Use of Graphs with Bayesian Networks and Q-Learning for Pass-Into-Space Decision Making in RoboCup Soccer Simulation 2D

by

Alejandro Garza Cuéllar, B.Sc.

Thesis

Submitted to the Graduate Program in the

School of Engineering and Information Technologies

in partial fulfillment of the requirement for the degree of

Master of Science

in

Intelligent Systems

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

November 2012

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

School of Engineering and Information Technologies

Graduate Program

Thesis committee members recommend the present thesis of Alejandro Garza Cuéllar to be accepted in partial fulfillment of the requirements for the **Master of Science** degree of:

Intelligent Systems

Thesis committee:

Leonardo Garrido Luna, Ph.D.
Thesis Advisor

Ramón Brena Pinero, Ph.D.
Examiner

Hugo Terashima Marín, Ph.D.
Examiner

Ramón Brena Pinero, Ph.D.
Graduate Program Director

November 2012

To all the people who have believed in me.

Acknowledgements

I would like to extend my sincere thanks to all the people that supported me during these last two years. I had the privilege of being taught by many amazing teachers and shared ideas with outstanding colleagues.

I want to thank the Tecnolgico of Monterrey, who provided me with a scholarship for my Master's degree studies. Without that scholarship, I would not have been able to pursue my goal. My sincere gratitude as well to the Mexican government and CONACyT for their financial support.

I would like to thank my thesis advisor, Ph.D. Leonardo Garrido, who granted me an opportunity by accepting me into the Autonomous Agent Research Group at Tecnolgico de Monterrey. I was encouraged and guided by him on the elaboration of this thesis. He also provided me the opportunity of working with great colleagues at the Research Group. I met many incredible minds with great aspirations, goals and ideas.

Alejandro Garza Cuéllar

Instituto Tecnológico y de Estudios Superiores de Monterrey
November 2012

Use of Graphs with Bayesian Networks and Q-Learning for Pass-Into-Space Decision Making in RoboCup Soccer Simulation 2D

Alejandro Garza Cuéllar, M.Sc.
Instituto Tecnológico y de Estudios Superiores de Monterrey, 2012

Thesis advisor: Leonardo Garrido Luna, Ph.D.

This thesis analyzes the problem of decision making in a soccer environment, specifically that of the RoboCup 2D simulation. When an agent has the ball, he must make the decision to pass the ball, run with it, shoot or wait for a better opportunity. Furthermore, when passing the ball this can be done directly to an agent or a pass into space can be used. A pass into space is a pass done into an empty place of the field, where a teammate will run to and get the ball. This particular problem is of special interest since it involves cooperation between the agents in order to make plays that lead into a goal.

In order to solve this problem, it is proposed the implementation of a Bayesian Network to establish the weights of a graph in order to use it for decision making in the *RoboCup 2D* environment. Once the Bayesian Network is used to obtain the weights, the minimum spanning tree is obtained and a different network is used to recalculate the weights in order to make decisions that avoid suboptimal decisions for the game. Also, Q-learning is used to let the agents update their networks in order to learn to play against the opponent they are facing. For this work, offline learning was implemented.

To test the proposed method, games were played between teams that had the same basic structure. Yet the decision making algorithm is different for each. The one proposed in this thesis is revised first without the use of reinforcement learning against a team that uses a graph for decision making. Games are then played implementing Q-learning when playing against the team using only graphs. Finally, in order to prove that it avoids suboptimal decisions, games are played using only one Bayesian Network for calculating the weights of the graph, one more time versus the team using only a graph. To test how well the team fares when using the proposed method, several statistics are measured on each game: ball possession percentage, goal ratio and difference, total and completed passes.

It is shown in this work that the use of this approach for graphs can greatly improve the results obtained in decision making that when using a simple weighted graph for this task. As well as the advantage gained from training a team's network using Q-learning.

Contents

Acknowledgements **v**

Abstract **vi**

List of Tables **x**

List of Figures **xi**

Chapter 1 Introduction **1**

1.1 Motivation . . . 1
1.2 Problem Definition . . . 2
1.3 Hypothesis and Research Questions . . . 3
1.4 Objectives . . . 4
1.5 Methodology . . . 5
1.6 Contributions . . . 5
1.7 Thesis Organization . . . 5

Chapter 2 Background and related Work **7**

2.1 RoboCup . . . 7
2.1.1 RoboCup @Home . . . 7
2.1.2 RoboCup Rescue . . . 8
2.1.3 RoboCup Soccer . . . 8
2.2 RoboCup Soccer Simulation League 2D . . . 9
2.2.1 RoboCup Soccer Server . . . 9
2.2.2 Agents . . . 10
2.2.3 Coach . . . 12
2.3 Graphs . . . 13
2.3.1 Minimum Spanning Tree . . . 14
2.3.2 Kruskal's Algorithm . . . 14
2.4 Bayesian Networks . . . 17
2.5 Reinforcement Learning . . . 19
2.6 Related Work . . . 20

2.7 Summary 22

Chapter 3 Proposed Solution: Graphs with Bayesian Networks and Q-learning 24

3.1 A Glimpse of the Proposed Solution 24

3.2 Graph Construction 31

3.2.1 Calculating the Weights 34

3.2.2 Solving the Graph 36

3.3 Using a Bayesian Network for Calculating the Weight of the Edges . . . 39

3.4 Guessing of the Pass into Space 45

3.5 Using Q-learning for updating the Bayesian Network 46

3.6 Summary 50

Chapter 4 Experiments and Results 51

4.1 Experiments 51

4.2 Use of MST vs. Base Team 57

4.3 Use of no-MST vs. Base Team 60

4.4 Use of Q-MST vs. Base Team 63

4.5 Summary 66

Chapter 5 Conclusion 67

5.1 Research Summary 67

5.2 Research Questions Revisited 68

5.3 Future Work 70

Bibliography 71

Vita 74

List of Tables

2.1 Adjacency Matrix for the graph on Figure 2.1. 14
2.2 Sorted weights of the graph shown in Figure 2.3. 17
2.3 Kruskal's algorithm for obtaining the MST of the graph shown on Figure 2.3. 17

3.1 Threat values on the six zones to analyze on the field. 27
3.2 Weights of the different edges in the graph. 28
3.3 Weights of the different edges in the graph using a BN. 30
3.4 Weights of the different edges in the graph using a second BN on the MST alone. 30
3.5 Weights of the different edges in the graph using a second BN on the MST alone and without the use of the MST. Although the weights of some edges is the same, now all but the strategy nodes appear. 31
3.6 Structure of the nodes in the graph. 37
3.7 Structure of the edges in the graph. 37
3.8 Structure of the weights in the graph. As can be seen, this structure is just an extension of E. 37
3.9 Different parameters of the agents shown in Figure 3.11 38
3.10 LogicAction probability table for a specific Position on the field. Note that these values present a balanced, non risky play style 42
3.11 Risk probability table for GameTime=Starting 42
3.12 Risk probability table for GameTime=Half 43
3.13 Risk probability table for GameTime=Ending 43
3.14 Matrix for the Rewards from the pairs (state, action). 47

4.1 Reduced ASCII code used to send the BN to the agents. 55
4.2 Comparison of different stats between the team that uses the MST and the team that uses a no-MST. 61
4.3 A part of the learned network after 500 games. 63
4.4 Comparison of different stats between the team that uses Q-MST, MST and the team that uses a no-MST. 63

List of Figures

2.1 Example directed graph. 14
2.2 Adjacency list of the directed graph shown in Figure 2.1. 15
2.3 Weighted graph along with its minimum spanning tree. 15
2.4 A Bayesian Network where nodes only depend upon their parentss. . . 18

3.1 Diagram of the process where two main parts can be seen: the in-game process, where all the decisions are made and the Bayesian Network training, where the trainer uses the data from the decisions to train the network. 25
3.2 Block diagram of the proposed method. 26
3.3 Situation to be solved by the agents in the game. 27
3.4 Situation to be solved by the proposed method. 27
3.5 Graph of the situation on Figure 3.4. 28
3.6 Graph after zones with high threat values are eliminated. 28
3.7 MST of the example. 29
3.8 MST of the example when using a BN. 30
3.9 In a pass, player number 1 kicks the ball to player's 2 position. 32
3.10 In a pass into space, player number 2 will kick the ball to a position to where player 1 can run to get the ball. 32
3.11 Formation: 4-3-3 that agents will be using in the games. 36
3.12 Initial Network proposed where the inputs affect the decision to be taken. The Score and GameTime are selected as modifiers since those variables affect directly the risk that can be taken by the agents. 40
3.13 The Position and CurrentState nodes are combined to determine the Logic Action that the attacker could find best to do without taking into account the whole state of the game, just their position and current action. 41
3.14 As mentioned, the score and game time give the agents a sense of how much risk con he take when making plays. 41

3.15 Bayesian Network proposed to address the problem of passes and passes into space. The *LogicAction* node is used to calculate the weights of the edges in the graph. While the output, *Action*, is used to recalculate the weights of the edges in the MST. 42

4.1 Ball possession percentage in each game when using the MST. It can be seen that this is almost always above the 50% of the time. 58
4.2 Goal difference for each of the games. It can be seen that in most of the games, the goal difference is well above 0. Also, the goal ratio for the games stood at 3.9:1. 59
4.3 Total passes per game along with the number of completed passes. The total passes average was at 30 passes per game. 59
4.4 Total passes per game along with the number of completed passes. The total passes average was above 30 passes per game. 60
4.5 Ball possession percentage in each game when using no-MST. 61
4.6 Goal difference for each of the games. It can be seen that in more than half of the games, the goal difference is above 0. The goal ratio stood at 1.5:1. 62
4.7 Total passes per game along with the number of completed passes. Using only the no-MST,the total passes average was below 15 per game . . . 62
4.8 Ball possession percentage in each game when using Q-MST. It can be seen that this is always above the 50% of the time. 64
4.9 Goal difference for each of the games. It can be seen that in most of the games, the goal difference is well above 0. Also, the goal ratio for the games stood at 6.8:1. 64
4.10 Total passes per game along with the number of completed passes. The total passes average was around 33 passes per game. 65

Chapter 1

Introduction

Since the 50's, study in the field of AI has led the world to several grand changes. Albeit at first a science that lost a lot of support during the late 70's [18], AI is now one of the most important fields of study in the world. It has grown in such a way that nowadays many sciences rely on it to accomplish several tasks. To help even more the research in this area, in 1997 the Robot World-Cup Soccer, best known as RoboCup, was created, "as a vehicle to revitalize research by offering an appealing but formidable challenge" [12]. Although RoboCup is comprised of several categories, the one of interest in this research is the 2D Soccer Simulation League (SSL). In this league, soccer games of 11 vs 11 agents are played. The agents of each team can sense the world partially and react accordingly. For the game, the same rules of soccer are used. The main difference between a real game and SSL in 2D, besides that no human player is playing, is that games take place in a 2D world. Still it is no easy task for the agents to compete against an opposing team. They need to be able to locate the ball, make decisions of whether to shoot, pass, move, etc. And they must do it fast and precisely. The main problem that is going to be tackled in this thesis is divided in two parts. First, how to decide between making a pass or a pass into space. Second, how to adapt the play style of the team in order to play better as the game progresses, using machine learning. For solving this part, an approach using a graph's MST, in conjunction with a Bayesian Network is presented. For the second problem, a reinforcement learning method is proposed. More precisely, Q-learning is used in order to let the agents play better a the time moves on.

1.1 Motivation

Some of the advantages that the RoboCup Soccer 2D simulation provides for the proper research and development of AI is due to the existence of the complete and complex environment that exists in the competition. It is, according to Russell and Norvig [18], labelled as:

Dynamic: The environment is constantly changing. The agents may take their time to think their actions. But as they do, the other agents could be acting.

Multi-Agent: The game is played by teams of 11 agents against 11 agents, so not only should the agents take into account their actions, but the actions the whole team could take in order to win.

Partially observable: The agents are only able to sense part of the environment. And not only that, but they not always are able to sense correctly what they "see."

Sequential: Although the game is divided into two semi-episodic half's, each half is completely sequential. If decision the agents or the team makes will create an advantage or repercussion almost immediately.

Stochastic: Because not only are there 11 players on the team, but also on the opposing team, an agent can never exactly know what is going to happen in the next state of the game, even though using AI they could create models to react accordingly.

This environment is ideal for research as it is never possible to make actions with full understanding or data of the world. Not only this, but as the system itself is a simulation, problems that robot may have can be set aside. There are no motors heating, no hardware problem, and beside the law of physics, no physical bound to do research on areas that due to robot limitations, would take years from now to cope with those issues and have robots that behave as the agents in the simulation. In other words, this simulation allows the research and solution of problems that otherwise would take decades to be able to even start looking at.

1.2 Problem Definition

In a soccer match, players need to decide which action they are going to take. If a player has the ball, he can pass it, make a pass into space, wait in the place he is standing or run with the ball. From these actions, the pass into space is one of the hardest to make. Not only does it require the player to evaluate the situation, but it also needs the help of another player. Thus, he either needs to verbally tell the teammate about the pass or the ally should be able to know when a pass is made for him. In the case of a multiagent environment, both would be equally good. Nevertheless the first option allows for information to be obtained by the opponents.

Although a pass into space is difficult to execute for the player, its rewards can be much higher. By sending the ball to a place in the field where there is no opponent, it is easier to break the defenses and score a goal. Thus, if a pass into space is to be done, the player with the ball needs to decide if it is the best decision. While the teammate to whom the pass is made to needs to know in anticipation about the play.

Nevertheless, the decision that the player with the ball takes needs to be the best not only for that precise moment, but for the whole game. From here, those decisions that look like the best at the moment but will not lead to the best possible play in the long run could be considered as suboptimal decisions for the game. The approach of taking into account the decisions in the near future is used by Wright Eagle, one of the best simulation teams in the world [1]. By looking one step ahead in order to make the best decision, the team can get ready for it and plan accordingly.

For solving the problem of passing and coordination, several learning algorithms have been proposed. Q-learning has been widely used to solve the problem [8], [23], [9], [15], but the nature of the environment allows for the use of multiagent techniques [10].

1.3 Hypothesis and Research Questions

As to how to solve the problem of a pass into space, a brief description is presented in here. Positions of players on the field, as well as positions of the field themselves can be seen as nodes of a graph were the ball can go from one position to another. In this case the cost for this would be a combination of distance, enemies in the trajectory, etc. But these kind of variables are not all that affect the possible actions. The stamina of the players, how much risk can the team take based on the score, how many enemies are or could there be. In order to correctly model all these, a Bayesian Network seems like a viable and logical option to use in order to determine this cost (Or in the case of graphs, the weight of an edge) due to the uncertainty that these variables present.

Bayesian networks are one of the used techniques to solve problems while coping with uncertainty [18]. The network's design models the environment and establishes conditionality and independence across several variables to be used in the probability calculation. This provides the freedom to alter the model of the network depending on the priorities to produce a different output.

Moreover, they also express variables of the domain through the use of nodes in their models. The more nodes in a BN, the more variables of the domain are being considered. Furthermore, due to variable representation and management, BN can be scaled significantly better than other combinatorial methods or non structured conjunction representation.

An advantage given by implementing this method is the possibility of generating probability values required for the network while using offline training. This enables the network's training set to include a high number of instances in order to perform a complete and meaningful training. Once the training is done, the values obtained through calculating probabilities can be used.

Bayesian networks share certain similarities with the probabilistic model of decision making that people use. In the book entitled Psychology of Patient Decision Making [2], the authors established that for taking a decision a person must be capable of evaluating probabilities from a spectrum of possibilities. Moreover, it is considered that a likelihood estimation should be made in order to proceed into deliberation in order to make a choice [24]. This process requires the conditional probabilities of the variables in the environment that affect the decision making process.

From here, the following hypothesis is proposed:

> By using Bayesian Networks to calculate the weights of a graph, better decisions can be taken by the attacker agents while the receiver will be able to know where the pass is aimed at. Also, by getting the minimum spanning tree of the graph and recalculating the weights of the edges with a different network, the decisions taken will lead to better results by avoiding getting stuck on a suboptimal decision for the game. Finally, offline learning in the form of Q-learning can be used in order to improve the results obtained when playing against an opponent.

The MST is decided to be used to reduce possible actions since it provides a graph of actions that connect all the nodes, while giving preference to those actions with high evaluations. From the hypothesis proposed in this thesis to solve the pass into space problem, the following research questions arise:

- Is it possible to use a model that lets the attacker agent know where a pass into space is being aimed at?
- Is it possible to use a model based on graphs and using Bayesian networks that lets the receiver agent know when a pass into space is for them?
- Will the agents be able to train in order to have a better play style?
- Can Q-learning be good enough for multiple games against the same team?

1.4 Objectives

The main objective of this thesis is to develop a method that can solve the problem of decision making for the player with the ball using graphs and Bayesian Networks to calculate the weights of the edges in the graph. Further, the specific objectives pursued in this thesis involve:

- Generate a Bayesian Network that is capable of correctly handling the weights of the edges in the graph in the different possible states of the game.

- Show that by recalculating the weights of a graph with a Bayesian Network after a MST is obtained can yield better results than by deterministically selecting the ”best” edge.
- Implement an offline learning method, using Q-learning in order to update the values of the BN and let the agents learn how to play even better.

1.5 Methodology

To achieve the previous objectives, the next methodology is implemented:

- Games will be played against a base team that does not uses Bayesian Networks.
- Using the output of only one node for calculating the weights of the edges, in order to prove that the method avoids suboptimal decisions for the game, games will be played against an similar team that does not uses Bayesian Networks.
- Using Q-learning, games will be played against a base team that does not uses Bayesian Networks.

