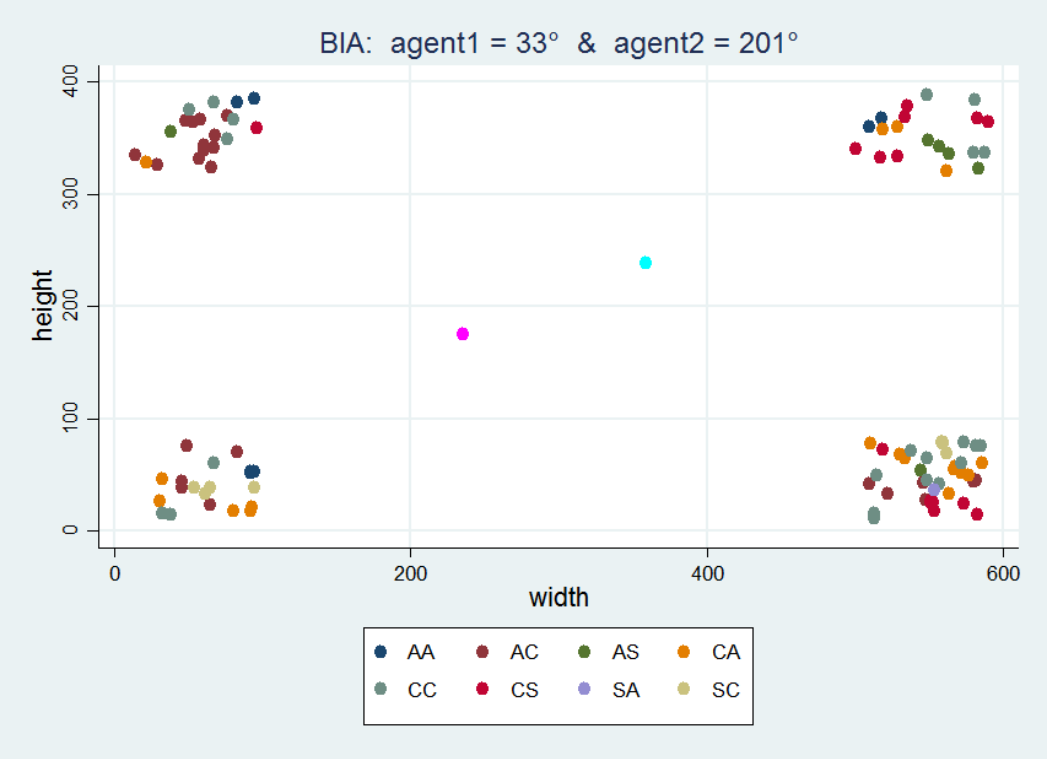

FIGURE 5.1. Classification of different positions for the intruder using the predictions of the Biologically inspired algorithm. Agent1 is represented by color cyan and Agent2 by color magenta

	Predicted classification	
Class	BIA: Freq.	Minimax: Freq.
AA	7	3
AC	23	21
AS	6	0
CA	18	57
CC	22	10
CS	15	3
SA	1	0
SC	8	0
SS	0	6
Total	100	100

TABLE 5.2. Predicted movement of the agents with both algorithms: Minmax and BIA

close positions.

Moreover an analysis using information of the winning games previously stored of humans vs.

