REPRESENTING UNCERTAINTY IN RTS GAMES

Björn Jónsson
Master of Science
Software Engineering
January 2012
School of Computer Science
Reykjavík University

M.Sc. PROJECT REPORT

Representing Uncertainty in RTS Games

by

Björn Jónsson

Project report submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Software Engineering**

January 2012

Project Report Committee:

Dr. Yngvi Björnsson, Supervisor
Accociate Professor, Reykjavík University, Iceland

Dr. Hannes Högni Vilhjálmsson
Accociate Professor, Reykjavík University, Iceland

Dr. Jón Guðnason
Assistant Professor, Reykjavík University, Iceland

Representing Uncertainty in RTS Games

Björn Jónsson

January 2012

Abstract

Real-time strategy (RTS) games are partially observable environments, requiring players to reason under uncertainty. The main source of uncertainty in RTS games is that players do not initially know the game map, including what units the opponent has created. This information gradually improves, in part by exploring, as the game progresses. To compensate for this uncertainty, human players use their experience and domain knowledge to estimate the combination of units that opponents control, and make decisions based on these estimates. For RTS game AI to mimic this behavior of human players, a suitable knowledge representation is required. The order in which units can be created in RTS games is conditioned by a game specific technology tree where units represented by parent nodes in the tree need to be created before units represented by child nodes can be created. We propose the use of a Bayesian Network (BN) to represent the beliefs that RTS game AI players have about the expansion of the technology tree of their opponents.

We implement a BN for the RTS game *StarCraft*® and give several examples of its use. In particular, we evaluate our design by improving strategy prediction under uncertainty from previously reported work [37]. Using our BN, we are able to increase the precision of strategy prediction up to 56%. These results show that the proposed BN can be used to infer creation time values for unobserved units in RTS games and that BNs are a promising approach for RTS game AI to represent and reason with uncertainty in RTS games.

Meðhöndlun óvissu í rauntíma herkænskuleikjum

Björn Jónsson

Janúar 2012

Útdráttur

Rauntíma herkænskuleikir (RHL) hafa þann eiginleika að umhverfi þeirra er að hluta til hulið. Þessi eiginleiki krefst þess af spilurum að þeir geti framkvæmt ákvarðanatöku undir óvissu. Aðal orsök óvissunnar í RHL er að spilarar hafa ekki yfirsýn yfir umhverfið sem spilað er í og vita því ekki hvernig her mótherjans er samsettur. Til að vega upp á móti þessari óvissu nota mennskir spilarar reynslu sína og þekkingu á reglum leiksins til þess að meta líklega samsetningu herafla mótherjans hverju sinni og byggja ákvarðanir sínar í leiknum á þessu mati. Til þess að RHL ármaður gæddur gervigreind eigi möguleika á því að herma eftir hæfileikum mennskra spilara til að meðhöndla óvissu í RHL þarf að nota hentuga þekkingartáknun. Við stingum upp á Bayesian netum (BN) sem þekkingartáknun fyrir tiltrú RHL ármanna á samsetningu herafla mótherjans.

Við útfærum BN fyrir rauntíma herkænskuleikinn *StarCraft*® og sýnum fram á ýmsa notkunarmöguleika þess. Sér í lagi skoðum við hvernig útfærsla okkar getur bætt nákvæmni í spágildum um hvaða leikkerfi mótherjinn er að spila. Bætingin sem náðist í spágildum með notkun Bayesian neta var allt að 56% yfir áður birtar niðurstöður [37]. Þessar niðurstöður sýna að BN er áhrifarík þekkingartáknun fyrir RHL og að notkunarmöguleikar þess eru miklir.

I dedicate this work to my three sons; Jón, Lárus and Hrafnkell. You are my inspiration and constant source of motivation to improve and become a better human being.

Acknowledgements

I want to thank Ben G. Weber for providing me with ideas, data and detailed information about his work. I want to thank Dr. Anders L. Madsen for his encouragement when I was starting to think about Bayesian Networks and their possible application to RTS games. I want to thank Dr. Santiago Ontañón for his advice on implementing a BN in the Darmok 2 case-based reasoning framework. I want to thank Gabriel Synnaeve for reading over and providing useful comments on my results. I want to thank Dr. Björn Þór Jónsson for his good will throughout my time at Reykjavik University and my supervisor Dr. Yngvi Björnsson for his support and guidance. I want to thank my dear friend and fellow student, Oddur Óskar Kjartansson, for countless hours of brainstorming and advice about life, the universe and everything. Finally, I want to thank my family for all their support during my masters studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Real-time strategy (RTS) games are complex domains with huge decision and state spaces. They are non-deterministic, partially observable and real-time [5, 28]. These characteristics present a major challenge for RTS game AI and offer a large variety of fundamental research problems. One particular challenge is handling the partial observability of the environment. Bayesian Networks (BN) [29] have in recent years been successfully applied to create consistent probabilistic representations of uncertain knowledge in many different kinds of domains. They are particularly effective for modelling situations where some information is already known and new data is uncertain or partially unavailable, as is the case in RTS games.