1.6 Contributions

The main contribution of this thesis is the implementation of a decision making method that uses Bayesian networks with a graph to achieve better results than when using only a graph. Also, by adding Q-learning, the overall performance of the method is boosted. The method proposed in here can solve the problem of decision making while achieving good results in the RoboCup 2D environment. Although further research can be done on the implementation of the method on different domains, this will not be addressed in here but is left as future work.

1.7 Thesis Organization

This thesis is divided into six different chapters. The first presents an introduction to the problem to be solved and the methodology of how it will be approached.

The second chapter provides a background of all the concepts the reader will need to know to easily understand what is presented. On it, background about the RoboCup competition is given to the reader. Basic information about the *rcsoccersim*, the program which runs the simulations, is presented. Then all the required AI concepts used to solve the problem at hand are given. At the end of the chapter, the related work is presented.

The third chapter contains the full explanation of the method proposed to solve the problem. First it is shown how the graph of the decisions will be generated. Then, how a Bayesian Network is used to calculate the weights is presented, along with the use of Q-learning to improve the effectiveness of the proposed method. Finally, how all these techniques fit together is shown.

The fourth chapter presents both the setup for the experiments and results obtained by them. First how all experiments are to be realized. Then, the results obtained from each experiment, along with their analysis is shown. Finally, some preliminary conclusions are presented.

Finally, in the fifth chapter, the conclusions obtained from the results are presented. The research questions are revisited and answered with the results obtained from the experiments.

Chapter 2

Background and related Work

This chapter provides a background of all the concepts and terms the reader will need to know to easily understand what is presented on this thesis. First, background about the RoboCup competition is given to the reader, from a little history of it to a roundup of the different competitions that it has spanned. Then, basic information about the *rcsoccersim*, the program which runs the simulations, is presented. Then all the required AI concepts used to solve the problem at hand are given. At the end of the chapter, the related work is presented.

2.1 RoboCup

The RoboCup competition started 15 years ago at Osaka Japan with only 8 different teams participating on it, and it has gradually grown to incorporate a large number of different competitions, each with its own research focus. This competition is held every year in a different country in the world, updating rules each iteration to promote more advanced research.The goal to achieve in the RoboCup is the following:

> By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup. [12]

One of the competitions in RoboCup is the Soccer Simulation League 2D, which pits two teams of eleven players each to a soccer match in a simulated 2D environment. This competition encourages researchers to focus on IA and multiagent systems rather than loose time building or learning to use robots. The importance of this category is that since it sets aside the physical restrictions of the robots, research can be done in areas that will not be available for many more years due to hardware limitations.

2.1.1 RoboCup @Home

This relatively new RoboCup league is centered on human-machine interactions. It is focused on applications to assist humans in everyday tasks such as finding objects within the house or serving cereal for the breakfast. The environment in which this competition is held is a living room and a kitchen, yet it is planned to include other areas such as a garden, a shop or another public place [16].

2.1.2 RoboCup Rescue

The main focus of this area of RoboCup is to promote research in search and rescue technologies to be used in large scale disasters. This category is subdivided into the following leagues [17]:

Rescue Robot: Using robots that must be capable of traversing almost any kind of terrain, this leagues focused on heterogeneous agents in a hostile environment. Here robots must be prepared to search, find and rescue people where humans would have a hard time getting to. In here, several scenarios are built in which robots must complete a task that usually involves rescuing a person in distress. -

Rescue Simulation: Similar to the one mentioned before with the difference that a whole urban environment is created. In here fire fighters, commanders, victims, volunteers, etc. are simulated through intelligent agents. Images acquired by helicopters can be accessed and used to predict and plan the best course of action

2.1.3 RoboCup Soccer

This category's main focus is to present a challenge focused on a soccer match to present an appealing yet formidable challenge. The main focus of research in this category are the cooperative multi-robot and multi-agent systems. Although the robots act in a predefined field, the environment behaves in a dynamical way, since it is constantly changing. The categories that fall in the RoboCup Soccer are [17]:

Humanoid: Each team develops its own humanoid robot design and deals with the problem of having them play against a totally different adversary. Of course certain restrictions like size must be met.

Middle Size: In this category, robots in teams of 6 play with a regular size FIFA soccer ball. In this case, every robot is equipped with an on-board computer to analyze the game situation.

Simulation: This category is purely focused on AI and team strategy since no actual robots are used. Simulation can take place in a 2-D environment, where physics dont play a big part on the field since there is no actual walking or running; or in a 3-D one, where the simulation is more realistic since virtual Nao robots are used to play the game.

Small Size: The game takes place between two teams of five robots each. In here, robots have a cylindrical shape. Instead of using cameras or other sensors on the robots, a global vision system is used by placing a camera above the soccer field, and the data is processed by an off-field PC.

Standard Platform: All teams use the same robot, Nao from Aldebaran Robotics, to let research focus itself on the AI and software development. There is no external control of the robots, neither by humans nor by computers since the robots are fully autonomous.

2.2 RoboCup Soccer Simulation League 2D

The 2D simulation league is based on a simulator called the "soccer server" [12], a physical simulation system. The games are displayed on a monitor in a computer screen in "real time." Each team consist of 10 players and a goalkeeper, just as in a real soccer match. Games are played in two halfs and rules very similar to those of the real game are used.

Each team is comprised by agents that play in the game and a coach. While the agents get only limited information from the game, the coach can get the information of the full state of the world and can be used for online learning during a match. Besides the coach there is also an offline coach called the trainer that not only can get as much information as the coach but can also change states in the game. Yet the trainer can only be used outside of the official competition.

Information of the competition, regarding its capabilities and limitations can be found on the RoboCup Soccer Server user manual [4]. Also, the information provided in the following subsections can be obtained in more detail on it.

2.2.1 RoboCup Soccer Server

This system enables autonomous agents to play a match of soccer against each other. The agents use a program that can be written in various languages such as C++ or Java. The server functions in a client/server style. While the server creates a virtualization of a soccer field along with the physical laws that must be applied, the clients control the decisions the agents can take. This communication is done in UDP/IP sockets.The soccerserver consist of 2 programs: the soccerserver and the soccermonitor.

The soccerserver simulates the movements of the players, the ball, handles the communication between clients and ensures the rules of the game are applied. The soccermonitor takes care of displaying the field and the actual position of each agent and the ball according to the data from the soccerserver.

Each client can send commands to a player and receive information from the sensors of that particular agent using UDP sockets. Therefore, the clients act as a brain for the players. Sensorial information is received from the outside and is the processed to act accordingly. The information the agents receive from the server is not perfect. The server manages uncertainty when obtaining data from the world as agents obtain noisy information.

2.2.2 Agents

Agents are the ones that play in the match. Each agent can sense the world around them but only in a limited sense, just as an actual robot could. The agents posses the following sensors [4]:

Aural Sensor

Detects messages sent by the referee the coaches and other players. The parameters that affect the aural sensor are:

hear_decay: A player can only hear a message if his hear capacity is at least this parameter. Each time a message is heard the hear capacity of the player is decreased by that amount.

hear_inc: Increases the hear capacity of the agent every cycle by this amount. *hear_max*. Delimits the maximum hear capacity

audio_cut_dist: Determines the range at which a message can be heard. Only those agents in this range will be able to hear the message.

Vision Sensor

Gets the objects that the player can see. This information is automatically sent to each player. The information the players get is [4]:

distance: Relative distance between the player and the seen object; *direction*, the relative direction of the object to the player's body.

distance change: How much the distance changed from the last cycle.

direction change: How much the direction changed from the last cycle.

body direction: Absolute facing direction of the body of the objective.

head direction: absolute facing direction of the head of the objective.

What the agent can see is limited by:

visible_angle: which determines how many degrees the visual cone covers. It is possible to change this angle.

visible_distance: Just as the aural sensor, the vision sensor has, this delimits what the agent can "see." However even within this range, depending on the distance not all information will be available for the agent.

Body Sensor

This sensor reports the "physical" state of the player. As in the vision sensor, this information is automatically sent to the agents. The agents get the following information [4]:

stamina: Just as the name suggest, this variables tells the agent how much stamina they have left. Stamina is used to make any physical action in game. The more actions an agent makes, the more the stamina decreases, the more they stay idle, the more it recovers.

speed: Which gives the *AmountOfSpeed* that is an approximation of the player's speed and *DirectionOfSpeed* which is and approximation of the player's speed.

head_angle: Returns the relative direction of the player's head.

Several counters are given to the player, which keeps how many commands of that type have been used, the counters are: *kick*, *dash*, *turn*, *say*, *turn_neck*, *catch*, *move*, *change_view*.

Besides from the sensing of the world, agents can also perform different actions in game. These are described as following:

Move: As the name suggests, this action lets the agents move in the field. Agents can move at different speeds.

Catch: This action is unique to the goal keepers, as it lets them catch the ball.

Dash: This action lets the agents accelerate in the direction of his body. Deacceleration is also possible in order to slow down the players.

Kick: This action lets the agents kick the ball if it is close enough, to a selected direction.

Say: Lets the agents "say" a message for other players or coach to listen to. The information is sent as text.

Hear: Lets the agents hear the messages of other players, the server or the coach.

Turn: Lets the agents turn the agents in a desired direction.

Turn Neck: Lets the agents turn only their necks, maintaining their body positions.

2.2.3 Coach

The coach is a special type of client that is used to provide assistance to players. There are two kinds of coaches, the one used for official matches or online coach and the trainer or offline coach. The trainer has more privileges than the online coach as he can move objects at will in the field, can change the state of the game etc [4]. Nevertheless, the second can only be used in the development stage and not in an actual game. But he is useful for use in machine learning and managing games as was done in this thesis.

The trainer has the following capabilities:

- Change the play-mode.
- Send audio messages for one or more agents.
- Move the agents or the ball to any position on the field. Also, he can set their direction and speed.
- Get noise free information from the state of the world.

In contrast, the online coach is designed to observe the game and provide advice and information to the players. The capabilities of the online coach are:

- Send messages to the agents.
- Get noise free information from the state of the world.

Unlike the agents that only get limited information from the world according to their sensors and their current condition, the coach can get noise-free information from the whole world state, which lets him create better models to train the agents. The information the coach can get is the following:

- For goals: X, Y.
- For the ball: X, Y, X_{speed}, Y_{speed}.
- For agents: X, Y, X_{speed}, Y_{speed}, bodyAngle, neckangle, pointingDirection.
- Team names.

- Side of each team.
- Messages from the server and both teams.

In this thesis the trainer was used to get information from the world and manage the machine learning between games. The techniques used will be properly explained in chapter 3.

2.3 Graphs

As defined by Base [3], a graph, informally, is a finite set of points, some of which are connected in one or two directions. The points defined in the set are known as *Nodes*, while the connections between these are *Edges*. When the edges only go in one direction, the graph is said to be *directed*, or a *digraph*. Meanwhile if the edges can go both ways, it is said to be an *undirected graph*.

The nodes in the graph can represent a wide array of things, for example places within a city, while the edges represent the physical way of reaching those places like a road or a bridge. This particular problem was presented by Euler in 1735 as the Konigsberg Bridge Problem. Yet, the nodes can symbolize a wide array of things, even players in a soccer field while the edges are the physical path that connects them. It is also possible to assign a cost or weight to the edges that connect two nodes. When this is used in a graph, it is said to be a *Weighted Graph*.

Mathematically, a graph is defined as:

$$G = (V, E, W) \tag{2.1}$$

Where:
V are the Nodes or Vertices of the graph.
E are the Edges.
W are the Weights.

In the example shown in Figure 2.1 of a *digraph*, the sets that compose the graph would be:
V = {1,2,3,4,5}
E = {(1,2),(2,3),(3,2),(3,5),(4,1),(4,5)}

Besides the formal mathematical definition, when programming, there are two ways of to represent a graph in a computer. Using an adjacency matrix or using an adjacency list. As the name suggests, these represent what connections exist between the nodes in the graph. An adjacency matrix, as shown in Table 2.1 uses 1's to represent a connection from one node to another and 0's otherwise. Although an adjacency matrix is very easy to construct, for a graph with n nodes, it would take *O(*n^2*)* to do so. Unlike

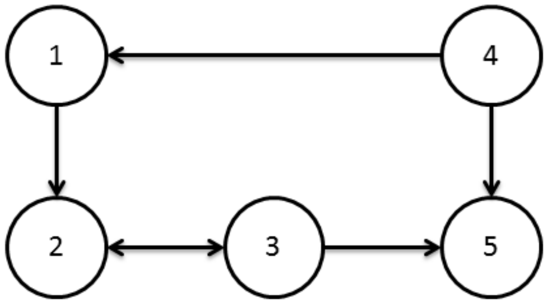

Figure 2.1: Example directed graph.

the adjacency list, that although is a little more complex, only takes *O(n)* to get. An adjacency list is shown on Figure 2.2.

node	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	0	0
3	0	1	0	0	1
4	1	0	0	0	1
5	0	0	0	0	0

Table 2.1: Adjacency Matrix for the graph on Figure 2.1.

2.3.1 Minimum Spanning Tree

For an *undirected graph* $G = (V,E,W)$, a *spanning tree* is a subgraph of G that contains all vertices of G and is an *undirected tree*, where the weight of a subgraph is the sum of all the weights of the edges in the subgraph. A *Minimum Spanning Tree*, or *MST*, is the spanning tree with the minimum weight [3].

To solve the problem of finding the MST of a graph, two algorithms come to shine as the most known: Dijkstra's algorithm and Kruskal's. Although both are very good algorithms that run in $O(ElogV)$, the first is a greedy search algorithm while the second uses a sorting method to be able to directly select the edges that compose the MST, making it more easy to program. Figure 2.3 shows a weighted graph along with its MST.

2.3.2 Kruskal's Algorithm

The main idea of Kruskal's algorithm is to always select the edges that have the lowest weight, in order to create the MST. Yet, as the algorithm is selecting the edges,

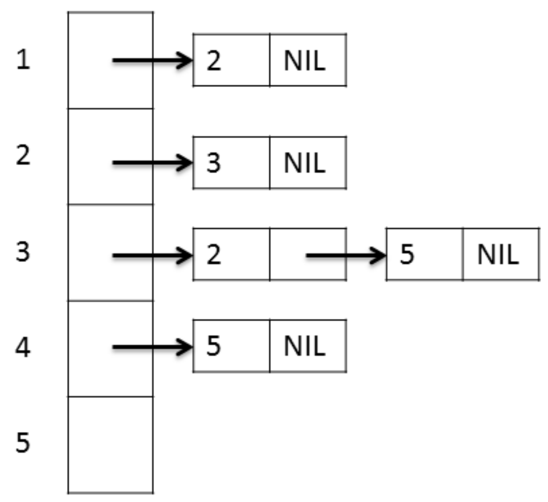

Figure 2.2: Adjacency list of the directed graph shown in Figure 2.1.

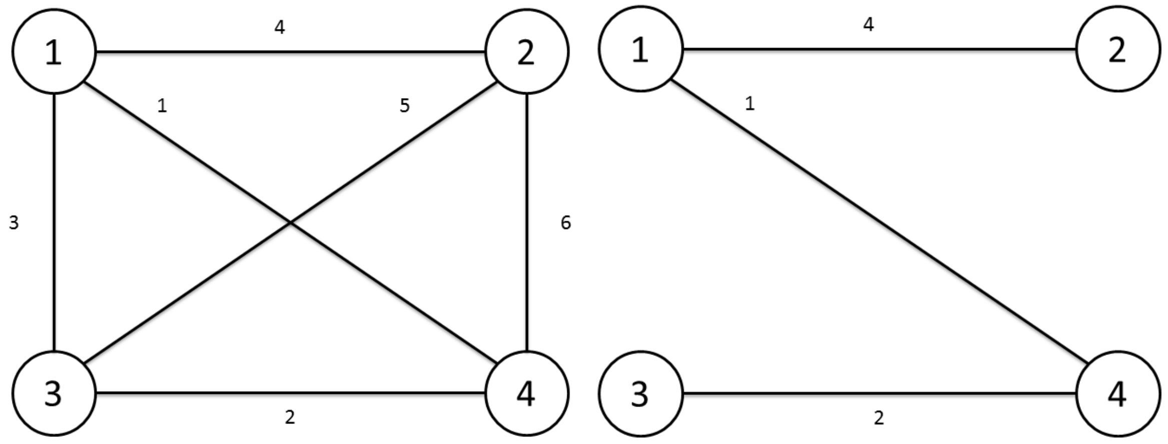

Figure 2.3: Weighted graph along with its minimum spanning tree.

some rules must be followed in order to avoid violating the principles that an MST have [3]:

- If the edge would form a cycle with those already chosen, it is discarded. If a cycle is formed, then it would no longer be an *undirected tree*.
- At any time, the edges chosen so far will form a forest but not necessarily one tree.
- Since the MST is an *undirected tree*, the subgraph is composed of *n-1* edges, where n is the order of G (number of nodes).Hence, once there are *n-1* selected edges, the algorithm finished.

A pseudocode for this algorithm is presented in algorithm 2.1. Although it may look hard to code due to revising if a cycle is formed in the forest, it actually is very simple to do so using the following:

> Let *F* be the forest which contains nodes *V'* from the graph. If an edge *e* connects two nodes from *V'*, then that edge creates a cycle.

From here, the algorithm can be presented in such a way that it can be easily implemented, as seen on algorithm 2.2

Algorithm 2.1 Kruskal's Algorithm

```
R ← 0 // remaining edges
F ← 0 // forest edges
while R is not empty do
  remove the edge $e$ with the minimum weight
  if $e$ does not make a cycle in $F$ then
    Add $V(e)$ to $F$
  end if
end while
```

Algorithm 2.2 Kruskal's Algorithm to implement

```
R ← 0 // remaining edges
F ← 0 // forest edges
E ← Sort(E,W) // edges sorted by their weight
n ← number of nodes - 1
while R is not empty and Order(F) < n do
  remove the first edge $e$ from E
  if Nodes connected by $e$ are not in $F$ then
    Add $e$ to $F$
  end if
end while
```

Taking the graph shown on Figure 2.3 as a reference, its MST can be obtained using Kruskal's algorithm. To do so, first the weights need to be sorted. The sorted weights are shown on Table 2.2. The first selected edge is (1,4), since it has the lowest value. The next edge is (3,4) and since node 3 is not on the tree, then the edge is added. Edge (1,3) is the next on the list but both its nodes are already on the tree, so it is not added. Next comes edge (1,2) which has the node 2 that is not on the tree and is added. At this point, all nodes are already on the tree, therefore the MST has been found. This process is shown on Table 2.3.