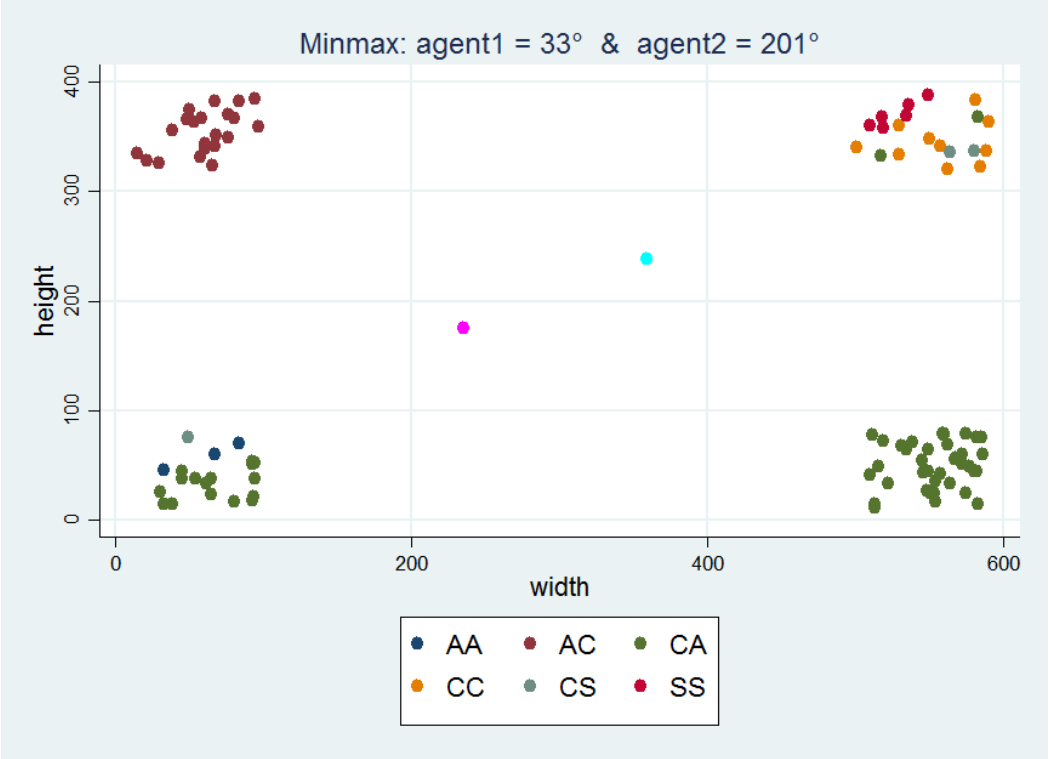

FIGURE 5.2. Classification of different positions for the intruder using the predictions of Minmax algorithm. Agent1 is represented by color cyan and Agent2 by color magenta

humans, and human vs. BIA was used for comparison. Figure 5.3, shows the average of winning games (games in which the agents stop the intruder) of human players and the biologically inspired algorithm; and it is possible to see that there is a good performance by the algorithm winning on average 16 times of every 20.

Table 5.3 shows the statistics of the game, were it is possible to see that on average human players prevent the intruder to enter the restricted area 13 times of every 20. A T-test ($t(11) = -2.4126$, $p = 0.0345$) was performed to compare the average of winning games achieved by humans($\mu = 13.1, \sigma = 3.15$) and the biologically inspired algorithm ($\mu = 16, \sigma = 1.8$) and it was found that there is a statistically significant difference between them, where in general the algorithm can secure better the protected area.