In an ongoing RTS game only partial information, acquired from scouting the environment, is available about what units and buildings your opponent has constructed. The order in which a player can construct units and buildings in RTS games is represented with a technology tree (tech tree)[1]. Expert human players use knowledge from previously played games to infer the probability of particular units or buildings existing at a given time, from observed evidence of other units or buildings. They make a mental model of how they believe their opponents are expanding their tech tree and make decisions based on this mental model. In this report we propose the use of a Bayesian Network to represent the beliefs that AI players have about the expansion of the tech tree of their opponents.

The concepts of prior and posterior probabilities and evidence used in Bayesian approaches fit well to our problem. We can use RTS game logs to find prior probabilities of different units and buildings being built at each time in a game and we can update

[1] A tech tree in RTS games describes in what order units and buildings can be constructed. A more detailed description is found in Chapter 2.1.1

the corresponding network in real-time with evidence about observed units and buildings. We believe that our approach, using BNs, mimics well the assessment that human players make about how their opponents have expanded their tech tree, for which most of their in-game decisions are based upon. Another strong point in favor of this approach is that the underlying data structures for both RTS game tech trees and BNs are directed acyclic graphs (DAG). This results in BNs that are both easy to construct and intuitive to work with.

Our approach of using a BN to represent uncertain knowledge has the potential to improve RTS games in various ways. We give an overview of several such directions, including work we have done towards using BNs in case-based reasoning and in-game commentary assistance. However, our main use case will be to improve strategy prediction. For this purpose, we compare classification results for the task of predicting opponent strategies in StarCraft games, with and without estimations from a BN. By training the network on a large collection of game logs we can use probabilistic inference algorithms to infer the probabilities of unobserved units and buildings, given observed evidence for a particular game, and use this information to augment the feature vectors used in the strategy prediction with the corresponding estimations from the network. We report results of around 60% increase in the precision of strategy prediction.

1.1 Problem Statement

RTS games are domains with interesting challenges for AI. One of the main reasons for this is that game playing agents in RTS games have to reason with uncertain knowledge that originates from the fact that they operate in a partially observable environment. With this in mind we aim to answer the following questions in this work:

- Is a BN an appropriate model to represent uncertain knowledge in RTS games?
- How can it be designed to give useful probabilistic inferences for the RTS domain?
- Is it possible to use the network to improve other AI techniques, and in particular to improve strategy prediction?

1.2 Goals and Contribution

The goal of this project is to establish BNs as a viable approach for representing uncertain knowledge in RTS games. We show how a BN can be constructed from RTS game

logs and used to improve the precision of strategy prediction in an imperfect information environment. Both the data used and the BN component will be made available to the community.

1.3 Overview of Report

We start by providing the necessary background information for RTS games in Chapter 2, describing the mechanics of RTS games and the challenges that they provide for AI research. We then cover the basics of BNs. In Chapter 3 we describe the process of constructing a BN from StarCraft game logs, discussing the reasoning and motivation for using BNs to represent tech trees in RTS games and describing an implementation of a BN from StarCraft game logs. Chapter 4 discusses the possible applications for the proposed BN, with specific sections for in-game commentary assistance, case based reasoning and strategy prediction. In Chapter 5 we then present a thorough case study on improving strategy prediction in imperfect information environments. We conclude our overall findings and results in Chapter 6.

Chapter 2

Background

2.1 Real-Time Strategy Games

From the inception of the field of artificial intelligence (AI), over half a century ago, classical games have played an important role as a test-bed for the advancement in the field. Some examples of how AI agents have surpassed human capabilities are from Chess, Checkers and most recently IBM's Watson's victory over current Jeopardy champions [10]. Lately, with advancements in computing, complex computer games like RTS games and massively multi-player online role-playing games (MMORPG) have become one of the largest sector of the entertainment industry. These games are particularly appealing for AI research as their characteristics make them a virtual environment that is in many ways similar to the real world. RTS games are complex domains with huge decision and state spaces. They are non-deterministic, partially observable and real-time [5]. These characteristics present a major challenge for RTS game AI and offer a large variety of fundamental research problems, including decision making under uncertainty, opponent modeling, learning and adversarial real-time planning [6].

RTS games are popular as an e-sport, with hundreds of professional players that compete in tournaments all over the world. Recent interest in AI research for RTS games has materialized in a StarCraft AI competition that is held in connection with the Artifical Intelligence for Interactive Digital Entertainment (AIIDE) conference each year. The competition is a forum for universities and independent researchers to bring their AI bots and let them compete against each other. In 2011 over 30 teams registered to participate in the competition.

2.1.1 The RTS Domain

Real-Time Strategy (RTS) is a category of strategy games that usually focuses on military combat. RTS games such as StarCraft require the player to control armies (consisting of different types of units) and defeat all opposing forces that are situated in a virtual battlefield (a map) in real-time. Players on different sides gather resources to build structures, produce units and research technology upgrades with the aim of attacking and eliminating their opponents. Resources are mined from specific map locations and hence controling and harvesting from as many resource locations as possible is vital for success in the game. In most RTS games, the key to winning lies in efficiently collecting and managing resources, and appropriately distributing these resources over the various game action elements [30].