Edge	Weight
(1,4)	1
(3,4)	2
(1,3)	3
(1,2)	4
(2,3)	5
(2,4)	6

Table 2.2: Sorted weights of the graph shown in Figure 2.3.

Edge	Weight	Nodes in MST	Added
(1,4)	1	{}	Yes
(3,4)	2	{1,4}	Yes
(1,3)	3	{1,3,4}	No, nodes 1 and 3 already in tree
(1,2)	4	{1,3,4}	Yes
(2,3)	5	{1,2,3,4}	No, MST completed
(2,4)	6	{1,2,3,4}	No, MST completed

Table 2.3: Kruskal's algorithm for obtaining the MST of the graph shown on Figure 2.3.

2.4 Bayesian Networks

A Bayesian Network (BN) or belief network, is a model used to represent a specific problem along with its variables and mutual relationships. BN are used in order to cope with uncertainty at a reasoning process. Their implementation is included in decision theory for risk analysis and prediction in decision making. As defined by Russell [18], a Bayesian network, or BN is a digraph in which each node is annotated with quantitative probability information. A BN is defined by the following:

- Each node corresponds to a random variable, which may be discrete or continuous.
- A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be a *parent* of Y. The graph has no directed cycles(and hence is a directed acyclic graph).
- Each node X_i has a conditional probability distribution $\mathbf{P}(X_i|\text{Parents}(X_i))$ that quantifies the effect of the parents on the node.

The topology of the network specifies the conditional independence relationships that hold. An advantage of using Bayesian Networks is that only the conditional probability of each node is needed to define the network. In contrast to a full joint probability distribution, where all probabilities need to be specified. In Figure 2.4 a BN along with its conditional probability table (CPT) is shown. As can be seen, the values that need to be specified for each node only depends of the parent nodes and not any other.

In a BN, the nodes are independent to those not connected to them. Yet, conditional dependence or independence may raise given evidence of a variable. For example, in Figure 2.4 D and E are dependent, since both come from the same parent C. But if C is known, then both nodes can be treated as independent. In this case, conditional independence is given by the knowledge of the parent.

Another case present on Figure 2.4, is the relation between A and D. The latter is dependent upon the first since it is a parent of its parent. Nevertheless, if C is known, then A and D become conditionally independent since it is now know what is causing them. The last case shown on the figure, is the relation between A and B. Normally, this two nodes would be independent, but if evidence from C arises, they become conditionally dependent.

Bayesian Networks use two different type of probabilities: prior and conditional probabilities. The first are those associated with each individual variable, the probability of a variable being in a state without any given information. For example, from Figure 2.4 the prior probability of A would be: P(A) = 0.001. The conditional probability is the one associated with an event happening given certain evidence or information. For example, the probability of C given A and B would be: P(C—A,B) = 0.95. Using Bayes' theorem, the conditional probability can be calculated:

$$P(h|e) = \frac{P(e|h)P(h)}{P(e)} \tag{2.2}$$

From this theorem, using the CPT all probabilities related to the problem modeled can be calculated.

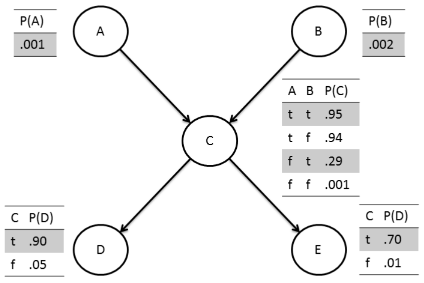

Figure 2.4: A Bayesian Network where nodes only depend upon their parentss.

2.5 Reinforcement Learning

When thinking about the nature in which humans learn, interaction with our environment is the main form for the first years. Using all the sensory data acquired, humans learn what actions are good or bad for them. This classification comes from the natural "reinforcement" of stimuli like pain, distaste, etc. From here future behaviors start to appear. In the field of AI, reinforcement learning raised as a way to let the agents in question know if the actions taken are "good" for them. Since not always is it possible to an agent, or even to a human to classify an action as good or bad, some kind of external reinforcement is needed. From this main idea comes the name and the way it works.

Yet a central and novel idea to reinforcement learning is **temporal difference** or TD learning [20]. Which is a combination of Monte Carlo and Dynamic programming ideas. From the first, TD learning burrows the ability to learn directly from raw experience without a model of the environment. Nevertheless, like dynamic programming, it update estimates based in part on other previously learned estimates, without waiting for a final outcome.

There are three possible agent designs when working with reinforcement learning [18]:

- An **utility-based agent** learns a utility function on states and uses it to select actions that maximize the expected outcome utility.
- A **Q-learning** agent learns an **action-utility function**, or **Q-function**, giving the expected utility of taking a given action in a given state.
- A **reflex agent** learns a policy that maps directly from states to actions.

An utility-based agent must also have a model of the environment in order to make decisions, because it must know the states to which its actions will lead. A Q-learning agent, on the other hand, can compare the expected utilities for its available choices without needing to know their outcomes, so it does not need a model of the environment. On the other hand, because they do not know where their actions lead, Q-learning agents cannot look ahead.

Q-learning is an alternative TD method which learns an action-utility representation instead of learning utilities. Although this may seem like any other way of sorting utility information, Q-learning has a very important property: it is a **model-free** method [18]. In other words, it is an off-policy TD algorithm. The way to calculate the *Q-values* is the following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s) + \gamma \max Q(s',a') - Q(s,a)\right] \tag{2.3}$$

Where:

Q(s,a) is the Q-value for the state-action pair (s,a).

a is the action the agent takes.

s is the state the agent will be by taking action a.

α is the learning rate, which goes from 0 to 1. At 0, no learning process happens. At 1, the past learning is not taken into account.

R(s) is the reward for reaching state *s*.

γ is the discount factor. This goes from 0 to 1 and penalizes the agent for the possible action-state pairs he can take in the next step if he choses now (s,a).

Q(s',a') is the Q-value for the state-action pair (s',a') that can be achieved by the agent if (s,a) is selected in the current step.

In practice, the Q-value function can be seen as a matrix where rows represent the states and columns the actions. While *R(s)* would have the same structure but the value of $R_{s,a}$ would correspond to the reward of the state-action pair (s,a).

2.6 Related Work

The problem of decision making has appeared constantly in the field of artificial intelligence and different approaches have been taken. In the RoboCup environment this is a problem that each team must address in order to play each game. Although this competition was started in 1996, there has been extensive research in the use of AI techniques such as graphs, Bayesian networks and reinforcement learning in this domain. In the following subsections work that has been done in these areas and that are relevant to this thesis is presented.

Thanks to the Robocup Soccer Simulation League and all the Soccer Leagues, a lot of research has been done in the area of strategy, cooperation and passes.

Wright Eagle, one of the best simulation teams in the world uses information from one "step" in the future in order to let the player position themselves much better while playing in the game [1]. This way of playing lets the team choose the best possible actions in the near future and lets them get ready for it. Yet this is not perfect due to the environment of the game being so dynamic. A similar but different approach is presented in this thesis. Even though prediction in the team is used, only one step ahead is checked in order to avoid getting stuck on a suboptimal decision for the long run. This will be further explained on chapter 3.

Several learning algorithms for passing and coordination have been proposed and used in these years. From Q-learning [8], [23], [9], [15] to Fuzzy logic [5]. Or just using Multiagent planning for solving the problem since the soccer game is nothing more than a multiagent system [10]. As stated by Chen, in a dynamic environment, each agent must be able to learn rapidly and certainly. Neural Networks can also be used to learn not only low level skills like passes, but even to learn high level tactics [14]. Even combinations of different techniques are useful, for example the use of Q-learning with fuzzy logic [21]. From here, it would not be so groundbreaking to combine techniques such as Q-learning with Bayesian Networks.

Due to nature of the environment, reinforcement learning and specially Q-learning becomes a very good candidate for use in RoboCup. Also, Q-learning provides the advantage if being model free. Not only can the agents use reinforcement learning from predefined functions, but they also could be taught by the coach. This would allow the agents to even change the models they use to adapt better to the game.

In the case of fuzzy logic, it is not an odd choice since there are always many factors to consider and not always they are exclusive of one another. From the known and expected positions of the 9 allies on the field (not counting the goal keeper) and 11 enemies, to the decision of whether the agent is too close or far away from the goal or its teammates. Fuzzy logic could be used to take all the decisions of the agents. Nevertheless, the use of this uncertainty could be managed instead with Bayesian Networks for the very same reason. Not only that, but it is easier to use a Bayesian Network than a large number of rules and membership functions for each one.

Setting the learning part aside, the soccer simulation game is a problem where two multiagent systems compete against each other. Due to this, it is possible to use a set of high level setplays to confront a specific scenario. Quintana proposed the use of adaptation by teamwork to certain specific game situations by having previously learned actions to take in the game [15]. Altough reinforcement learning was used to achieve this, it is possible to hardcode these and use simpler decision algorithms for the team to react to the game.

Not only is the online learning important, but also the offline. Being able to learn the formations of opponents lets the players obtain valuable information about the plays the opponents might take. To solve this problem several approaches have been taken, from neural networks [11], as well as k-means, naive Bayes, support vector machines [6].

To solve some of the problems at hand, what is proposed is to use the information stored in the game logs. Using Machine Learning techniques it is possible to obtain relevant data [11]. The advantage of game logs is that they contain information about the whole state of the world. Contrasting the information a single agent can possess, game logs can give the agents an extra boost in the learning rate by letting them access information that is not normally at hand for them.

Game logs can be used for more than analyzing the opponents [19]. For example, they could be used by the coach to analyze how well the team is playing and fairing. This can let the players learn more by using information from the whole game, as opposed to the limited information each agent possesses in game. Using the information from the logs, supervised learning can be used to obtain useful information. As another approach, reinforcement learning could be used from the logs. It may seem as an odd choice, but the added information can give the agents a much better learning than when learning by themselves.

As to how the agents can take the decisions at hand when playing, several approaches can be taken. From the basic idea of having multiple choices, a decision tree could be used [18]. A more refined way of doing this would be to use a Petri Net for decision making [25]. Or even one more focused on the problem simulating a decision graph since it is a generalization of a decision tree [13].

The use of graphs is not out of hand because of the nature of the decision process. When an agent has the ball, he must decide what action should be taken, whether it is a pass, a drible, or a shot there are many positions of interest in the field. These could be other agents, goals or even places on the field. Those could be considered as nodes of a graph where the edges are the paths that link them. To obtain the topology of the graph, supervised learning could be used to create edge-weighted graphs [22].

A way to create the graph and solve it to obtain its minimum spanning tree is proposed in this thesis, by using a Bayesian Network (BN) to represent the weights an agent would give to a designed path according to his beliefs. Although a BN would not give a physical topology, the weighted the graph can be solved for its MST to obtain a subgraph that represents a topology of paths between places of the field. This would let the agent not only make a decision of those available to him, but would also let other agents check the decision the ally took in order to anticipate the action for them to take.

Although a BN could very well solve problems in game, a static one could be detrimental in the long run. To address this issue, machine learning can be used to let the Network learn and adapt in the game or for a specific team. Kaitwanidvilai proposed using q-learning as an active method to let BN learn and adapt to solve in a better way the proble at hand [7]. This approach is taken in this thesis in order to adapt the BN as the game progresses and to let the players play better against their opponents.

2.7 Summary

In this chapter, background for the environment in which this thesis is being done is presented. Also, theory of all the methods that comprise the proposed solution is explained: graphs, Bayesian networks and Q-learning. Work related to the proposed method is shown at the end of this chapter. With all the theory presented in here, the reader should have no trouble understanding the terminology used in the following chapter, in which the proposed solution to the problem at hand is presented.

Chapter 3

Proposed Solution: Graphs with Bayesian Networks and Q-learning

During a soccer game there are many actions that can be taken when a player has the ball. He can pass the ball, run with it, shoot to the goal, make a pass into space, etc. Or simply wait to perform the best play he considers at the moment. To decide what action he wants to perform, a lot of information has to be taken into consideration. To make a pass, the position of my teammates must be known. To make a pass into space, positions on the field must also be taken into account. This problem can be seen as finding routes for the ball to go from one place on the field to another. Hence, the problem can be viewed as one of a graph with many nodes all interconnected. This chapter presents the methodology used to transform the problem of decision making for the player, in this case agents, to one that can be solved by a graph.

Section 3.1 shows an example of how the method proposed in this thesis works. Section 3.2, presents how the problem can be transformed to a graph and solved. Section 3.3, describes the inclusion of a Bayesian Network to weight the edges of the graph. Section 3.4, discuss how reinforcement learning can be used to adapt the playing style of the agents to have the best outcome in a match.

3.1 A Glimpse of the Proposed Solution

The method proposed in this thesis is comprised of two parts, the in-game process and the Bayesian Network training. The in-game process that the agents do is the decision making they use in order to play the game. All this decisions are sent to the trainer via the server and stored for future use. Once the game, or several games are finished, the trainer uses the data from the agents to train the BN using Q-learning. Once this is finished, the new network is sent to the agents. This process is shown on Figure 3.1

To give the reader a better explanation of how the method works, an example will be presented. Figure 3.2 presents the steps to be followed when using the method. First, all the possible nodes are selected: players and spaces of the field. Then the

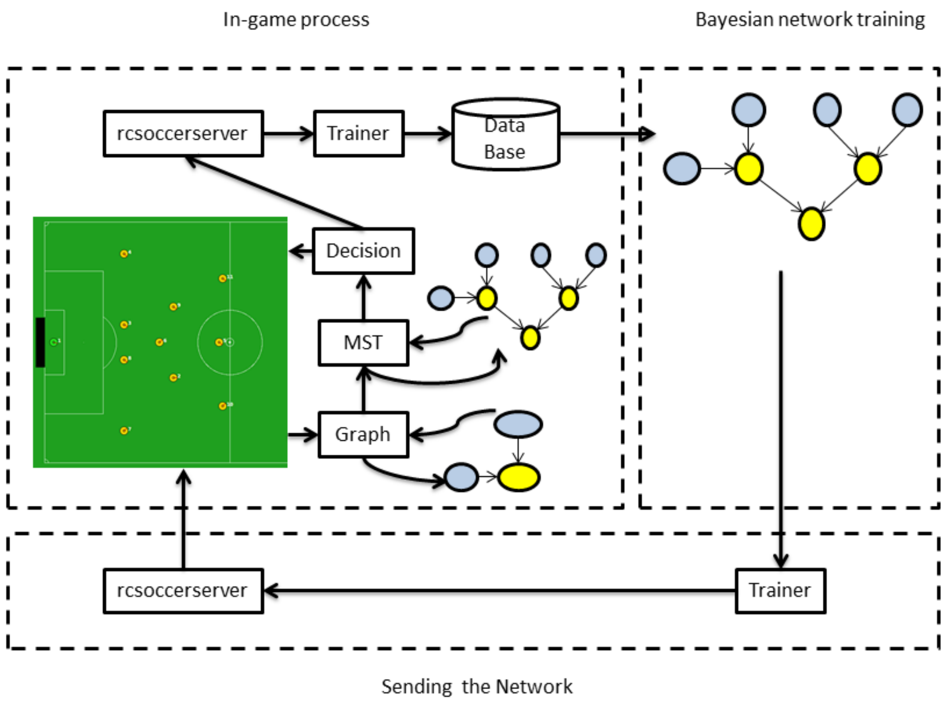

Figure 3.1: Diagram of the process where two main parts can be seen: the in-game process, where all the decisions are made and the Bayesian Network training, where the trainer uses the data from the decisions to train the network.

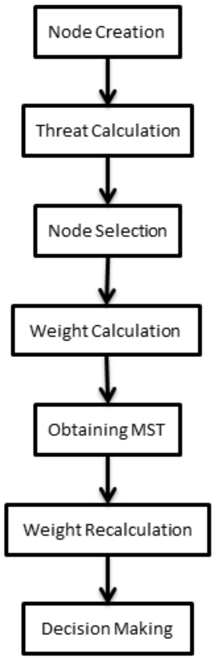

Figure 3.2: Block diagram of the proposed method.

threat of each node, using its coordinates is obtained. From here, those nodes that represent a high threat are not considered and the remaining are used for the graph. Now it is possible to obtain the weights of all the edges in the graph. The edges are all connections from each node to every other node. To reduce the problem, the MST is obtained and the weights are recalculated. Why this is done will be explained latter. Finally, from the options available in the MST, the best is selected to execute.

Figure 3.3 presents a situation that the agents would have to solve during the game. To solve this by hand would take a lot of effort and would take away the purpose of easily showing how the method works. Because of this, only a small portion of the field, where the ball is will be considered. In Figure 3.4 the situation to be solved is presented where an attacker (a team member with the ball) and a possible receiver (an agent that receives a pass). It is to be noted that, since this is just to exemplify how the method works, only two agents against one enemy will be analyzed. Also, only the six sections of the field shown on the Figure will be analyzed.