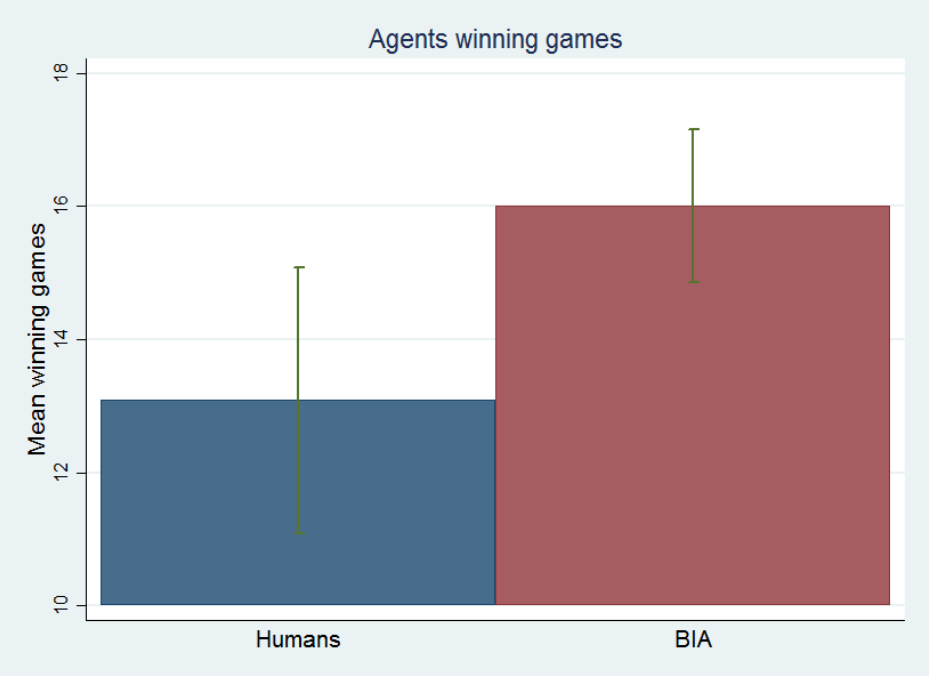

FIGURE 5.3. Comparison of the performance by humans and biologically inspired algorithm (BIA) considering the winning games

	Obs	Mean	Std. Dev.	Min	Max
Human players	12	13.08333	3.146667	8	17
Biologically Insp. Algm.	12	16	1.809068	13	19

TABLE 5.3. Number of winning games by human players and Biologically inspired algorithm (BIA)

5.2 Discussion & Conclusions

Drones industry is considered as one of the fastest-growing industries and it is predicted to keep growing by leaps and bounds over the following years; implying an increase in accessibility and availability of drones. Allowing the growth of multiple applications in industries such as agriculture, energy, mining, construction, news media and film production, among others.

Nevertheless this accessibility will also create opportunities for criminal purposes, mainly in exposed airspace where there it is no longer possible to have fences, video cameras or security guards to protect this space. Therefore, it is necessary to provide technological solutions to protect these areas.

Until now different ways to stop drones to enter a restricted area have been considered, most of them requiring a human operator. Some examples of this are the use of hawks to catch drones, implemented by the Dutch police or the use of a projectile which fires a net attached to a parachute to capture drones, developed by the british firm Openworks Engineering, Section 1.1.

However, the ability to cooperate and learn has been found extremely important and the opportunity to use autonomous multi-agent systems is expected to be suitable to guarantee a good performance to solve this task. In order to test this, the problem of protecting a specific area from an intruder was abstracted to a 2D-environment and a computer simulator was designed where two defending agents, one intruder and a protected area can be visualized.

The idea of a 2D simulated environment was inspired by the research of Coluccini (Coluccini,) where a defense task for a multi-agent autonomous system formulated as a zero-sum game was proposed. In his research, Coluccini determined the movement of the defending agents from the outcome of a Minmax algorithm, where a heuristic proposed by himself was used.

Coluccini found that the computational time required to make a decision, on which movement is the optimal, was too big for real-time applications. Besides the algorithm is fully dependent on the heuristic, meaning that a bad formulation of this function can lead to undesirable and unpredictable behaviors for the agents.

Considering this, a new strategy was proposed. Aiming to find a policy, using information from human behavior on a cooperation task, to obtain a biologically inspired heuristic. Which also reacts faster than the previously proposed Minmax algorithm by Coluccini.

As previously mentioned, a computer simulator was programmed as game that highlights the main features of a defense system. It was developed in python and 21 different teams of participants were asked to play as defenders and intruder to obtain the heuristic.

A biologically inspired algorithm (BIA) was developed based on the idea of K-Nearest Neighbor algorithm, and it was compared against Minmax. And it was found that the BIA is 2000 times faster than Minmax algorithm on computational time for decision making, Section 5.1.