In general, there are two main competencies that RTS game play requires [38]. The first competency is micromanagement, which is the task of controlling individual units in combat scenarios in order to maximize the utility of a squad of units. The second is macromanagement or strategy selection, which is the task of determining what types of units to produce and which upgrades to research. This area of expertise focuses on planning and opponent modeling, and involves formulating strategies in order to counter opponent strategies. The unit types and buildings in RTS games are defined by a tech tree, which is a DAG that models the possible paths of research a player can take within a game. Strategies within RTS games largely revolve around executing different build orders, which define the order in which a player expands the tech tree. There are many types of strategies but the two most common categories are [35] rush strategies, that try to overwhelm the opponent with inexpensive units early in the game, and timing attacks, where the opponent is engaged based on a trigger event or estimation that maximum damage will be inflicted by attacking at that time. What kind of strategy is executed therefore often depends on how the player believes his opponent has expanded his tech tree. Since RTS games enforce a partially observable environment, there is always some uncertainty about how the opponent has expanded his tech tree at a particular moment.

In RTS games an action can be viewed as an atomic transformation in the game situation. Typical game actions in RTS games include constructing buildings, researching new technologies, and coordinating units in combat.

The action space in RTS games, which is defined as the set of possible actions that can be executed at a particular moment can be estimated as [3]:

$$O(2^w(A * P) + 2^T(D + S) + B(R + C)) \tag{2.1}$$

Where:

- w = current number of workers
- A = number of assignments workers can perform
- P = average number of workplaces
- T = number of troops
- D = average number of directions that a unit can move
- S = choice of troops stance (stand, patrol, attack)
- B = number of buildings
- R = average choice of research objectives at a building
- C = average choice of units to create at a building

To get a feeling for the vast complexity of RTS games, one can imagine playing chess on a 512×512 tile board with hundreds of pieces moving simultaneously in real-time. Early game decision complexity for RTS games is according to this estimation around 1.5×10^3, which is substantially higher than the average number of possible moves in most board games (e.g., for chess, this is approximately 30).

2.1.2 AI Research for RTS Games

RTS game developers working in the industry commonly use simple techniques like finite state machines, fuzzy state machines, scripting, expert systems and other rule-based approaches for the AI they require [9]. While these approaches have been proven to be successful for the industry, they are limited in terms of adaptability and dealing with uncertainty. Using these techniques often leads to game-playing agents that become predictable and less enjoying for the user to interact or compete with after playing the particular game for a while. Other less deterministic techniques such as neural networks, evolutionary computation, hierarchical task networks and BNs have a much greater potential to deliver better and more adaptable solutions that will provide the user with a consistently challenging experience. These approaches are however often not used in

practice because the agents behavior has been viewed as unreliable and these techniques are much more computationally expensive than traditional rule based ones.

Along with the lack of CPU resources, the game industry focus on graphics technology has probably played the biggest role in holding back commercial game AI development. Since graphics processing has moved to specialized graphics hardware and graphics have now reached near photo-realistic experiences, game developers are increasingly looking towards AI as key features in future games and the main area of innovation and excitement within the industry.

In summary, the term game AI is used differently by game developers and academic researchers [30]. Academic researchers restrict the use of this term to refer to intelligent behaviours of game characters that are mostly the result of using techniques from machine learning and automated planning [30]. In contrast, for game developers, game AI is used in a broader sense to encompass techniques such as pathfinding, animation systems, level geometry and overall behavior of non-playing characters or gameplaying agents.

Setting these differences aside, the main arguments in favor of RTS game AI research are [6]:

- RTS games constitute well-defined environments to conduct experiments in and offer straight-forward objective ways of measuring performance
- RTS games can be tailored to focus on specific aspects such as how to win local fights, how to scout effectively, how to build, attack and defend a base, etc.
- Strong game AI will likely make a difference in future commercial games because graphics improvements are beginning to saturate.

As stated before RTS games provide many challenges for AI research, in part because they feature hundreds or thousands of interacting objects, require simultaneous moves of these objects in real-time and enforce imperfect information through a concept called "fog of war".

This results in a multi-agent, real-time and partially observable task environment. Given these properties three key areas of RTS game AI where research is needed have been identified [6]:

- **Adversarial planning under uncertainty**

  Because of the huge number of possible actions at any given time and their microscopic effects in RTS games, agents cannot afford to think at the game action level. Instead, abstractions of the world state have to be found that allow programs to conduct forward searches in abstract spaces and to translate found solutions back into the original state space.

- **Learning and opponent modeling**

  One of the biggest shortcomings of current RTS game AI systems is their inability to learn quickly. Human players only need a couple of games to form accurate hypotheses about their opponents playing styles and weaknesses.