On the next sections, the mathematical model used to calculate the threat and weights is presented. For now, only the obtained values, shown on Table 3.1 will be presented. From here, those nodes with values equal to or lower than 0 are eliminated from the graph, obtaining the one shown at Figure 3.6. The next step is to calculate the weights of the remaining edges. In order to simplify the calculation for the reader the weight of the edge will be calculated with the average of the threat of the nodes connected by the edge. The actual model used in the method is presented on following

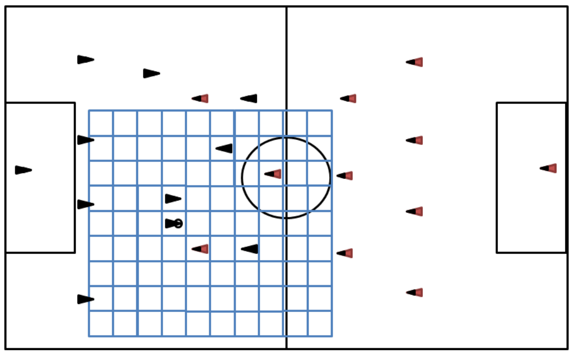

Figure 3.3: Situation to be solved by the agents in the game.

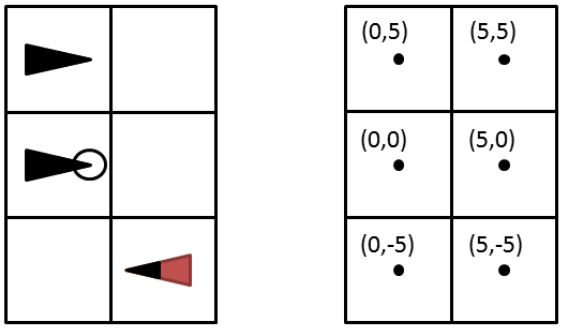

Figure 3.4: Situation to be solved by the proposed method.

sections.

Now that these initial weights are obtained, the length of the edges is taken into account to obtain a weight that favors those edges with small distance between the nodes. From here, the MST can be obtained. This is shown on Figure 3.7. From here, the action to take would be a pass from agent B to agent A.

H($placeOnField$)	value
H(A)	0.75
H(B)	1
H(1)	0
H(2)	0.54
H(3)	0.07
H(E)	-0.94

Table 3.1: Threat values on the six zones to analyze on the field.

But what if instead of using those values for the edges, a Bayesian Network was to be used in conjunction to them? For this particular example lets assume that the BN gave the values 0.1, 0.2 and 0.7 for the actions $Wait$, $Pass$ and $PassIntoSpace$ respectively. It is to note that there is no $Wait$ action in the graph, this is the action

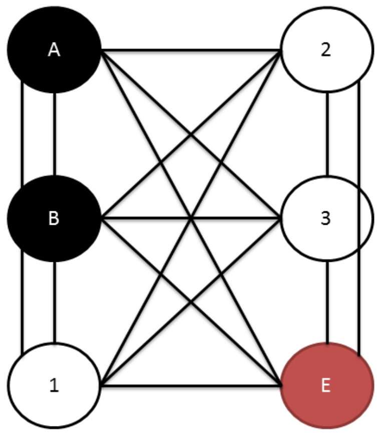

Figure 3.5: Graph of the situation on Figure 3.4.

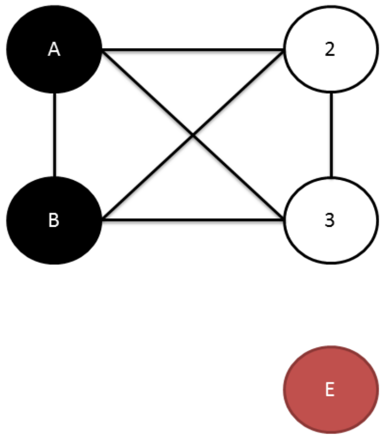

Figure 3.6: Graph after zones with high threat values are eliminated.

Node1	Node2	edgeType	W_0	$W_{distance}$	W
A	B	Pass	0.875	1	0.875
A	2	PassIntoSpace	0.645	1	0.645
A	3	PassIntoSpace	0.82	0.707	0.58
B	2	PassIntoSpace	0.77	0.707	0.54
B	3	PassIntoSpace	0.535	1	0.535
2	3	Strategy	0.305	1	0.305

Table 3.2: Weights of the different edges in the graph.

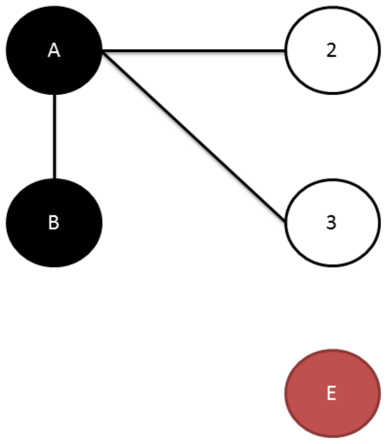

Figure 3.7: MST of the example.

of not choosing an option from those available. Also, as can be seen on 3.2, there is a *Strategy* edge type which will not be addressed on this work but is defined in the next section. In order to calculate the new weights, the probabilities given by the BN are used as follows:

$$W_{BN}(E_{type} = z) = P(z) * W \tag{3.1}$$

In Table 3.3 the weights of the edges are presented. As can be seen on Figure 3.8, the actions to take for player B are different than last time. This time the only possible action to take is the *PassIntoSpace* from B to 2. But before the action is taken the weight of the selected action is compared to P($Wait$). In this case, W(B,2) > P($Wait$), so the action the agent would take is to make an action into space to node 2.

Nevertheless, how would the decision be affected if before making a decision the MST was subject to another network? One more time a Bayesian Network is used to change the weights of the edges. With the difference that this time only the ones in the MST are taken into account. Lets assume that the BN gave the values 0.5, 0.3 and 0.2 for the actions $Wait$, $Pass$ and *PassIntoSpace* respectively. Table 3.4 shows the results for this case. This time around, W(B,2) = 0.0378 and P($Wait$—$Wait_{t-1}$)=0.1x0.5=0.05, hence W(B,2) < P($Wait$), the action selected by the agent would be to wait for the team to reposition themselves and have a better opportunity for a play.

Finally, what is the difference between using the MST or the whole graph? To show the difference, the example will be solved using all the possible decisions. The weights of all the edges are shown on Table 3.5. From here, it can be seen that the selected action would be a pass from B to A. Then why use the result from the MST if those values are lower? As mentioned in the first chapter, it is believed that by doing this, suboptimal decisions, those decisions that appear to be the best at the moment

Node1	Node2	edgeType	W_0	$W_{distance}$	W	W_{BN}
A	B	Pass	0.875	1	0.875	0.175
A	2	PassIntoSpace	0.645	1	0.645	0.4515
A	3	PassIntoSpace	0.82	0.707	0.58	0.406
B	2	PassIntoSpace	0.77	0.707	0.54	0.378
B	3	PassIntoSpace	0.535	1	0.535	0.3745
2	3	Strategy	0.305	1	0.305	-

Table 3.3: Weights of the different edges in the graph using a BN.

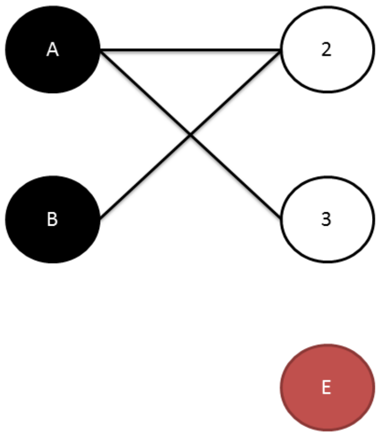

Figure 3.8: MST of the example when using a BN.

Node1	Node2	edgeType	W_0	$W_{distance}$	W	W_{BN}	W_{BN2}
A	B	Pass	0.875	1	0.875	0.175	-
A	2	PassIntoSpace	0.645	1	0.645	0.4515	0.04515
A	3	PassIntoSpace	0.82	0.707	0.58	0.406	0.0406
B	2	PassIntoSpace	0.77	0.707	0.54	0.378	0.0378
B	3	PassIntoSpace	0.535	1	0.535	0.3745	-
2	3	Strategy	0.305	1	0.305	-	-

Table 3.4: Weights of the different edges in the graph using a second BN on the MST alone.

but lead to plays that are not as good as other possible plays, in the long run are avoided. Maybe the pass from B to A seems like the best decision, but a pass into space to 2 may put the team in a position where the next action is better than the ones that the pass to A can lead to. It is believed this can happen by drawing a comparison to a local search.

Node1	Node2	edgeType	W_0	$W_{distance}$	W	W_{BN}	W_{BN2}	W_{total}
A	B	Pass	0.875	1	0.875	0.175	-	0.0525
A	2	PassIntoSpace	0.645	1	0.645	0.4515	0.04515	0.04515
A	3	PassIntoSpace	0.82	0.707	0.58	0.406	0.0406	0.0406
B	2	PassIntoSpace	0.77	0.707	0.54	0.378	0.0378	0.0378
B	3	PassIntoSpace	0.535	1	0.535	0.3745	-	0.03745
2	3	Strategy	0.305	1	0.305	-	-	-

Table 3.5: Weights of the different edges in the graph using a second BN on the MST alone and without the use of the MST. Although the weights of some edges is the same, now all but the strategy nodes appear.

Now that a brief example of what is proposed has been shown, the method will be formally explained.

3.2 Graph Construction

As mentioned before, there is certain information the attacker agents needs to consider in order to make a decision for passing the ball. What the attacker needs to know is:

Visible teammates positions. Which allies can be seen and where are they positioned. This is very important for passing the ball. If the attacker is not sure where the other players are on the field, a pass will most probably fail.

Non-visible teammates positions. Not only is it important to know where the teammates are, but also *where should they be*. In knowing these, it is possible to make more risky plays such as making a pass into space.

Enemies positions. The attacker may know where all the team is located, but if the enemies are not considered, the passes could be intercepted or even directed towards the contrary team.

Field positions. If only *simple* passes are considered, all the agent needs to know are his teammates positions and the enemies. Yet if more elaborate plays, such as a pass into space are to be done, then information about certain places of the field is essential.

With all that information is possible for the agent to make a rational decision for an action to take. In this thesis only three actions will be taken into account:

Pass. Passing the ball to the position of the selected teammate. How this is done can be seen on Figure 3.9

Pass into space. Making a pass to a place in the field where there is no player right now. Nevertheless, an ally can go to that position to get the ball when it arrives in place. How this is done can be seen on Figure 3.10

Wait. If no action seems rational at that moment, the attacker can wait in his position to re-evaluate and let his allies get into a better position.

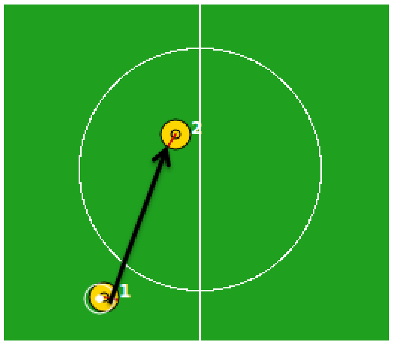

Figure 3.9: In a pass, player number 1 kicks the ball to player's 2 position.

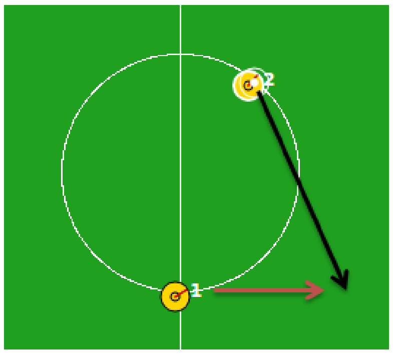

Figure 3.10: In a pass into space, player number 2 will kick the ball to a position to where player 1 can run to get the ball.

From this, it can be seen that the positions in the field where the attacker may want the ball to send to are either an ally or an empty position in the field. So, a graph using these positions as nodes is to be created. On this graph, all the nodes are to be interconnected having weights affected by several conditions of the field, creating the graph:

$$G = (V_i, E_i, W_i) \tag{3.2}$$

As mentioned before, there are two types of positions to be used for the graph: teammates positions and empty places on the field.

From the nodes that represent allies, only the ones which the attacker knows their position are selected. This is for two simple reasons: to make a pass, the agent needs to know exactly the position of the destination ally; to make a pass into space, he needs to know that an ally is highly likely to be near the position of the pass.

In the case of the nodes that represent empty places on the field, these are selected based on the position of the ball. To make precise passes on RoboCup 2D, the distance must not be higher that $30m$, since it is possible that even kicking as hard as the attacker can, the ball will not reach the destination. Because of this, points in the field that are in the range $[x_{ball} - 15, x_{ball} + 30]$ and $[y_{ball} - 30, y_{ball} + 30]$ are selected. The x range is smaller backwards for keeping the team from playing too conservative. The area considered for the empty places is divided into regions of at least $5m$ per side. The size of these is the same in a direction but not necessarily equal in both. The center of each region is taken as the position to be added to the graph. Only 10 of those positions will be added into it.

For instance, lets assume that $ballPosition = (0, 15)$. In this case, the range will be: x: $[-15, 30]$ and y: $[-15, 34]$. As to why the y range goes to the one shown and not to $[-15, 45]$, let us remember the dimensions of the soccer field, which goes from x: $[-54, 54]$ and y: $[-34, 34]$. In the x direction there are clearly 9 regions $((30-(-15))/5)$ of size $5m$, but in y, there could be 10 regions, with 9 of size $5m$ and one of $4m$. Or, according to what we said before that the region must be at least $5m$ long, 9 regions of size $5.44m$. From here, we have a total of 81 regions or empty places of the field. How to select the 10 places to add to the graph will be discussed later.

In the case of the edges, one connecting two *player nodes* represent a possible pass. An edge connecting a player and an empty place on the field represents a pass into space. While an adge connecting two empty places on the field can be considered as more strategic pair of nodes. This strategic nodes are empty places on the field to which two agents can run to and then make a pass, or the player with the ball can run to one and make a pass into space to the other node. Hence why those are a strategic pair, because strategies can be planed to be carried out in those nodes. Which leaves three types of edges: *pass*, *pass into space* and *strategic*. Since this thesis focus are the passes and passes into space, only the first two will be properly addressed.

To calculate the weights of the edges is not as straight forward as the inclusion of the nodes themselves. In order to do this the attacker must consider the positions of the allies he can see, the positions of the ones he can not see and the positions of the enemies.

3.2.1 Calculating the Weights

As mentioned before, to calculate the weight of the nodes, the positions of both the allies that the attacker can and can not see must be taken into account. To make calculations easier, a Gaussian model is proposed in order to evaluate the impact of any ally into any point of the field. The following distribution is used:

$$f(x,y) = exp\left\{ - \left[\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2} \right] \right\} \tag{3.3}$$

Where:
x_0: Ally x coordinate
y_0: Ally y coordinate
σ_x: The distance an agent can affect the most
For the allies the attacker can see:
σ_y: The distance an agent can affect the most
For those he can not see:

$$\sigma_y = \frac{ylim_M - ylim_m}{4} \tag{3.4}$$

For σ, the attacker knows a priori that the team will try to stay close to the ball. The σ_x parameter says that those x *meters* near to an ally are the ones that can be most affected by him. In the case of $ylim$, it tells the agent the zone in which an ally most probably is. The team plays using a designed formation, and agents tend to respect that formation. From that formation and the behavior for following the ball, these parameter is set as proposed above. In the case that the attacker is seeing an ally, the distribution is more precise since the it is more concentrated where the ally actually is. From here two equation can be obtained:

For the allies the attacker can see, setting $\sigma_x = \sigma_y = 3$ because there will be little variance in their positions, although they may still move

$$F(x,y) = \sum_{j=0}^{m} exp\left\{ - \left[\frac{(x-x_j)^2 + (y-y_j)^2}{18} \right] \right\} \tag{3.5}$$

For the allies the attacker can not see but has a model, with $\sigma_x = 5$ because there can be more variance in their positions since the agent is assuming them, he is not sure of them

$$F'(x,y) = \sum_{i=0}^{9} exp\left\{ - \left[\frac{(x-x_i)^2}{50} + \frac{(y-y_i)^2}{2\sigma_y^2} \right] \right\} \tag{3.6}$$

In the case of the enemies, a similar model is used, but only those that the attacker can see are taken into account:

$$G(x,y) = \sum_{k=0}^{n} exp\left\{ - \left[\frac{(x-x_k)^2 + (y-y_k)^2}{18} \right] \right\} \tag{3.7}$$

Using the three models, the threat of any point in the field can be calculated by the following equation:

$$H(x,y) = \alpha F(x,y) + \beta F'(x,y) + \gamma G(x,y) \tag{3.8}$$

The factors α, β and γ only represent the weigh given to each individual function based on the observations. For example, an adequate value could be: α = -γ, since both represent functions from which the attacker knows exactly the positions of the players. While $\mid \alpha \mid > \mid \beta \mid$ could be a good value, since β represents those allies that the agent is guessing their positions.

Now that an equation to measure the threat of any point in the field is defined, a function to obtain the weight of an edge connecting two points, $P_i = (x_i, y_i)$ and $P_j = (x_j, y_j)$ can be proposed as the weight of the point that has the maximum evaluation in the straight line that connects both nodes:

$$W_0 = \max H(L) \tag{3.9}$$

$$L = \left\{ (x,y) | x \frac{y_i - y_j}{x_i - x_j} + y = \frac{x_j y_i - x_i y_j}{x_j - x_i} \right\} \tag{3.10}$$

But this function takes only into account any obstacle that may lay in the path from one node to another, and is not considering the physical distance between the nodes. To compensate for this, the actual equation for the weight of the edges is defined as:

$$W_{distance} = \frac{r_{safe}}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}} \tag{3.11}$$

$$W = W_0 W_{distance} E_{factor} \tag{3.12}$$

Where r_{safe} is the maximum distance at which a pass can be almost completely safe. In this case it could be set at $5m$. While E_{factor} refers to a factor determined by the type of edge. For instance, passes could have a higher value that passes into space in order to make more passes. Or vice versa, in order to have more passes into space.