The difference on computation time between the algorithms lies on the amount of calculations that needs to be done. Considering an example where one intruder has 9 possible movements and two defending agents have 3 possible movements each, combined 9; the amount of data that needs to be analyzed by the Minmax algorithm scales with the number of option to the power of

depth, which in this case is 9^4 compared to the biologically inspired algorithm which only needs to made one operation to understand the position of the players.

The winning chance of the BIA cannot be compared well with the Minmax, because Minmax is not fast enough to react in a game. However when sufficient time is given to Minmax it is expected to win every game which is better than BIA. Therefore it was decided to compare the performance of BIA against humans.

For this, 6 different participants were asked to play as intruders against the BIA. The performance of the algorithm was measured comparing the winning games (game in which agents do not allow the intruder to enter the protected area) achieved by human players against the BIA. And it was found that on average the BIA ($\mu = 16, \sigma = 1.8$) performs better than human ($\mu = 13.1, \sigma = 3.15$) players and with less variation in the amount of the winning and loosing games.

Considering all the aforementioned, it is found that for situations where it is necessary to think multiple steps in advance and where the possible options to consider are many, a BIA is better than Minmax algorithm since on this scenario most of the times the Minmax algorithm will not be able to compute all the necessary options to choose a good move.

However, not everything is good for the BIA. Since it is found that for situations where there is a representative amount of information, the algorithm is able to predict steps ahead, because it is based on human behavior. But for the same reason, the algorithm makes errors in the situations where there is little information available. This situations are the ones that humans try to avoid.

The lack of information in certain situations makes it difficult for BIA to predict what to do when these situations appear; it may be possible to enhance the BIA by making a simple heuristic that can lead the algorithm to situations where there is a lot of information available. So, it will be expected that the complex decisions are covered by humans and the not so difficult ones by an heuristic. Preventing the problem of the definition of the heuristic.

On average humans react better when they have more time to process the information leading to better decisions. However BIA's will react as good as always no matter the time given, which is one of the advantages and it can be applied on situations where there is a process that needs to be done on a high speed and humans are not able to react on time. In this case, the process can be simulated using a lower speed to gather the information, then the BIA can be acquired and used on the required speed, since the BIA can react much faster than humans.

One possible application that is considered, is to use BIAs for video games, where the necessary information can be acquired from multiplayer games allowing the possibility to learn from human behavior how to perform in specific situations.

In short, the present study demonstrates the advantages of using biologically inspired algorithms compared to Minmax in problems where anticipation is required and where there are a lot of possible movements.

Moreover, it shows how BIA can perform comparable to an average person. Intuitively, this follows from the fact that the BIA relies only on data from humans corresponding to good behavior. However it is still necessary to enable the algorithm to keep learning with every game played and to research until which point and situations the proposed method can be applied.

BIBLIOGRAPHY

Breiman, L. (2001). Random forests. *Machine Learning*.

Caruana, R., Niculescu-Mizil, A. (n.d.). An empirical comparison of supervised learning algorithms. *Department of Computer Science, Cornell University*.

Coluccini, R. (2015). *Navigation and collision avoidance algorithms for a cooperative drone defence system*. Politecnico di Torino.

Cortes, C. . V. V. (1995). Support-vector networks. *Kluwer Academic Publishers, Boston. Machine Learning*.

Cover, T. M. . H. P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems*.

E, S. (2003). Human cooperation: perspectives from behavioural ecology. *In: Hammerstein P (ed) Genetic and cultural evolution of cooperation. MIT Press, Cambridge*.

Group, T. (n.d.). *Press release: Uav production will total 93 billion.* Retrieved from `http://www.tealgroup.com/index.php/teal-group-news-media/item/press-release-uav-production-will-total-93-billion`

Harrington, P. (2012). *Machine learning in action*. Manning.

Honkela, A. (n.d.-a). *Multilayer perceptrons.* `http://users.ics.aalto.fi/ahonkela/dippa/node41.html`.