- **Spatial and temporal reasoning**

  Understanding the importance of static terrain features like choke points and dynamic spatial properties such as visibility and enemy influence is crucial for generating successful plans in RTS games. Another important aspect is the temporal relationship among actions and the temporal reasoning needed to guide decisions in RTS games.

There are other areas of research in RTS game AI, such as research into the use of multi-agent potential fields [13], cognivite architectures [39] and many more. But these three key areas encompass the bulk of RTS game AI research. New efficient machine learning techniques have to be developed to tackle these important problems.

2.1.3 StarCraft

StarCraft: Brood War is a canonical RTS game, as Chess is to board games. It was published in 1998, has sold over 10 million copies and has numerous international competitions like the World Cyber Games. The single objective of players is to destroy all opponents in a military scenario. There are three races in StarCraft (Terran, Protoss and Zerg) that have different units, tech trees and thus gameplay styles. We decided to use StarCraft in our work because it is the most active RTS research platform, with AI competitions, open source bots and publicly available game log data.

2.2 Bayesian Networks

A Bayesian network (BN) [29] is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). A BN reflects the simple conditional independence statement that each variable is independent of its nondescendents in the DAG given the state of its parents. Each node in the DAG represents a random variable and is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. In many cases this is represented with a conditional probability table (CPT) for each node. The joint probability distribution (JPD) of a collection of variables can be determined uniquely by these local CPTs and the corresponding reduction in the number of parameters that are required to characterize the JPD provides an efficient way to compute the posterior probabilities of a subset of variables given evidence about other variables in the distribution [29]. Bayesian networks are generally used for modeling knowledge and performing probabilistic reasoning with uncertain information. These networks are particularly effective for modeling situations where some information is already known and incoming data is uncertain or partially unavailable. BNs are currently the dominant uncertainty knowledge representation and reasoning technique in AI [12].

To a large extent, BNs are a marriage between probability theory and graph theory [18], in which dependencies between random variables can be expressed graphically. We begin by briefly discussing these foundations of BNs.

2.2.1 Probability Theory

Probability theory deals with the formal foundations for discussing estimates of the degree of confidence that an event of an uncertain nature will occur, and the rules that these estimates obey. We denote Ω as the space of possible outcomes that an event can take and we assume that there is a set of measurable events $\mathcal{S}$ to which we are willing to assign probabilities.

For $\alpha \in \mathcal{S}$ as a subset of Ω, the probability $P(\alpha)$ quantifies the degree of confidence that α will occur. If $P(\alpha) = 1$, we are certain that one of the outcomes in α occurs, and if $P(\alpha) = 0$, we consider all of them impossible.

A probability distribution P over $(\Omega, \mathcal{S})$ is a mapping from events in $\mathcal{S}$ to real values that satisfies the following conditions:

- $P(\alpha) \geq 0 \,, \forall\, \alpha \in \mathcal{S}$
- $P(\Omega) = 1$
- If $\alpha, \beta \in \mathcal{S}$ and $\alpha \cap \beta = \emptyset$, then $P(\alpha \cup \beta) = P(\alpha) + P(\beta)$

The above discussion of probability distributions deals with events. We often want to refer to attributes of events in a clean mathematical way. The formal machinery for discussing attributes and their values in different outcomes are random variables.

Using the notion of random variables we call:

- $P(X = x_i)$, the **marginal probability** of $X = x_i$
- $P(X = x_i, Y = y_i)$, the **joint probability** of $X = x_i$ and $Y = y_i$
- $P(X = x_i | Y = y_i)$, the **conditional probability** of $X = x_i$ given $Y = y_i$

Independence and conditional independence between random variables can be defined as:

Definition 1. *(X and Y are independent)*
$P(X = x, Y = y) = P(X = x)P(Y = y)(\forall x, y)$.

Definition 2. *(X and Y are conditionally independent, given Z)*
$P(X = x, Y = y | Z = z) = P(X = x | Z = z)P(Y = y | Z = z)(\forall x, y, z)$.

The difference between these definitions is very important, the first says that knowing the value of X tells us nothing about the value of Y, while the second is saying that if we know Z, then knowing X tells us nothing more about Y, that is, even though X and Y may be dependent, their dependence is entirely "explained" by Z. If we skip using the cumbersome notion of $P(X = x)$ and instead simply write $P(X)$ to denote the distribution over the random variable X, we can write the two fundamental rules of probability theory in the following form.