Now that a way to determine the weight of the edges has been proposed, this function can be used to determine which of the regions of empty places on the field will be selected. The simplest way to do that is to select the ones with the lowest evaluation, that are within a given threshold:

$$V = \min[H(Region_i) < 0] \tag{3.13}$$

By extracting the selected *Region* and repeating, the nodes can be selected.

3.2.2 Solving the Graph

So far all the parts of the graph have been defined, but the structure of each variable on it may be not so clear. Table 3.6 presents the structure of V. It can be seen that it presents all the characteristics of each node: whether its an ally or an empty position, and the physical coordinates of that point on the field. The index simply gives a number to the node. Table 3.7 shows the structure of E. As with the nature of edges on graphs, one connects two nodes, so it is not surprising that the first two elements of E represent those two nodes. The last element of E represent the type of edge that connects both nodes. As was mentioned earlier, there can be *pass*, *pass into space* and *strategic* edges. The first one connects two allies nodes. The second one an ally node and an *empty space on the field* node. The last one stands for an edge that connects two of the last mentioned nodes. Finally, W, presented on Table 3.8 can be seen as an extension of E, or both can be viewed as one, where an element that expresses the cost of going from one node to another is presented.

To make the calculations of the equation 3.9, some parameters must be defined first. An example of those parameters will be presented based on the formation presented on Figure 3.11, taken into account agent 6. From here, the limits presented on Table 3.9 are proposed, so that the players respect play zones in the field.

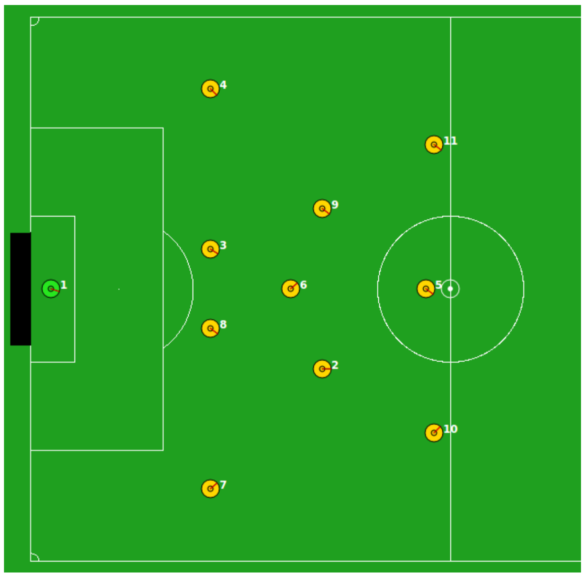

Figure 3.11: Formation: 4-3-3 that agents will be using in the games.

index	nodeType	xCoordinate	yCoordinate
0	MyPosition	-20	-5
1	Ally	-10	0
2	Ally	-5	-10
3	Ally	-2	-18
4	Ally	-3	-0
5	Position	-20	-25
6	Position	-20	15
7	Position	-15	-5

Table 3.6: Structure of the nodes in the graph.

node1	node2	edgeType
0	1	Pass
0	2	Pass
0	3	Pass
0	4	Pass
0	5	PassIntoSpace
0	6	PassIntoSpace
0	7	PassIntoSpace

Table 3.7: Structure of the edges in the graph.

node1	node2	edgeType	Weight
0	1	Pass	W_{01}
0	2	Pass	W_{02}
0	3	Pass	W_{03}
0	4	Pass	W_{04}
0	5	PassIntoSpace	W_{05}
0	6	PassIntoSpace	W_{06}
0	7	PassIntoSpace	W_{07}

Table 3.8: Structure of the weights in the graph. As can be seen, this structure is just an extension of E.

playerNumber	x_0	y_i	$ylim_M$	$ylim_m$
2	-2	18	$ylim_M$	$ylim_m$
3	-5	-10	$ylim_M$	$ylim_m$
4	-20	-25	$ylim_M$	$ylim_m$
5	-2	-18	$ylim_M$	$ylim_m$
6	-20	-25	$ylim_M$	$ylim_m$
7	-20	5	$ylim_M$	$ylim_m$
8	-10	0	$ylim_M$	$ylim_m$
9	-3	0	$ylim_M$	$ylim_m$
10	-20	-25	$ylim_M$	$ylim_m$
11	-5	10	$ylim_M$	$ylim_m$

Table 3.9: Different parameters of the agents shown in Figure 3.11

Algorithm 3.1 Action to take

```
kAllies ← number of visible allies
V ← My position
V ← Visible allies positions
Regions ← center of areas created by dividing the area of $[x_{ball} - 15, x_{ball} + 30]$ and
$[y_{ball} - 30, y_{ball} + 30]$ into rectangles of at least $5m$ by side
for $n = 1$ to 10 do
  V ← min [H($Regions_i$) < 0]
  kRegions++
end for
for $n = 1$ to $kAllies$ do
  for $m = 1$ to 10 do
    E ← (n,m)
    W ← ($V_n$,$V_m$)
  end for
end for
G = (V,E,W)
Kruskal(G)
Select best path from my node
```

Once the complete graph has been defined, it can be easily solved using widely known techniques. With Kruskal's algorithm, the minimum spanning tree (MST) of the graph can be obtained in order to obtain the optimal routes from each node. With these, the attacker can select from the options available to him in that moment, the one that represents the best action. Although it is not necessary to solve the whole MST for the agent to make a decision, it can function as a strategy planner for the whole team. That application of the MST will not be addressed in this thesis. The method presented in this section works as shown in algorithm 3.1.

So far, it has been presented how to take decisions based on the creation of a graph and solving the MST. But no proper way of making sure that the decision taken is the one that should be done.

3.3 Using a Bayesian Network for Calculating the Weight of the Edges

One way to decide the final action that the attacker will make is to use a Bayesian Network (BN). To construct it, the parameters that affect the final decision must be analyzed first.

Ball Position. Depending on the coordinates of the ball, actions may be more or less likely to succeed. If the ball is on a zone where the opponent has many agents in tight spaces, for example near the opponents' goal, it may be more risky to make a pass into space or waiting. Yet there may be more open areas on the field, where it could be more convenient to risk a pass into space.

Current State. Another important factor is the last action before the receiver received the ball. Is he receiving a pass? Has he just arrived at a position of a pass into space? has he been waiting for a better pass opportunity? Depending on that past action, it may be better to make one action than another. If the agent just received a pass into space, he may be on a good position to make another one or to shoot, than waiting for the opposing agents to reach him. If he has waited long enough, it may be too risky to keep waiting.

Score. Is the team loosing? Winning? in a tie? depending on the game situation, the agents may be able to take more risks, or need to keep the plays safe.

Game Time. If the game is just beginning, it will be very likely that the agents have full stamina. If it is near the end, most players will be exhausted, so the attackers can not risk making too many passes into space.

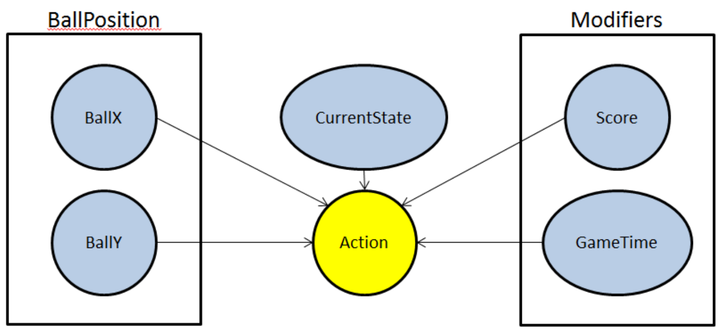

Figure 3.12: Initial Network proposed where the inputs affect the decision to be taken. The Score and GameTime are selected as modifiers since those variables affect directly the risk that can be taken by the agents.

Taking this into account, the basic model on Figure 3.12 is proposed. From here, the domain of these variables must be defined:

x Ball Position. {1, 2, 3} The field is divided into 3 regions across the x direction, from 1 (closest to own goal), to 3 (closest to opponent goal).

y Ball Position. {1, 2, 3} The field is divided into 3 regions across the y direction, from 1 (closest to the upper band), to 3 (closest to lower band). Combining the $xBallPosition$ and $yBallPosition$, a single Position variable, with 9 regions can be used.

Current State. {Wait, Pass, PassInToSpace} As stated before, the permitted actions will be those.

Score. {losing, tie, winning} As the name suggests.

Game Time (Half). {Starting, Half, Ending} Each phase is comprised of 1,000 game cycles.

Action. {Wait, Pass, PassInToSpace} What action should the attacker take.

Using the structure shown in Figure 3.12 would require to solve the problem by brute force. To avoid this, two new nodes that combine data from previous ones are proposed and are presented in Figures 3.13 and 3.14. These new nodes will be:

Logic Action. {Wait, Pass, PassInToSpace} This represents the action that the attacker considers is the best to take considering its **Current State** and the **Ball Position**.

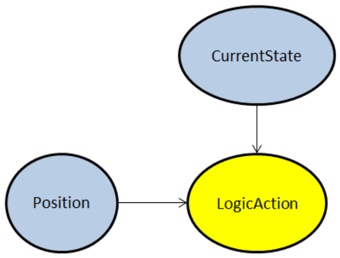

Figure 3.13: The Position and CurrentState nodes are combined to determine the Logic Action that the attacker could find best to do without taking into account the whole state of the game, just their position and current action.

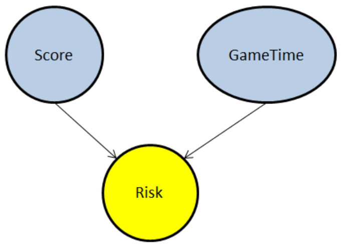

Figure 3.14: As mentioned, the score and game time give the agents a sense of how much risk con he take when making plays.

Risk. {Low, Normal, High} This variable represents how much risk can an agent take. It is affected by the **Game Time** and the **Score**.

From these, the Bayesian Network shown in Figure 3.15 is created to evaluate the decisions of the agents. From here, by using Bayes Theorem it is possible to calculate the outputs of the Network.

$$\mathbf{P}(z) = \alpha \sum_{x \epsilon X, y \epsilon Y} \mathbf{P}(z|LogicAction = x \wedge Risk = y) P(LogicAction = x \wedge Risk = y) \tag{3.14}$$

But from the structure of the BN, it is known that *LogicAction* and *Risk* are independent values.

$$\mathbf{P}(z) = \alpha \sum_{x \epsilon X, y \epsilon Y} \mathbf{P}(z|LogicAction = x \wedge Risk = y) P(LogicAction = x) P(Risk = y) \tag{3.15}$$

With all this information, the data needed for the attacker to calculate the action can be presented. First of all, defining the data from logic action, a matrix of 3x3x9

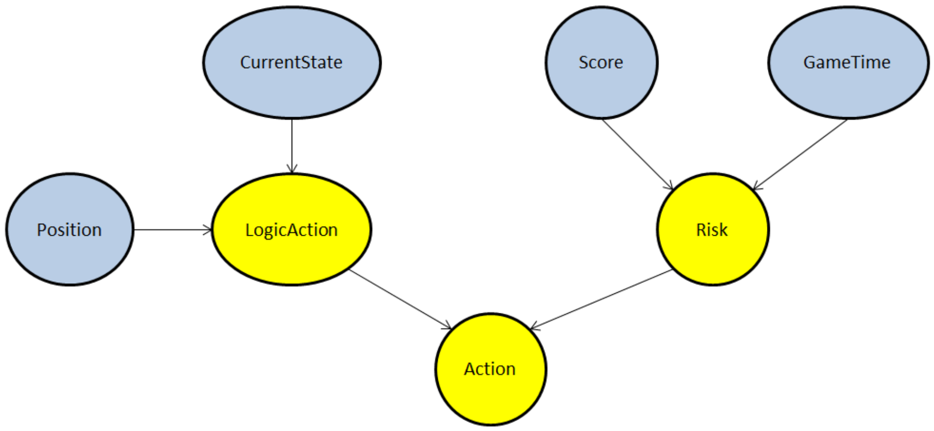

Figure 3.15: Bayesian Network proposed to address the problem of passes and passes into space. The *LogicAction* node is used to calculate the weights of the edges in the graph. While the output, *Action*, is used to recalculate the weights of the edges in the MST.

must be used. A probability distribution corresponding to only 1 region for the *Position* variable is shown in 3.10. For the *Risk* variable, a conservative play style is presented on 3.11, 3.12 and 3.13. Finally, probability relations for the final action to take need to be defined. In this case, a uniform distribution like the one for LogicAction can be used. The reason to use this distribution will be explained later.

CurrentState/LogicAction	Wait	Pass	PassIntoSpace
Wait	.3	.4	.3
Pass	.3	.4	.3
PassIntoSapce	.3	.4	.3

Table 3.10: LogicAction probability table for a specific Position on the field. Note that these values present a balanced, non risky play style

Score/Risk	Low	Normal	High
Losing	.2	.45	.35
Tie	.1	.35	.55
Winning	.05	.45	.5

Table 3.11: Risk probability table for GameTime=Starting

Now that the distributions have been set and the needed calculation was presented, the main question arises: *How is all this data going to be used in the graph for making decisions?*

The graph manages more logic actions, so it could be bound to the result of *LogicAction*. By giving an evidence from a position on the field, and adding the

Score/Risk	Low	Normal	High
Losing	.25	.5	.25
Tie	.15	.4	.45
Winning	.1	.5	.4

Table 3.12: Risk probability table for GameTime=Half

Score/Risk	Low	Normal	High
Losing	.3	.55	.15
Tie	.2	.45	.35
Winning	.15	.55	.3

Table 3.13: Risk probability table for GameTime=Ending

current state of the agent, *LogicAction* returns a probability distribution over the actions. With that information, the weight of the edges can be calculated. To do so, the weight obtained previously is used and the result of the BN is factored in to this:

$$W_{BN}(E_{type} = z) = P_{LogicAction}(z)W \tag{3.16}$$

Once the MST has been calculated, the best action to take would then be the *LogicAction* selected by 3.1. By adding this parameter to the BN, a probability distribution for *Action* is calculated. With those probabilities, is possible to recalculate the weights of the edges from the MST.

$$W_{MSTBN}(E_{type} = z) = P_{action}(z)W_{BN} \tag{3.17}$$

As to why that approach is taken, lies a fairly reasonable argument. If the agents try to find the best action in each situation they are in, the combination of best possible choices at those moment can lead the agents to get stuck on a suboptimal decision. If the MST is constructed from the best possible logical action and then, using a new policy to evaluate the possibilities at hand, the choice taken may not always be the optimal. This is because maybe a better choice using the new policy could lie in the edges that are not part of the MST calculated. Yet by allowing the method to make some errors, it is more likely that the agents will make the best plays in the long run. This idea is taken from search algorithms like *simulated annealing* that allow errors to avoid getting stuck in suboptimal solutions.

It is also possible to calculate from the beginning the weight of the edges using the result from *Action*:

$$W'_{BN}(E_{type} = z) = P_{Action}(z)W \tag{3.18}$$

This particular approach can lead to a peculiar problem already explained. Since this function would calculate the weights of the graph from the beginning, the agent

Algorithm 3.2 Action to take using the Bayesian Network

```
kAllies ← number of visible allies
V ← My position
V ← Visible allies positions
Regions ← center of areas created by dividing the area of $[x_{ball} - 15, x_{ball} + 30]$ and $[y_{ball} - 30, y_{ball} + 30]$ into rectangles of at least $5m$ by side
for $n = 1$ to 10 do
    V ← min [H($Regions_i$) < 0]
    kRegions++
end for
$P_{LogicAction}$ ← BN(xBallPosition, yBallPosition, CurrentState)
for $n = 1$ to $kAllies$ do
    for $m = 1$ to 10 do
        E ← (n,m)
        W ← ($V_n$,$V_m$,$E_{type}$,$P_{LogicAction}$(z=$E_{type}$))
    end for
end for
G = (V,E,W)
Kruskal(G)
$P_{Action}$ ← BN(LogicAction, Score, GameTime)
W ← W * $P_{Action=E_{type}}$
Select best path from my node
```

would always seek the best option. Drawing a parallel with local search, since both look for the best action from the current position, this could lead to getting stuck in a local maxima or a suboptimal decision for the problem at hand. Both approaches will be addressed in this thesis.

The method using a Bayesian Network does not change much from the original. The only difference is that before calculating the weights of the edges, the probability distribution of *LogicAction* must be calculated. Then, after the minimum spanning tree is obtained, the Network reevaluates the weights using the probability of *Action*. The new method can be seen on algorithm 3.2.

3.4 Guessing of the Pass into Space

Although it has already been explained how to select an action, it was not stated how can the teammate who is supposed to receive the pass know that. That task is actually very simple. To do that, instead of running the method as is, the agent substitutes his position for the position of the player with the ball. By doing that, the agent can determine what action the teammate is going to do. Even though the values obtained by each agent will not always be the same, they should be fairly similar. If the player knows that the ally is doing a *PassIntoSpace*, he can check if, according to the probabilistic teammate model, if he is the one expected to go for the ball. The method the agent is to run in this case is presented on algorithm 3.3.

The use of this, is precisely what leads to a $CurrentState = PassIntoSpace$. The agent, knowing that is going to receive a pass, will go to the destination. If he could get the ball, his current state will be *PassIntoSpace*. If he never arrived, then the state-action pair for the teammate would be: $(CurrentState, \overline{PassIntoSpace})$.

With the use of what has been presented so far is now possible for the agents to play as a team and not only taking advantage of opportunities for making passes.

A question that may have arised so far, because it has not been explained is: *When do the agents wait?*. Since there never is an edge type wait, technically, the players are never going to choose this action. So, how to implement it? To answer the question, first it is needed to define when and why should the agent wait.