Honkela, A. (n.d.-b). *Nonlinear switching state-space models.* `http://www.hiit.fi/u/ahonkela/dippa/dippa.html`.

Insights, R. (n.d.). *Commercial unmanned aerial systems (uas), market shares, strategies, and forecast, worldwide 2015-2021.* `http://www.radiantinsights.com/research/commercial-drones-highways-in-the-sky-commercial-unmanned-aerial-systems-uas-market/request-sample`.

Jensen, W. a. (2008). *Decision trees for business intelligence and data mining: Using sas enterprise miner*. Technometrics.

Journal, W. S. (n.d.). *Serbia, albania soccer game abandoned after drone incident.* `https://youtu.be/uE5f0_FaS_k`.

Kulkarni, G., Sanjeev R. & Harman. (2011). Statistical lerning theory: A tutorial. *Princeton University*.

LaValle, S. M. (2015). *Planning algorithms*. University of Illinois.

Murphy, K. P. (2011). *Machine learning a probabilistic perspective*. The MIT Press Cambridge, Massachusetts.

News, X. (n.d.). *Drone carrying drugs crashes near the us-mexico border*. https://youtu.be/MuxphH6t7rU.

One, J. N. (n.d.). *Drone crashes in front of german chancellor angela merkel*. https://youtu.be/qKV6g47hgRs.

OpenCV. (n.d.). *Introduction to support vector machines*. http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html.

Org, S. (n.d.). *Structural risk minimization & vc dimensions*. http://www.svms.org/.

Panait, S., L. & Luke. (2005). Cooperative multi-agent learning: the state of the art. *J Auton Agents Multi-Agent Syst*.

Perrier, B., Jean-Sebastian & Anctil. (n.d.). *The projection method for finding nearest neighbors*. http://pages.videotron.com/djihess/knn/ProjectionMethod.html.

Press, C. U. (2008). *Cambridge online dictionary*.

Rafaello, E. G. M. . D. (2006). A decomposition approach to multi-vehicle cooperative control. *ScienceDirect, Robotics and Autonomous Systems*.

Raschka, S. (n.d.). *Machine learning faq*. http://sebastianraschka.com/faq/docs/logisticregr-neuralnet.html.

Russell, P., Stuart & Norvig. (2013). *Artificial intelligence a modern approach*.

Samuel, A. L. (1953). Some studies in machine learning using the game of checkers.

Schapire, R. E. (2013). Explaining adaboost. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*.

Sewell, M. (2008). Ensemble learning. *IEEE/ACM*.

Singh, S. . G. P. (2014). *Comparative study id3, cart and c4.5 decision tree algorithm: A survey*. International Journal of Advanced Information Science and Technology.

Stobiecki, P., Sniezynski, B. (2014). Training example generation method for supervised learning agents in sequential scenarios. *Procedia Computer Science 35*.

Sutton, R. S., Barto, A. G. (2012). *Reinforcement learning: An introduction*. The MIT Press Cambridge, Massachusetts.

Tang, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. *In Proceedings of the tenth international conference on Machine Learning, Morgan Kaufmann*.

Times, N. (n.d.). *Drones come within meters of planes at schiphol*. http://www.nltimes.nl/2016/04/04/drones-come-within-meters-of-planes-at-schiphol/.

Vapnik, V. (1992). Principles of risk minimization for learning theory. *Advances in neural information processing systems*.

Vidhya, A. (n.d.). *Ensemble modeling*. https://www.analyticsvidhya.com/blog/2015/09/questions-ensemble-modeling/.

Xindong Wu, J. R. Q. J. G. Q. Y. H. M. G. J. M. A. N. B. L. P. S. Y. Z.-H. Z. M. S. D. J. H.,

Vipin Kumar, Steinberg, D. (2007). Top 10 algorithms in data mining. *Springer-Verlag London Limited*.

Zhang, H. (2004). The optimality of naive bayes. *American Association for Artificial Intelligence*.