The Rules of Probability

(sum rule)

$$P(X) = \sum_{Y} P(X, Y) \tag{2.2}$$

(product rule)

$$P(X, Y) = P(Y | X)P(X) \tag{2.3}$$

Here $P(X, Y)$ is a joint probability and is verbalized as "the probability of X and Y". Similarly, the quantity $P(X|Y)$ is a conditional probability and is verbalized as "The probability of X given Y"', whereas the quantity $P(X)$ is a marginal probability and is simply "the probability of X". These two simple rules form the basis for all of the probabilistic machinery that is used in this report. From the product rule, together with the symmetry property $P(X, Y) = P(Y, X)$, we immediately obtain the following relationship between conditional probabilities:

(Bayes Theorem)

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{2.4}$$

In Bayes theorem, each probability has a conventional name:

- $P(X)$ is the prior probability or marginal probability of X. It is prior in the sense that it does not take into account any information about Y.
- $P(X|Y)$ is the conditional probability of X, given Y. It is also called the posterior probability because it is derived from or depends upon the specified value of Y.
- $P(Y|X)$ is the conditional probability of Y given X. It is also called the likelihood.
- $P(Y)$ is the prior or marginal probability of Y, and acts as a normalizing constant.

As we will see, Bayes theorem is the foundation of learning and inference in Bayesian Networks.

2.2.2 Graph Theory

The cornerstone of BNs is the representation of a probability distribution using a graph as a data structure. The underlying directed acyclic graphs (DAG) in BNs not only allows the user to understand which variables affect others, but also serves as the backbone for efficiently computing marginal and conditional probabilities that may be required for inference and learning in BNs. For our purposes, a brief definition of a DAG is sufficient. For a full background on graph theory see [2].

Definition 3. *(Directed Graph)*
A directed graph $\mathcal{G}$ = ($\mathcal{N}$,$\mathcal{E}$) is a set $\mathcal{N}$ of nodes and a set $\mathcal{E}$ of ordered pairs (i,j) where $i, j \in \mathcal{N} : i \neq j$.

Definition 4. *(Directed Acyclic Graph)*
A cycle in $\mathcal{G}$ is a directed path $N_1, ..., N_k$ where $N_1 = N_k$. A Directed Acyclic Graph (DAG) is a directed graph with no cycles.

2.2.3 Representation

In general the goal of a BN is to represent a joint distribution over some set of random variables $X = X_1, ..., X_n$. Even in the simplest case where these variables are binary-valued, a joint distribution requires the specification of $2^n - 1$ numbers, namely the 2^n different assignments of values $x_1, ..., x_n$. For all but the smallest n, the explicit representation of the joint distribution is unmanageable from every perspective. Computationally, it is very expensive to manipulate and generally to large to store in memory. Cognitively, it is in most cases impossible to acquire so many numbers from a human expert; moreover, the numbers are very small and do not correspond to events that people can reasonably contemplate. Statistically, if we want to learn the distribution from data, we would in most cases need very large amounts of data to estimate this many parameters robustly. BNs overcome these problems by exploiting the structure of the underlying DAG and using independence properties in joint probability distributions to represent them much more compactly [29].

Definition and Properties
If for the following we let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a directed acyclic graph and let X be a set of random variables. Then X is a Bayesian network with respect to $\mathcal{G}$ if it satisfies the *local Markov property*, each variable is conditionally independent of its non-descendants given its parent variables. Every entry in the corresponding joint probability distribution can be computed from the information in the BN by the chain rule:

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | Parents(X_i)) \tag{2.5}$$

Following the above discussion, a more formal definition of a BN can be given [11]:

Definition 5. *(Bayesian Network)*
A Bayesian network B is an annotated acyclic graph that represents a joint probability distribution over a set of random variables $\mathbf{X}$. The network is defined by a pair $B = \langle \mathcal{G}, \Theta \rangle$ where $\mathcal{G}$ is the directed acyclic graph whose nodes $N_1, N_2, ..., N_n$ represents random variables, and whose edges represent the direct dependencies between these variables. The graph $\mathcal{G}$ encodes independence assumptions, by which each variable X_i is independent

*of its nondescendents given its parent in $\mathcal{G}$. The second component, Θ, denotes the set of parameters of the network. This set contains the parameter $\Theta_{x_i|\pi_i}$ for each realization x_i of X_i conditioned on π_i, the set of parents of X_i in $\mathcal{G}$. Accordingly, B defines a unique joint probability distribution over **X**, namely:*

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|\pi_i) = \prod_{i=1}^{n} \Theta_{X_i|\pi_i} \tag{2.6}$$

Constructing Bayesian Networks

To construct a BN, for a set of variables $X = X_1, ..., X_n$, we first develop a directed acyclic graph $\mathcal{G}$ such that we believe X satisfies the local Markov property with respect to $\mathcal{G}$. We then ascertain the conditional probability distributions of each variable given its parents in $\mathcal{G}$.

Judea Pearl presented an algorithm for the construction of a BN [29]:

- Choose the set of relevant variables X that describe the domain.
- Choose an ordering for the variables, $X_1, ..., X_n$
- While there are variables left:
  1. Add the next variable X_i to the network.
  2. Add arcs to the X_i nodes from some minimal set of nodes already in the net, Parents(X_i), such that the following conditional independence property is satisfied:
     $P(X_i|X_1, ..., X_m) = P(X_i|Parents(X_i))$
     where $X_1, ..., X_m$ are all the variables preceding X_i , including Parents(X_i).
- Define the conditional probability table (CPT) for X_i

As described above, the nodes in $\mathcal{G}$ represent random variables and are usually drawn as circles labeled by the variable names. The edges represent direct dependence among the variables and are drawn by arrows between nodes. In particular, an edge from node X_i to node X_j represents a statistical dependence between the corresponding variables. Thus, the arrow indicates that a value taken by variable X_j depends on the value taken by variable X_i , or roughly speaking that variable X_i "influences" X_j.