If, from all the possible actions the agent has, all of them represent a very high threat to execute, the agent should not execute any of those actions. This may be because there are enemies blocking key areas for a *PassIntoSpace*, or allies very close to opposing players. Maybe enemies blocking the path for making a simple *Pass*. To define when should the agents wait, a threshold should be used in the following way:

$$\alpha[1 - P(Action = Wait|s)] < threshold \tag{3.19}$$

In here, α represents the weight of the selected action. If the probability of waiting is close to one, the value will be very small, very likely below the threshold and the agent will wait. If the probability is too small, it will have no impact on the selected action.

Algorithm 3.3 Algorithm to determine the selected action of the teammate with the ball

```
kAllies ← number of visible allies
V ← Position of ally with ball.
V ← Visible allies positions
Regions ← center of areas created by dividing the area of $[x_{ball} - 15, x_{ball} + 30]$ and
$[y_{ball} - 30, y_{ball} + 30]$ into rectangles of at least $5m$ by side
for $n = 1$ to 10 do
    V ← min [H($Regions_i$) < 0]
    kRegions++
end for
$P_{LogicAction}$ ← BN(xBallPosition, yBallPosition, CurrentState)
for $n = 1$ to $kAllies$ do
    for $m = 1$ to 10 do
        E ← (n,m)
        W ← ($V_n$,$V_m$,$E_{type}$,$P_{LogicAction}$(z=$E_{type}$))
    end for
end for
G = (V,E,W)
Kruskal(G)
$P_{Action}$ ← BN(LogicAction, Score, GameTime)
W ← W * $P_{Action=E_{type}}$
Select best path from the ball node.
```

3.5 Using Q-learning for updating the Bayesian Network

It might seem like an odd choice to combine Bayesian Networks(BN) with Q-learning. Nevertheless, both fit very well in the task at hand. A BN can easily handle the calculation of the weights in the graph taking into consideration the whole state of the game. While having a *coach* that can oversee the team and even give feedback to the agents, it can be seen as a reinforcement learning problem. Just as in a real soccer team, when a player makes good decisions, he gets rewarded. When the team as a whole make good plays, it goes up in rank and even can earn more money. In the case that the team keeps playing badly, it goes down in rank and even in the payment.

To give a reinforcement, first the *coach* needs to define how he is going to manage it. In this particular case, the $Q - value$ given to the players. This is defined as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s) + \gamma \max Q(s',a') - Q(s,a) \right] \tag{3.20}$$

Where:

s: State. For *LogicAction* it would be ⟨ Position, CurrentState ⟩. For *Action*: ⟨LogicAction, Risk ⟩.

a: Action, for both *LogicAction* and *Action*: ⟨Wait, Pass, PassIntoSpace ⟩.

R(s): Reward corresponding to doing action a in state s.

s': Next state. The state the agent would be after performing Action a.

a': Next action. The action the agent can make from state s'.

α: Learning rate. To what extent the new information will impact old information.

γ: Discount factor: how the Q-value is updated by taking into account the possible reward from the next state.

From here, a reward function needs to be defined. The one proposed in here is presented on Table 3.14 as a reward matrix.

s/a	Wait	Pass	PassIntoSpace	$\overline{Wait}$	$\overline{Pass}$	$\overline{PassIntoSpace}$
Wait	10	10	10	-20	-10	-15
Pass	10	15	20	-10	-5	-5
PassIntoSpace	10	15	30	-10	-5	-10

Table 3.14: Matrix for the Rewards from the pairs (state, action).

Nevertheless there is a problem in the definition. A same action can yield two outcomes: a completed action or an interrupted one. And in $R(s)$ these outcomes are split. To overcome this, the Q-value can be separated into two:

$$Q_{complete,ij} = Q_{ij}, i = \{1,2,3\}; j = \{1,2,3\} \tag{3.21}$$

$$Q_{incomplete,ik} = Q_{i(k+3)}, k = \{1,2,3\} \tag{3.22}$$

$$Q_{BN}(s,a) = \alpha Q_{complete} + \beta Q_{incomplete} \tag{3.23}$$

Where both factors, α and β can be set to 1. Taking into account how the Q-value matrix is structured, it can be seen that, if each row is normalized, a probability distribution for the possible actions given a state is obtained. In other words:

$$Q'_{BN,nm} = \frac{Q_{BN,nm}}{\sum Q_{BN,ni}} \tag{3.24}$$

$$P_Q(a|s) = Q'_{BN,nm} \tag{3.25}$$

From here, it is clear that from the Q-value is a new probability distribution for the Bayesian Network. But another question appears after: *How is the BN going to be updated using the normalized Q-value?* There are several ways to accomplish this task. To ensure a smooth transition between the model that the agents have and the one the *coach* gave through reinforcement is to make a pondered sum:

$$P'(a|s) = \alpha P(a|s) + \beta P_Q(a|s) \tag{3.26}$$

Where α = 1 - β.

It is possible then for the agents to receive reinforcement for their plays via the coach, in order to let them play better and better when facing an opposing team. For the purpose of this thesis, the learning will be done off-line between games against a similar team using the trainer or offline coach.

It is to be noted that what is obtained can use different inputs in order to calculate both *LogicAction* and *Action*. It is possible then to train both nodes in the BN in order for the agents to learn better decisions to make in game. Also, not necessarily the same reward matrix needs to be used. It can be accommodated as the programmer sees fit.

If on-line learning is to be used, the *coach* would be in charge of acquiring the data from all the plays in the game. Each 300 cycles, he could give the players the reinforcement they need in order to fair better against the opponents. As part of future work, the learning will be implemented on-line for a 2D Soccer Simulation team.

Algorithm 3.4 Algorithm to determine the action for the agent that has the ball and the agent to receive the *PassIntoSpace*

```
kAllies ← number of visible allies
V ← Position of ally with ball.
if agent has ball then
  V ← My Position
else
  V ← Position of ally with ball
end if
V ← Visible allies positions
Regions ← center of areas created by dividing the area of $[x_{ball} - 15, x_{ball} + 30]$ and $[y_{ball} - 30, y_{ball} + 30]$ into rectangles of at least $5m$ by side
for $n = 1$ to 10 do
  V ← min [H($Regions_i$) < 0]
  kRegions++
end for
$P_{LogicAction}$ ← BN(xBallPosition, yBallPosition, CurrentState)
for $n = 1$ to $kAllies$ do
  for $m = 1$ to 10 do
    E ← (n,m)
    W ← ($V_n$,$V_m$,$E_{type}$,$P_{LogicAction}$(z=$E_{type}$))
  end for
end for
G = (V,E,W)
Kruskal(G)
$P_{Action}$ ← BN(LogicAction, Score, GameTime)
W ← W * $P_{Action=E_{type}}$
if agent has ball then
  $W_{wait}$ ← Select weight of best path from my node
  Action ← Select best path from my node
else
  $W_{wait}$ ← Select weight of best path from the ball node
  Action ← Select best path from the ball node.
  if (destination node is connected to my supposed position) and(Action=$PassIntoSpace$) then
    Action ← GoTo(DestinationNode)
  else
    Action ← normal action // Action not determined by this algorithm
  end if
end if
if $W_{wait}$ $[1 - P(Action = Wait|s) < threshold]$ then
  Action ← Wait
end if
Execute Action
```

3.6 Summary

In this chapter it was presented how to use the minimum spanning tree (MST) of a graph, along with a Bayesian Network (BN) to calculate the weights of the edges to let the agents decide what action was best at that moment. For this particular case, the actions would be to wait, to pass the ball or to make a pass into space. It was also shown how reinforcement learning, in specific Q-learning, can be used to let the agents play better each time.

The basic course of action, as presented on algorithm 3.4, would be to construct the graph, send observations to the BN. Then calculate the weights of edges using the information from the Network. Solve the MST to recalculate the weights using the BN. Select the best action. In the following chapter, the experiments and results of the proposed method in this thesis will be discussed.

Chapter 4

Experiments and Results

This chapter is divided in two parts. The first part explains in detail how the experiments where conducted and prepared. The hypothesis and research questions are revisited in order to link them to the experiments needed. The second part, comprised of three sections, presents the results obtained from the different experiments. Also, these results are analyzed in order to check the validity of this research.

4.1 Experiments

As was stated initially, the purpose of this thesis is to develop a method that can solve the problem of decision making for the player with the ball using graphs and Bayesian Networks to calculate the weights of the edges in the graph. This is done in order to solve the problem of the pass into space in the RoboCup 2D environment. To do so, attackers must be able to know where to put the ball, while the receiver needs to know where is the pass aimed to.

To be able to complete the objectives, a form of experimentation must be implemented. In this particular case, soccer matches are the experiments to use. In order to have a benchmark to which all experiments can be compared, a base team will be used as opponent for all experiments.

The base team has the same architecture as the one used to check the method proposed in here. Just as this last one, uses a 4-3-3 formation where all players respect their positions to a certain degree. Agents go up when the ball is getting closer to the opposing teams' goal and go down in the opposite case. The main difference between the teams will be the decision system. While the base team will always use a graph in order to make decisions, the home team will use variations of the proposed method in order to check the different objectives. In this regard, the team will first use what is proposed in chapter 3 without the use of Q-learning. Games will then be played without using the MST for the recalculation of the weights. Finally, the team will play using the method with reinforcement learning.

In order to be able to run experiments in the SoccerSim, data between the agents must be stored in some way. Also, information of the statistics of games must be stored. Offline learning must be implemented between games and that information needs to be sent to agents also. To be able to do so, the best option is to use the trainer or offline coach in the simulation.

To store the data of the agents, the trainer can receive messages from them regarding the actual actions and processes they went through. This is done because to use Q-learning, not only is it necessary to know the action taken by the agents, but also the evidence in the node used to make the decisions. Furthermore due to the fact that learning is being used in two nodes, and the evidence and output of one of them is "only visible" to the agent taking the decision, each agent must let the coach know what evidence and choices were used.

To make this possible, the agents have access to the "say" command. Nevertheless, they are limited by the server to the length of the message that they can actually send. In the case of the agents, messages are limited to 10 characters at most, so the information sent must be codified without a case of information loss. The information that needs to be sent to the trainer, according to the Bayesian Network presented on chapter 3, is:

Current State. The action the agent is currently performing. These actions can be: wait, pass and pass into space. This information can be sent as W, P or S respectively.

Position. The position the agent is currently in. As explained before, the field is segmented into 9 regions in total. 3 regions in x and 3 in y. As this is a number from 1 to 9, or 0 to 8 in order to use a much simpler coding when programming, the number can be sent as is.

Logic Action. The action the agent would normally take without taking into account the state of the game as a whole, but only the state and position they currently are in. These actions can be: wait, pass and pass into space. This information can be sent as W, P or S respectively.

Risk. The risk the agent can currently take depending the game time and the score. The values this variable can take are: low, normal and high. This information can be sent as L, N or H respectively.

Action. The action the agent actually performed when taking into consideration the state of the game as a whole. These actions can be: wait, pass and pass into space. This information can be sent as W, P or S respectively.

In this way, the message sent to the trainer has a body of 5 elements. Thus, the message is long enough to contain all information and short enough to let the message have a head to not interfere with other type of messages. For example, a message of the decision taken by an agent could look like this: *(say "dec W7PLS").* Where *say* is the command used to send messages, *dec* would represent *"decision"*, the kind of information sent. The last part of the message is the body, the information to communicate to the trainer.

All the information obtained by the trainer during the game can be used in turn to perform the offline training of the Network. Once the data is processed and the new BN is ready to be used in the next game, the next problem arises: how to send all the data to the agents?

Unlike the agents, the trainer can send messages containing up to 128 characters. Yet the networks contain much more data than allowed to send. From the two where learning is used, one consists of 81 elements, while the second one has 27 (108 elements in total). This elements can take any value from 0 to 1. To overcome the limitation of the message space, a space transformation can be used in order to map each probability to a specific one element character. Taking advantage of the programming languages and avoiding encryption (since in the experiments the opposing team is not trying to codify the data of the message), ASCII code can be used to directly map probabilities into characters. Although these are limited, at least 101 values can be directly mapped (0 to 1 in increases of 0.01).

The problem that arises is that the SoccerSim does not let the messages contain just any characters, only the following are allowed: [$-0-9a-zA-Z$ ( ).+ $*$ / ?< > _]. To overcome this limitation, a reduced ASCII code can be used. This reduced code is presented on Table 4.1 and can be easily used using the following conversion:

$$ASCII = A + C \tag{4.1}$$

$$A = floor(61P) \tag{4.2}$$

Where: ASCII is the equivalent value in *ASCII* code, P is the probability from the BN and C is a constant dependant of the value of A.

$$C(A) = \begin{cases} 48 & \text{if A} < 10 \\ 55 & \text{if } 10 \leq \text{A} < 36 \\ 61 & \text{if } 36 \leq \text{A} \end{cases}$$

In the case of the agents, to convert the message to probability, the following conversion can be used:

$$P = \frac{ASCII - C'}{61} \tag{4.3}$$

Where:

$$C'(ASCII) = \begin{cases} 48 & \text{if ASCII} < 58 \\ 55 & \text{if } 58 \leq \text{ASCII} < 97 \\ 61 & \text{if } 97 \leq \text{A} \end{cases}$$

Although not all probability values have a direct transformation and may be subject to information loss, 59 values between 0 and 1 can be represented with this reduced ASCII code. With this transformation, the trainer can freely send the agents the BN to use after each game. It can even be used in game by the coach in order to use online learning during an actual match. It is to note that this particular case will not be addressed in this thesis, but is still a main subject for future work.

In order to get results from the experiments that give more accurate data, many games must be played. All of them can be of lower time than an actual match. For the purpose of this research, the games will be played as 4 minutes matches (2400 cycles or 240 seconds), as opposed to the actual 10 minutes matches. The matches will still be subdivided in two half's and will follow the normal rules of the game. All the experiments will consist of 500 games.

To get data from many matches, two approaches can be taken. Either a script can be used to restart the game each time it ends and let the trainer access the information of the past game, or the configuration of the SoccerSim can be adjusted for this. One of the advantages of the simulator is that it lets the user change parameters from the actual games. Two of the main things that can be changed for running the experiments of this thesis are:

nr_normal_halfs This variable determines how many "half's" the game will have. Normally set to 2, it can be set to any even number to simulate that many games are being played one after the other. This can be set for example to 200, to get data from 100 matches.

half_time As the name suggests, this variable determines the length of each "half" in the game. In this case, it will be set to 300 (since the variable is set in seconds)

In order to escape from problems of the actual data of the server, the players can store internally the number of goals in the "current" game. As well as the time that has elapsed from the current experiment. This is necessary to avoid the information from the server that will report the overall time of all the experiments, as well as the apparent score from the game.

Nevertheless, another problem may arise during the experiments. The stamina of the players will be substantially reduced during experiments. To overcome this, the trainer has access to a command that resets the stamina of all the agents so that each new experiment they are fully "rested."

character	value	ASCII code	character	value	ASCII code
0	0	48	V	31	86
1	1	49	W	32	87
2	2	50	X	33	88
3	3	51	Y	34	89
4	4	52	Z	35	90
5	5	53	a	36	97
6	6	54	b	37	98
7	7	55	c	38	99
8	8	56	d	39	100
9	9	57	e	40	101
A	10	65	f	41	102
B	11	66	g	42	103
C	12	67	h	43	104
D	13	68	i	44	105
E	14	69	j	45	106
F	15	70	k	46	107
G	16	71	l	47	108
H	17	72	m	48	109
I	18	73	n	49	110
J	19	74	o	50	111
K	20	75	p	51	112
L	21	76	q	52	113
M	22	77	r	53	114
N	23	78	s	54	115
O	24	79	t	55	116
P	25	80	u	56	117
Q	26	81	v	57	118
R	27	82	w	58	119
S	28	83	x	59	120
T	29	84	y	60	121
U	30	85	z	61	122

Table 4.1: Reduced ASCII code used to send the BN to the agents.

As presented in the methodology, several types of experiments must be realized. For all these, the teams will play against a base team. This team has the same structure as the home team, the one used to test the solution. The difference between the teams would be the decision making system. The base team using a simple graph, while the home team the method that this thesis presents.:

MST team: When not using Q-learning, games will be played against the base team.

no-MST team: Using the BN the weights of the edges will be calculated without the recalculation using the MST. Games will be played against the base team. This in order to prove that the method avoids suboptimal decisions in the long run.

Q-MST team Using Q-learning, games will be played against the base team.

For the first case, the values used to obtain the weights of the graphs will not be affected by the state of the world, nor the state of the agent. In other words, the way to calculate the weights of the edges as seen in chapter 3, will be as follows:

$$W = \frac{r_{safe}}{\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}} W_0 E_{factor} \tag{4.4}$$

$$W_0 = \max H(L) \tag{4.5}$$

$$L = \left\{ (x, y) | x \frac{y_i - y_j}{x_i - x_j} + y = \frac{x_j y_i - x_i y_j}{x_j - x_i} \right\} \tag{4.6}$$

$$H(x, y) = \alpha F(x, y) + \beta F'(x, y) + \gamma G(x, y) \tag{4.7}$$

$$F(x, y) = \sum exp \left\{ - \left[\frac{(x - x_j)^2 + (y - y_j)^2}{18} \right] \right\} \tag{4.8}$$

$$F'(x, y) = \sum exp \left\{ - \left[\frac{(x - x_i)^2}{50} + \frac{(y - y_i)^2}{2\sigma_y^2} \right] \right\} \tag{4.9}$$

$$G(x, y) = \sum exp \left\{ - \left[\frac{(x - x_k)^2 + (y - y_k)^2}{18} \right] \right\} \tag{4.10}$$

Where:
$x_{i,j}$: Ally x coordinate
$y_{i,j}$: Ally y coordinate
x_k: Opponents' x coordinate
y_k: Opponents' y coordinate
r_{safe}: The minimum distance considered as safe for a pass.

E_{factor}: A value dependent of the type of edge.

α, β and γ: Weigh given to each individual function based on the observations

$$\sigma_y = \frac{ylim_M - ylim_m}{4} \tag{4.11}$$

In the case of the no-MST, when choosing which decision to take, the Agent evaluates first all the possible edges using the *Logic Action* evaluation from the Network. But at the end, the action taken is dependent of the reevaluation of the selected possible actions in the MST. This lets the reevaluation consider only some of the possible actions, possibly relegating an action that may yield a much better weight. The purpose of this is to avoid making a method that always chooses the best choice at hand and may end up not considering a better sequence of actions.

In order to check the avoidance of suboptimal decisions, the result from the *Action* node will be used to calculate the weights of all the possible actions. From here, the best action will be selected.

For the case of the Q-MST, the team will be trained using Q-learning and then will play games against the same team of previous experiments.

To be able to evaluate how good the method is, some sort of measurement must be used. In order to check this, several data of the game related to actual soccer statistics will be analyzed. This analysis will see how those variables change along the experiments. The statistics to analyze are:

Ball possession. The percentage of the game time that each team had the ball. This value can be taken as the sum of all the total times that passes between an agent of a team touching the ball and the next time that an opponent gets is back.

Number of passes. How many passes each team made.

Completed passes. How many passes each team made and was successful.

Goals. How many goals each team could score in the game

Although many more soccer statistics can be measured , this are the most important for this study. It is to note that statistics for both teams will be presented. All this information can be gathered by the trainer by either using the information the server gives to him, or from the messages each agent sends when making a pass.

4.2 Use of MST vs. Base Team

For these experiments two teams were used: one that used a graph for taking the best decisions at a given time and one that used the method proposed in this thesis without using Q-learning. The first relevant result lies in the ball possession, which is

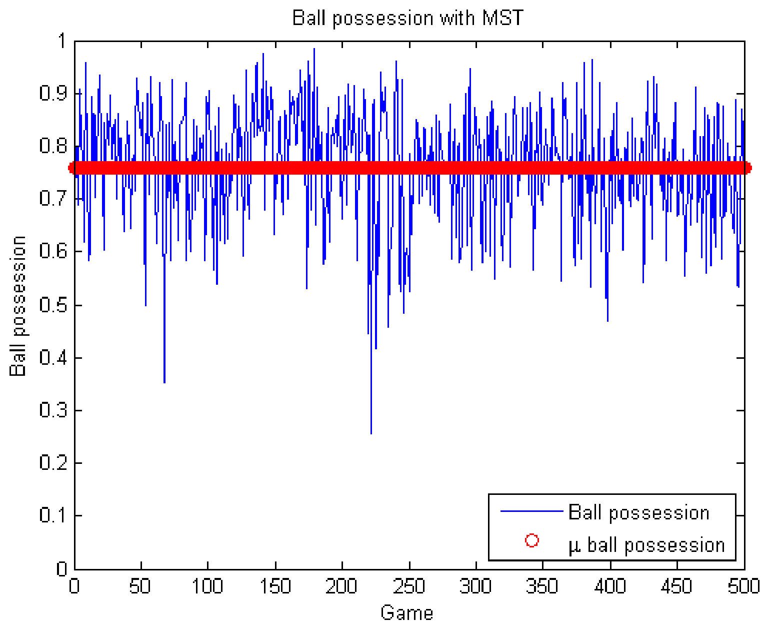

Figure 4.1: Ball possession percentage in each game when using the MST. It can be seen that this is almost always above the 50% of the time.

shown in Figure 4.1. From here it can be seen that the ball possession stood well above the 50% of the time for virtually all of the games.

The next result comes from the goal difference in the games. As shown in Figure 4.2, the majority of the games there was a goal difference above 1. Also, the goal ratio from the games had an average of 3.9 goals to 1. This marks a real improvement from using graphs for decision making in RoboCup2D.

Finally, the number of passes increases greatly when using the proposed method. As can be seen on Figures 4.3 and 4.4, the number of total passes is more than double than when using a normal graph. Also, the number of completed passes is almost 80% of the total passes with the proposed method while it stands at around 40% for the normal graph.

These results show that there is a great improvement in decision making when using Bayesian Networks for calculating the weights of a graph.

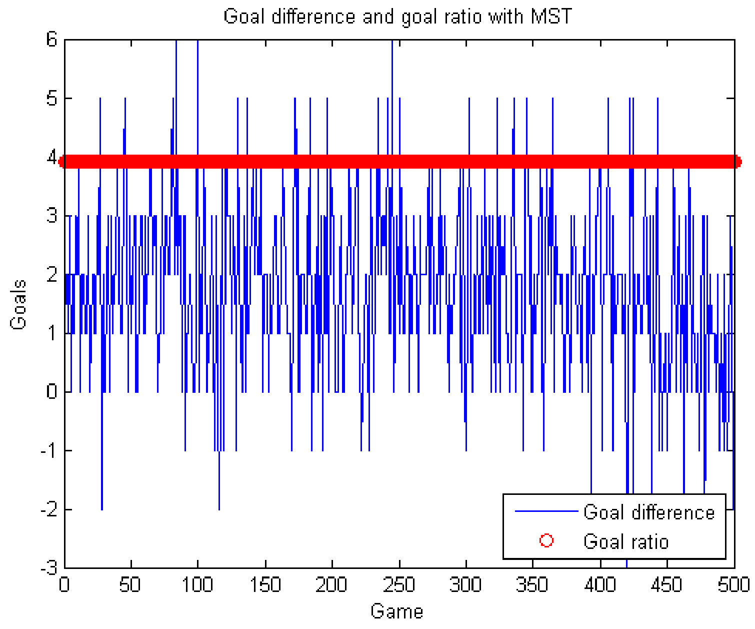

Figure 4.2: Goal difference for each of the games. It can be seen that in most of the games, the goal difference is well above 0. Also, the goal ratio for the games stood at 3.9:1.

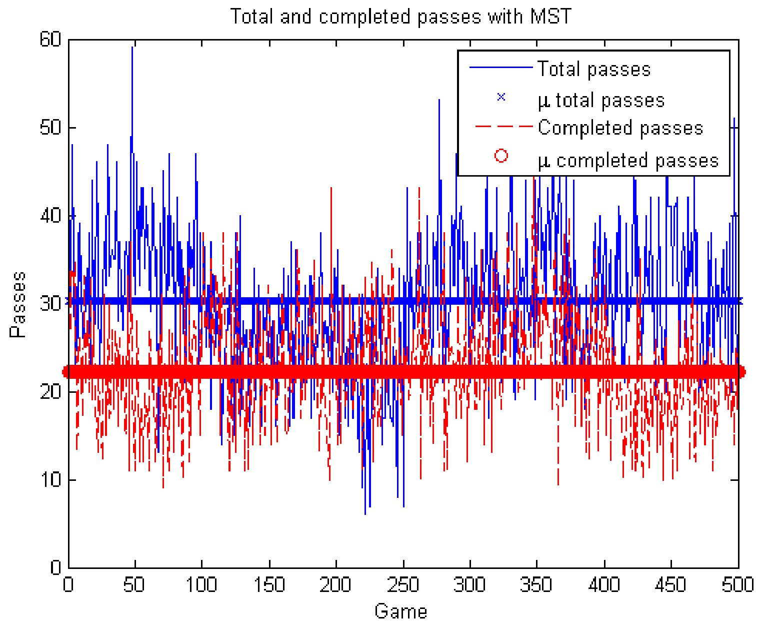

Figure 4.3: Total passes per game along with the number of completed passes. The total passes average was at 30 passes per game.

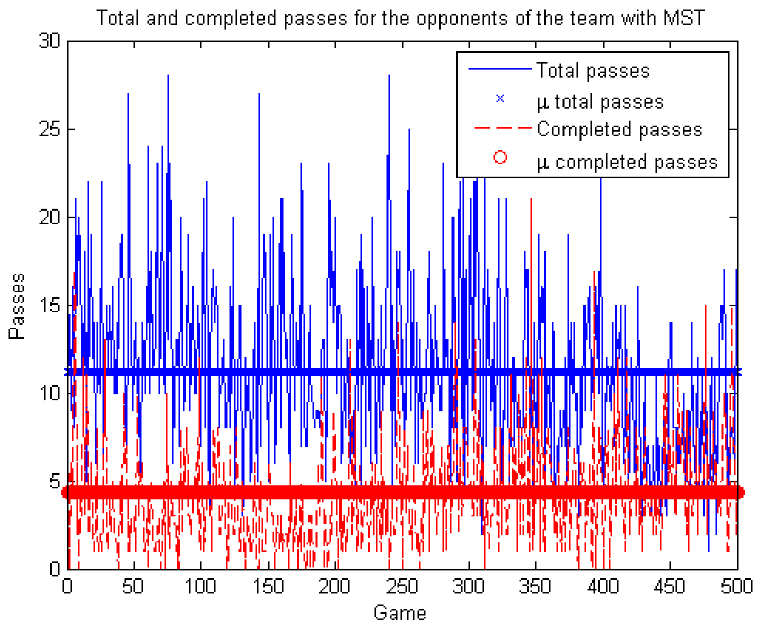

Figure 4.4: Total passes per game along with the number of completed passes. The total passes average was above 30 passes per game.

4.3 Use of no-MST vs. Base Team

As mentioned since chapter one, it is believed that this method is good for avoiding suboptimal decisions in the selected decisions. In order to prove this, the method was modified in order to use only the result from the Action node in the Bayesian Network. With the value from this node, the weights of the edges from the whole graph are calculated. Then the MST is generated and from here the best decision is taken. From the point of view of the player with the ball it may be a bit redundant to do this, since the best action from the MST is actually the best action from the graph. For a player that is deciding if a pass is to be done to him, the search in the MST is much easier than the one in the whole graph. The opposing team used only a graph for decision making.

In Figure 4.5 the ball possession from the games is presented. Comparing this result to the ones obtained in previous experiments, there seems to be little change. The possession time is above 50% of the time, centered around 70%. This was below the results from Figure 4.8, where the method proposed in this thesis was used.

As for the goal difference, it was above 0 for most of the games, centered around 0.5, as shown on Figure 4.6. The goal ratio from this team stood at only 1.5. Meaning that most games were won at a ratio of 1.5:1 goals.

Total and completed passes also took their toll when using only one result from the Bayesian Network. As shown on Figure 4.7 the average total passes was 14, with

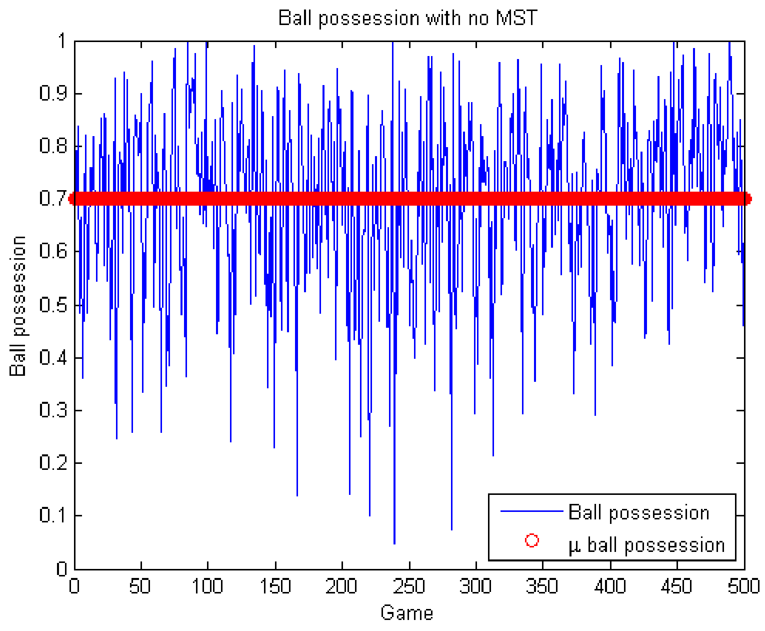

Figure 4.5: Ball possession percentage in each game when using no-MST.

	MST	no-MST
Games won	77.28%	44%
Games Lost	9.27%	25%
Ball Possession	76%	70%
Goal Ratio	3.9	1.5
Passes	30	14.5
Completed Passes	23	10

Table 4.2: Comparison of different stats between the team that uses the MST and the team that uses a no-MST.

an average of only 10 completed passes. Although the ratio of completed passes stands at 70%, only 10% below that of the proposed method, the actual quantity dropped to less than half that it was before.

From these results, it can be seen that using two Bayesian Networks for recalculating the weights of the graph actually increases the outcome of the game. More that doubling the results for passes and goals. As stated earlier it is believed that this is due to letting the method make ”mistakes,” in order to avoid suboptimal decisions in the long run. In Table 4.2, different statistics from the matches are presented, where it can be seen that the MST team vastly outmatches the no-MST team.

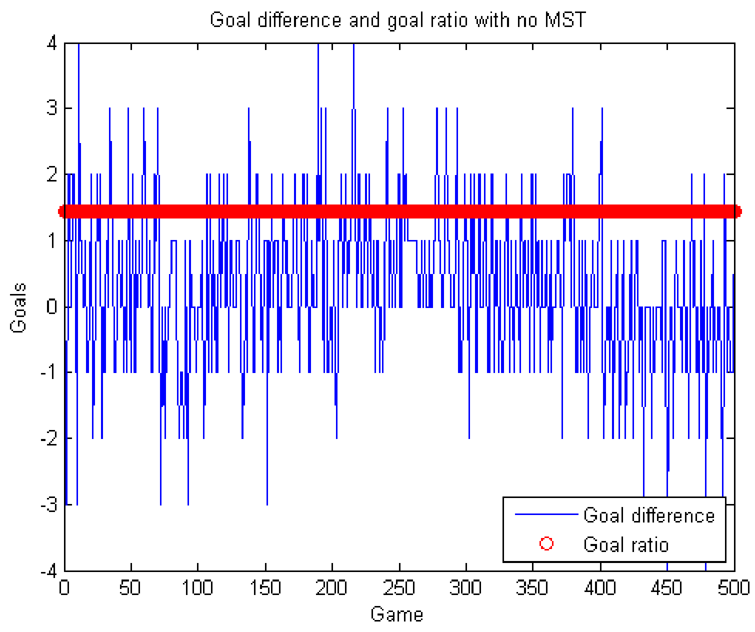

Figure 4.6: Goal difference for each of the games. It can be seen that in more than half of the games, the goal difference is above 0. The goal ratio stood at 1.5:1.

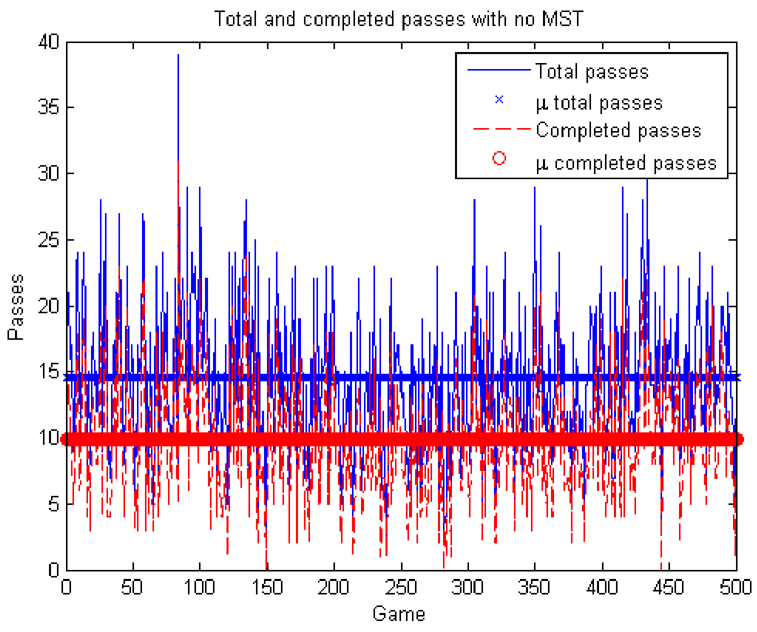

Figure 4.7: Total passes per game along with the number of completed passes. Using only the no-MST,the total passes average was below 15 per game

4.4 Use of Q-MST vs. Base Team

For these experiments, 500 games were played with the configuration of the MST team experiment. With these games Q-learning was implemented to train the team and play against it. After the team was trained, the resulting network showed that the agents learned to make mostly passes-into-space. As shown on Table 4.3 the values of the network are towards 1 for the pass-into-space decision while the wait decision got a value of 0. From here, the agents would practically make only passes-into-space.

CurrentState/LogicAction	Wait	Pass	PassIntoSpace
Wait	0	.1	.9
Pass	0	.1	.9
PassIntoSapce	0	.1	.9

Table 4.3: A part of the learned network after 500 games.

As shown on Figure 4.8, ball possession was above 50%. Nevertheless this time, in contrast to the experiment without RL, the possession percentage average stood at 79%. This marks an improvement when using Q-Learning.

In Figure 4.9 the goal difference for the games is shown. As can be seen, the goal ratio obtained was 6.8. This marks great improvement when using Q-learning, almost 80% than when not using it.

The total passes average did change when using Reinforcement Learning but only by a little. The mean was at 33 passes per game. The number of completed passes also rose by a small amount, to 26 passes, as can be seen on Figure 4.10.

From these experiments there seem to be a great improvement in the number of goals. Although other statistics also improved, this was only by little. In Table 4.4, different data from the matches are presented, where it can be seen that the one using Q-learning clearly dominates.

	Q-MST	MST	no-MST
Games won	81%	77.28%	44%
Games Lost	6%	9.27%	25%
Ball Possession	79%	76%	70%
Goal Ratio	6.8	3.9	1.5
Passes	33	30	14.5
Completed Passes	26	23	10

Table 4.4: Comparison of different stats between the team that uses Q-MST, MST and the team that uses a no-MST.