2.2.4 Learning

In many practical settings the BN is unknown, so one needs to learn it from data. This problem is known as the *BN learning problem*, which can be stated informally as follows: Given training data and prior information in the form of expert knowledge or causal relationships, estimate the graph topology and parameters of a joint probability distribution in the BN.

In general, we can divide learning in BN into four cases, depending on the observability of the variables in the network and if the network structure is known. For our purposes only the two simpler cases where the network structure is known and the observability of the variables is full or partial needs to be considered. For parameter learning with known structure and full observability, the goal is to find the values of the BN parameters that maximize the log likelihood of the training data. Given a training dataset $\Sigma = \{x_1, ..., x_m\}$, where $x_l = (x_{l1}, ..., x_{ln})^T$, and the parameter set $\Theta = (\theta_1, ..., \theta_n)$, where θ_i is the vector of parameters for the conditional distribution of variable X_i, the loglikelihood of the training dataset is a sum of terms, one for each node.

$$logL(\Theta|\Sigma) = \sum_m \sum_n logP(x_{li}|\pi_i, \theta_i) \tag{2.7}$$

For parameter learning with latent variables the Expectation Maximization (EM) algorithm [7] can be used to find an (locally) optimal Maximum Likelihood Estimate (MLE) of the parameters. The basic idea behind EM is that, if we knew the values of all the nodes, learning (the M step) entails finding the maximum likelihood estimation of the conditional probability distribution (CPD), that is the CPD that maximizes the likelihood of the training data. So in the E step, we compute the expected values of all the latent variables using an inference algorithm, and then treat these expected values as though they were observed. Given the expected counts, we maximize the parameters, and then recompute the expected counts, etc. This iterative procedure is guaranteed to converge to a local maximum of the likelihood surface. In any case, we see that inference becomes a subroutine which is called by the learning procedure; hence fast inference algorithms are crucial.

2.2.5 Inference

Because a BN is a complete model for the variables and their relationship, it can be used to answer probabilistic queries about them. For example, the network can be used to find

out updated knowledge of the state of a subset of variables when other variables (the evidence) are observed. This process of computing the posterior distribution of variables given evidence is called probabilistic inference. In general, the objective of inference in BN is to calculate $P(X|Y)$, the posterior probabilities of query nodes X, given some observed values of evidence nodes Y. Inference is just answering queries using the distribution as our model of the world. Inference algorithms compute the posterior probability of some variables given evidence on others. For these kinds of queries, inference is generally orders of magnitude faster using a BN than manipulating the joint distribution explicitly.

For the problem of performing inference in a graphical model like BN, the structure of the network, both the conditional independence assertions it makes and the associated factorization of the joint distribution, is critical for our ability to perform inference effectively. Although there exist polynomial time inference algorithms for specific classes of BNs, ie., trees and singly connected networks, in general both exact and approximate inference in the networks is NP-hard [12]. Approximate BN inference algorithms include stochastic simulation algorithms, model simplification methods, search-based methods and loopy belief propagation. These algorithms are not used in this work.

Exact Inference

The most popular exact BN inference algorithm is the clique-tree propagation algorithm [19]. It first transforms a multiply connected network into a clique tree by clustering the triangulated moral graph of the underlying undirected graph, then performs message propagation over the clique tree. The clique propagation algorithm works efficiently for sparse networks, but its complexity is exponential in the size of the largest clique of the transformed undirected graph. In our work we are using a variant of the clique-tree algorithm called Hugin or Junction Tree Algorithm (JTA) [15]. The JTA is a method to extract marginalization in general graphs. In essence, it entails performing belief propagation on a modified graph called a junction tree. The basic premise is to eliminate cycles by clustering them into single nodes. The JTA can be stated as follows:

- If the graph is directed then moralize it to make it undirected
- Introduce the evidence
- Triangulate the graph to make it chordal
- Construct a junction tree from the triangulated graph
- Propagate the probabilities along the junction tree (via belief propagation)

For further background knowledge on exact inference in BNs see [18].

2.3 Related Work

Bayesian Networks

Dynamic Bayesian Networks are used by Albrecht et al. [4] to predict the next actions and locations of a player in a multi-user dungeon game. Their results show that Dynamic Bayesian Networks are promising behavior models when the causal structure of the network can be identified. Bayesian Grid Models are used by Jensen in [16] to identify the opponent strategy when given the positions of opponent units on the game map. Synnaeve et al. [34] used a Bayesian model with a variable exploiting the structure of the build tree to reduce the effects of unobserved values. They then use provided and their own labels to learn the parameters of this model, which in turns predicts the distribution on strategies. Ethan et al. [8] use hidden Markov models for strategy discovery from StarCraft game logs, seeking to avoid any potential biases of engineered (rule based) labels. They do not provide any evaluation of their approach using imperfect information.

Strategy Prediction

There are several techniques reported for strategy prediction in RTS games. Some researchers refer to predicting the strategy of an opponent as opponent modeling, plan recognition or behavior recognition. Weber et al. [37] examine strategy prediction using various supervised learning techniques on StarCraft game logs to classify build orders according to a set of rule based labels. They provide a novel representation of encoding the game logs and evaluate their approach for both perfect and imperfect information environments. Schadd et al. [32] implemented a hierarchical opponent model in the RTS game *Spring*, classifying opponents playing style (aggressive/defensive) at the top level and specific strategies at the bottom level. Their approach is rule based, using opponent unit counts. Kabanza et al. [17] use hidden Markov models for plan recognition in PHATT, an agent architecture used for opponent modeling, by encoding strategies as hierarchical task networks. Several papers report on related work using case-based reasoning [3, 14, 28]. Hsieh et al. [14] use a case-based reasoning approach using StarCraft game logs to evaluate and learn decisions and behaviors of human players and to train their system to predict player strategies. Their model cannot predict timings of past or future events becouse they do not use any notion of time.

Chapter 3

Constructing a Bayesian Network from RTS Game Logs

In this chapter we will show how Pearl's algorithm can be used to construct a Bayesian Network (BN), from RTS game logs, that is capable of representing uncertainty about an opponents tech tree expansion in RTS games. We start by discussing the motivations for using BNs and then we describe the overall design of the proposed network. Details for a StarCraft specific implementation are given in Section 3.2 and evaluations of the implemented network are provided in Section 3.3.

3.1 Design

A major strategic element in RTS games like StarCraft is build order, which defines the order in which a player expands their tech tree. Estimating the expansion of the tech tree of your opponent is a necessary part of expert gameplay in all RTS games, since most of the decisions made by a player are based on his belief of how the tech tree of the opponent has been expanded at each time or will be expanded at some specific time in the future. An expert human player uses knowledge from previously played games to infer the probability of particular units or buildings existing at a given time, from observed evidence of other units or buildings. He makes a mental model of how he believes his opponent is expanding his tech tree and makes decisions based on this mental model.

We propose the use of a BN to represent the beliefs that an AI player has about the expansion of the tech tree of his opponent. A BN can be used to find out updated knowledge of the state of a subset of variables in the network when other variables (the evidence)

is observed. This process of computing the posterior distribution of variables in the network given evidence is called probabilistic inference. The concepts of prior and posterior probabilities and evidence used in Bayesian approaches fits well to our problem. We can attain the prior probabilities of different units and buildings being built at each time in a game from game logs and then use that model in real-time to predict the likelihood of the existence of other buildings given that one or more buildings or units have been observed in an ongoing game.

3.1.1 Network Structure and Parameters

To see what is needed to construct the proposed network it is intuitive to follow Pearls algorithm for the construction of BN from the perspective of our RTS game requirements:

- Choose the set of relevant variables X that describe the domain.
- Choose an ordering for the variables, $X_1, ..., X_n$
- While there are variables left:
  1. Add the next variable X_i to the network.
  2. Add arcs to the X_i nodes from some minimal set of nodes already in the net, Parents(X_i), such that the following conditional independence property is satisfied: $P(X_i|X_1, ..., X_m) = P(X_i|Parents(X_i))$ where $X_1, ..., X_m$ are all the variables preceding X_i , including Parents(X_i).
- Define the CPT for X_i

For RTS games the variables are units and buildings of interest that sufficiently describe the domain. Selecting which units or buildings to include as variables in the network is optional as different selections are optimal for different purposes. The order of the selected units and buildings is decided by the tech tree and temporal constraints in the particular game. If building A is needed to construct building B, then A has higher order then B. If more than one unit or building of any type is included in the network then the order in which they are built is the order of the corresponding variables. The arcs between nodes are causal relationships between buildings and units in the particular RTS game. If building C produces units of type D, then an arc from C to D is added to the network. Also, an arc from the first unit produced of type D to the second unit produced of type D is added if more than one unit of that type was selected to be in the network. Finally, to set the parameters of the network (the CPT for each node) we use data extracted from