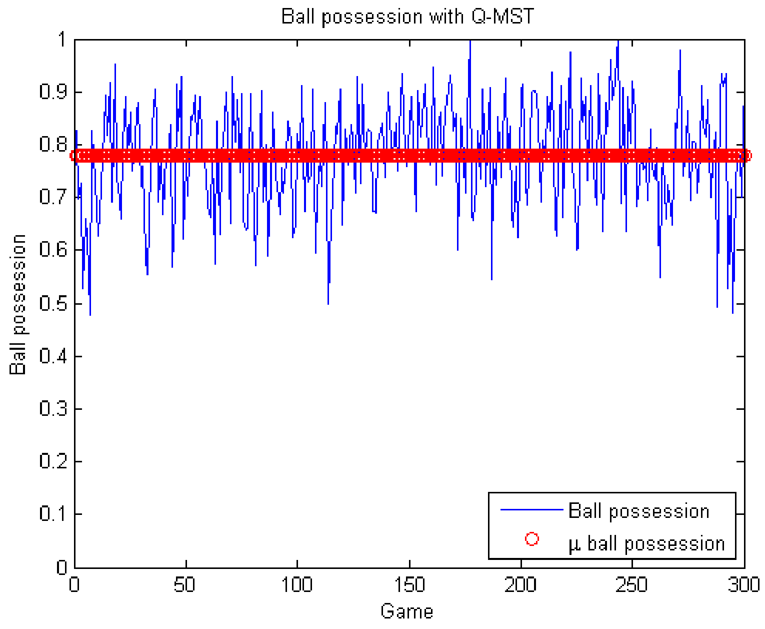

Figure 4.8: Ball possession percentage in each game when using Q-MST. It can be seen that this is always above the 50% of the time.

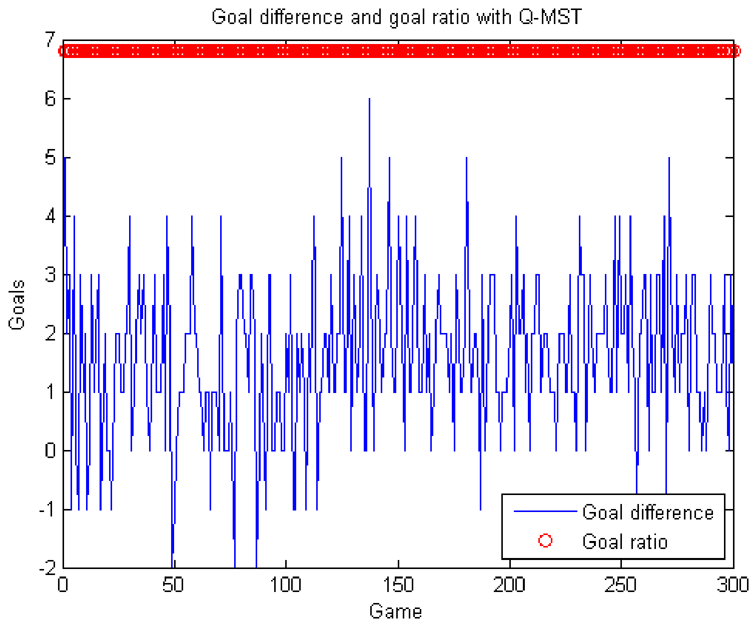

Figure 4.9: Goal difference for each of the games. It can be seen that in most of the games, the goal difference is well above 0. Also, the goal ratio for the games stood at 6.8:1.

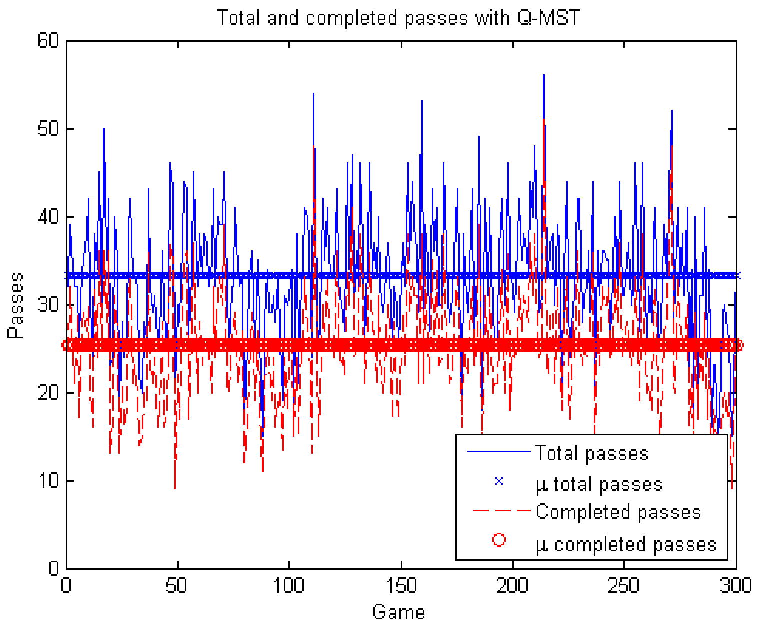

Figure 4.10: Total passes per game along with the number of completed passes. The total passes average was around 33 passes per game.

4.5 Summary

The experiments presented in this chapter show promising results. When comparing the proposed method to a simple graph for decision making, great improvements are obtained. An increased possession time, a much higher goal ratio and more total and completed passes. The increased possession time and passes lets the team have more opportunity to score more goals and takes that chance from the opposing team. Also, when using Q-learning, the team increases its effectiveness when playing, increasing the goal ratio by almost 80%. Finally, these results show that the method is good enough for decision making.

Chapter 5

Conclusion

This chapter states the conclusions obtained from the research process. First, a summary of the research presented so far is given. Then, revisit the hypothesis and the research questions to answer them according to the results acquired from the experiments. Finally, the chapter ends with several proposals for future work related to this thesis that can be done in the near future.

5.1 Research Summary

One of the biggest challenges that a soccer match presents in a game is not only that of the physical training the players need, but that of team play. Players need to be proficient at making passes and shots, as well as advanced plays as a pass into space. The problem this thesis tackles is that of the pass into space. In this play, a player that possesses the ball, the attacker, tries to make a pass into an empty space of the field to which another agent, the receiver, can run to to get the ball.

To solve this problem in the RoboCup soccer 2D environment, there are two aspects that must be taken into consideration. First, the attacker needs to correctly evaluate the situation to know if he can risk doing a pass into space. Second, the agent that the pass is ”targeted at” needs to know he is the receiver. The solution proposed uses a graph to model paths along the field while using Bayesian Networks to calculate the weights of the edges. Nevertheless, two BN are actually used, one to calculate the weights of the graph and obtain the MST and another to recalculate the weights in the MST and select the best action. By selecting the best action from the MST, it is possible that the one selected is not the best from the whole graph. Yet it is believed that by doing this, suboptimal decisions in the long run are avoided. Finally, to obtain even better results, Q-learning is used for offline learning of strategies. This with the purpose of adapting to the opposing team’s play style.

In order to test the method proposed, several experiments were designed. All of the soccer games are played against a base team that has the same structure as the home team, the one used to test the solution. The difference between the teams would be the decision making system. The base team using a simple graph, while the home team the method that this thesis presents. From here games are played first without the use of reinforcement learning. This team is called the MST team. Then without the use of the MST to prove the avoidance of suboptimal decisions for the game. This other team is called the no-MST team. Finally the first case is retaken with a team that has trained using Q-learning. With the last one being called Q-MST.

From the results obtained, and presented on the previous chapter, it can be seen that the use of the proposed method can deal with the problem of decision making a lot better than a simple graph. By adding a Bayesian network to the calculation of the weights of the edges, this process becomes more dynamic and reflects the current state of the game. Not only that, but by recalculating the weights after some of them have already been selected, for example by using the MST, the method can avoid getting stuck at a suboptimal decision. Q-learning is implemented offline to let the players be able to train against an opposing team.

5.2 Research Questions Revisited

In order to solve the problem of the pass into space, the following hypothesis was proposed: "By using Bayesian Networks to calculate the weights of a graph, better decisions can be taken by the agents. Also, by getting the minimum spanning tree of the graph and recalculating the weights of the edges with a different network, the decisions taken will lead to better results by avoiding getting stuck on a suboptimal decision for the game. Finally, offline learning can be used in order to improve the results obtained when playing against an opponent." From here, the following questions arose:

> *Is it possible to use a model based on graphs that lets the receiver know when a pass into space is for them? and is it possible to use a model that lets the attacker know where a pass into space is being aimed at?*

As seen on the results, the use of the proposed method greatly increases the total and completed passes the team had. When using the MST against the base team, the first one achieved an average of 30 passes in game, with an average of 22 completed. While the opponents got 12 passes per game and an average of 5 completed passes per game. This marks an improvement of more than 150% in passes. This comes as no surprise as the MST team had a ball possession average of 76%.

For the no-MST team, games did not fared as well. Although it obtained better results than the base team in this experiment, the total passes in game only reached 15, with 10 completed passes. This marks an improvement of the MST vs. the no-MST of 100%. In the case of the ball possession, this stalled at 70%, showing that the problem was not one of having the ball less time but one of suboptimal decisions.

In the case of the Q-MST there was not much change in the passes obtained. Similar results to the MST team showed with 33 passes per game, with 26 completed in average. This small improvement can be neglected as it is far too small. Nevertheless, there was also a small improvement in the total possession time, scaling to 79% in average. Again not big enough to show a difference.

Based on these results, it can be concluded that the use of the proposed method greatly increases the total number of passes into space as well as the number of completed passes done in the game.

Will the agents be able to train in order to have a better play style? and can offline learning be good enough for multiple games against the same team?

When using the MST against the base team, games were won with high scores. Standing at a 3.9:1 goal ratio, this team fared greatly against a simple graph for decision making. Not only that but also 77% of the games were won, while only losing 9% of the played games. This marked a large improvement when using this thesis method. When comparing this results to the no-MST team, the difference in the goal ratio is clearly marked. With a ratio of 1.5:1, it is clear that the use of the MST and two Bayesian Networks greatly improves the results obtained. This ratio also shown in the number of games won, that stood at 44%, with 25% games lost.

When comparing these results to the Q-MST team although no improvement seemed to have occurred from previous results, the scores and games won are a different story. The goal ratio of the team with learning stood at 6.8:1, which almost doubled the previous result (3.9:1). The total matches won rose to 81% while the lost games were reduced to 6%. All these show that training against the team vastly improved the result of the method.

From here, it can be concluded that although there is no increase on the number of passes and in the possession, each pass counted a lot more than those from previous experiments. Q-learning allowed to update the network to values that favored more the team than the ones previously used.

All the results presented point out that the method proposed in this thesis can be effectively used for decision making. Not only that, but the problem of the pass into space can be easily handled by the team with great results. Finally, the inclusion of Q-learning can increase the good results even more.

Although the main focus of this thesis was the implementation of a decision making method to let the agents make more and better passes into space, the method proved to be more than just that. The addition of Bayesian Networks to the graph allowed the method to make decisions that could avoid suboptimal solutions. The inclusion of a method to solve problems that uses graphs could be extended to more than the environment of the RoboCup.

5.3 Future Work

Even though the results obtained in this thesis are very promising for use in decision making, there is still a lot of work that can be done with it both in the RoboCup environment and in different ones as well.

- The proposed learning method was extensively tested on offline learning environments yet online learning needs to be researched more deeply. The use of different learning parameters in order to check or dismiss the usefulness for learning in game is very important. A team that could use not only a good decision making method, but one that could greatly improve in the same match would give the team a great edge against any opponent.
- Use of the proposed method as the main decision making algorithm on a team that competes on the RoboCup Soccer simulation 2D category.
- Increase the number of possible actions to take by the agents. For example, the edges labeled as strategic could be used to let the agents create dynamic plays in the game. Or let the agents run with the ball to another node.
- The use of the method in different scenarios during the game, for example when making defense strategies as a team.
- The use of a graph with Bayesian Networks for calculating the weights of the edges in different domains. If great improvements could be achieved in this environment, it is plausible that it works in other areas as well.

Bibliography

[1] Aijun Bai, Haochong Zhang, G. L. M. J., and Chen, X. Wrighteagle 2d soccer simulation team description 2012. `http://www.wrighteagle.org/2d/`, march 2012. Online; accessed 16-June-2012.

[2] B. Max, M., and Lynn, J. Symptom research interactive clinical research textbook. Interactive Textbook. `http://painconsortium.nih.gov/symptomresearch/index.htm`.

[3] Baase, S. *Computer algorithms: introduction to design and analysis.* Addison-Wesley series in computer science. Addison-Wesley Pub. Co., 1988.

[4] Chen, M., Dorer, K., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Murray, J., Noda, I., Obst, O., Riley, P., Steffens, T., Wang, Y., and Yin, X. *Users Manual: RoboCup Soccer Server — for Soccer Server Version 7.07 and Later.* The *R*obo*C*up Federation, February 2003.

[5] Esmaeil Aatashpaz-Gargari, B. N. A., and Lucas, C. Optimal fuzzy passing strategy for robot soccer players. In *2008 IEEE International Conference on Fuzzy Systems* (2008), IEEE, pp. 1576–1581.

[6] Faria, B., Reis, L., Lau, N., and Castillo, G. Machine learning algorithms applied to the classification of robotic soccer formations and opponent teams. In *Cybernetics and Intelligent Systems (CIS), 2010 IEEE Conference on* (june 2010), pp. 344 –349.

[7] Kaitwanidvilai, S., and Parnichkun, M. Active bayesian feature weighting in reinforcement learning robot. In *Industrial Technology, 2002. IEEE ICIT '02. 2002 IEEE International Conference on* (dec. 2002), vol. 2, pp. 1090 – 1095 vol.2.

[8] Li Xiong, Chen Wei, G. J. Z. Z. H. Z. A new passing strategy based on q-learning algorithm in robocup. In *2008 International Conference on Computer Science and Software Engineering* (2008), IEEE, pp. 524–527.

[9] Liang Liu, L. L. Regional cooperative multi-agent q-learning based on potential field. In *Fourth International Conference on Natural Computation* (2008), IEEE, pp. 535–539.

[10] Luis Mota, Nuno Lau, L. P. R. Co-ordination in robocups 2d simulation league: Setplays as flexible, multi-robot plans. In *2010 IEEE Conference on Robotics, Automation and Mechatronics-* (2010), IEEE, pp. 362–367.

[11] Nakashima, T., Uenishi, T., and Narimoto, Y. Off-line learning of soccer formations from game logs. In *World Automation Congress (WAC), 2010* (sept. 2010), pp. 1 –6.

[12] Noda, I., Suzuki, S., Matsubara, H., Asada, M., and Kitano, H. Overview of robocup-97. In *RoboCup-97: Robot Soccer World Cup I*, H. Kitano, Ed., vol. 1395 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 1998, pp. 20–41.

[13] Oliver, J. J. Decision graphs - an extension of decision trees, 1993.

[14] Plant, W., and Schaefer, G. An overview of neural networks in simulation soccer. In *Nature Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on* (dec. 2009), pp. 1620 –1625.

[15] Quintana, C. Un enfoque de aprendizaje por refuerzo para generar estrategias de pase de baln en la liga de plataforma estndar de robocup. Master's thesis, May 2011.

[16] RoboCup. Robocup @home. Internet webpage `http://www.ai.rug.nl/robocupathome/`, 2012. Online; accessed 29-October-2012.

[17] RoboCup. Robocup rescue. Internet webpage `http://www.robocuprescue.org/`, 2012. Online; accessed 29-October-2012.

[18] Russell, S., and Norvig, P. *Artificial intelligence: a modern approach.* Prentice hall, 2009.

[19] Sheehan, M., and Watson, I. On the usefulness of interactive computer game logs for agent modelling. In *PRICAI 2008: Trends in Artificial Intelligence*, T.-B. Ho and Z.-H. Zhou, Eds., vol. 5351 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 1059–1064.

[20] Sutton, R., and Barto, A. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning. MIT Press, 1998.

[21] Takahashi, K., Ueda, H., and Miyahara, T. Agent learning in simulated soccer by fuzzy q-learning. In *TENCON 2004. 2004 IEEE Region 10 Conference* (nov. 2004), vol. B, pp. 338 – 341 Vol. 2.

[22] Torsello, A., and Dowe, D. Supervised learning of a generative model for edge-weighted graphs. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on* (dec. 2008), pp. 1 –4.

[23] Wei Chen, Jing Guo, X. L. J. W. Hybrid q-learning algorithm about cooperation in mas. In *2009 Chinese Control and Decision Conference* (2009), IEEE, pp. 3943–3947.

[24] Wilson, Robert A. ; Keil, F. *The MIT Encyclopedia of the Cognitive Sciences (MITECS).* The MIT press, 1999.

[25] Zishen, L., Wei, L., and Maoqing, L. Modeling decision and cooperation of multi-agent system using petri net. In *Computer Science Education, 2009. ICCSE '09. 4th International Conference on* (july 2009), pp. 643 –646.

Vita

Alejandro Garza Cuéllar was born in 1988, in Cd. Victoria, in the Tamaulipas state in Mexico. He obtained his B. Sc. in Physics Engineering from the Instituto Tecnológico y de Estudios Superiores de Monterrey in 2010. In January 2011, he pursued a Masters Degree in Intelligent Systems under the guidance of his thesis advisor Ph.D. Leonardo Garrido Luna. He is grateful to all the people who aided him along his work and he is glad he could pursue studies in the research area he always dreamed, Artificial Intelligence.

The present thesis was typeset with LaTeX[1] by Alejandro Garza Cuéllar.

[1]The macro package, `ITESMtesis.sty`, used in the formating of this thesis was written by Ph. D. . Horacio Martínez Alfaro <`hma@itesm.mx`>, Associate Professor of the Intelligent Systems Center of Instituto Tecnológico y de Estudios Superiores de Monterrey, Campus Monterrey.