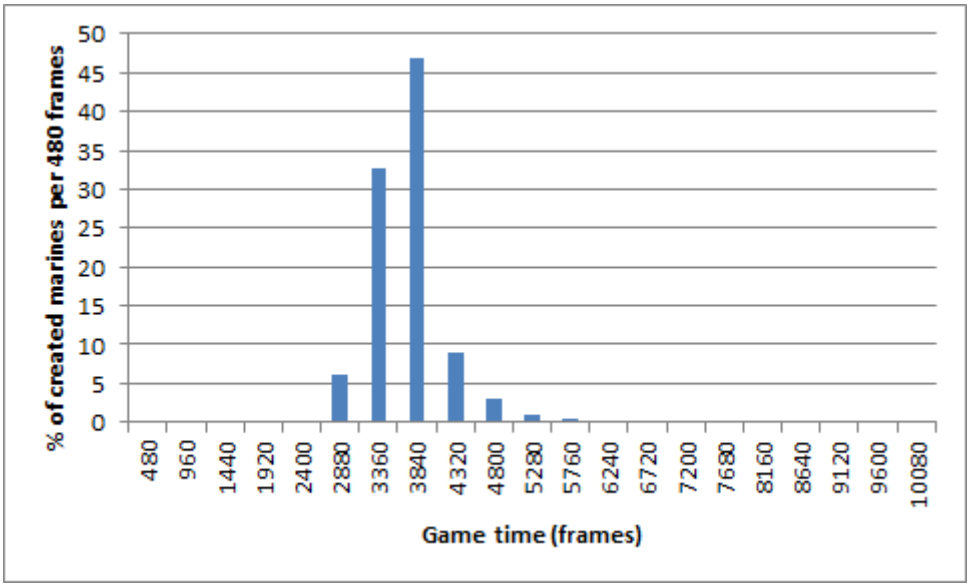


Figure 3.1: Histogram for Terran marine creation times using 20 states and state intervals of 480 frames.

game logs about the creation time of each unit and building. This allows the networks to be trained on general game logs from expert matches or specific known opponents.

This design captures the probabilistic relationships between unit and building creation times in RTS games. If we know the creation time of a building E then this information can change the probabilities for different creation times of building F, depending on the data used to train the network.

In our design, nodes represent discrete random variables. The state space for each node has a finite number of states, and both the number of states and their interval length can be decided when the network is constructed. In Figure 3.1 we can see an example of a histogram for the distribution of marine creation times for the RTS game StarCraft, using state intervals of 480 frames and 20 states. The semantics of arcs in the network is that the incoming arcs represent probabilistic dependency, in that the distribution of a node is conditionally independent of all non descendants of the node given its parents. The quantitative part of the relationship between a node and its parents is represented with the CPT for the node.

In summary, we are going to let the RTS game tech tree and temporal constraints for unit and building construction specify the structure of the BN and use data extracted from RTS game logs to learn the parameters of the network.

3.1.2 Functionality Provided by the Design

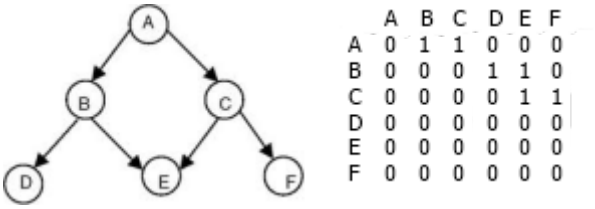
The following functionality is possible using the proposed BN:

- **Updating the network when new units or buildings are discovered in a game**
  As an RTS game progresses, more and more information about the opponent becomes available. In order for the network to use this information, when inferring probabilities of the existence for units or buildings that are still unobserved, it needs to be entered as evidence into the network. We can accomplish this by providing the name and id of the particular unit or building observed and the current game time to the network.

- **Getting the probability of units existing at the current time or at some time in the future**
  A common consideration for expert RTS human players is estimating when an opponent will get a particular unit or building. For example, estimating when an opponent will get units or buildings capable of detecting cloaked units, can be vital when deciding when to attack with an army of cloaked units. If provided with the name of the unit and the specified game time, the BN can be used to get the probability of that unit existing at that time, given what other units or buildings have been observed in the game.

- **Getting the estimated creation time values for unobserved units and buildings**
  For many purposes, it can be convenient for a RTS game AI to have an estimation of the creation time of units and buildings that are represented by nodes in the network and have not yet been observed in an ongoing game. These estimations can be used in state representations for other AI techniques. This functionality is possible, by e.g. using the expected value of the corresponding random variable for the particular unit or building.

- **Clear all observations and inferred values to start a new game**
  To start a new game without constructing and training the network all over again, our design allows all evidence entered to be retracted and the network potentials cleared to start a new game.



	A	B	C	D	E	F
A	0	1	1	0	0	0
B	0	0	0	1	1	0
C	0	0	0	0	1	1
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Figure 3.2: An example of an adjacency matrix representation of a DAG

3.2 StarCraft Specific Implementation

In this section we will discuss how to implement our proposed BN from StarCraft game logs. The game log data we will be using is provided by Weber et al. [37] and can be downloaded from [26]. Although we are focusing on StarCraft the same construction process applies to any RTS game with the canonical gameplay of gathering resources to extend a technology tree and produce military units. We used the Hugin Decision Engine [15] for Java to create our proposed Bayesian Network. The Hugin engine employs an exact and efficient algorithm for updating probabilities [19]. The Hugin API also contains a high-performance inference engine that can be used as the core of systems built using BNs. Using Hugin, it is possible to build models from data and/or probabilistic descriptions of causal relationships in the domain and given this model the Hugin inference engine can perform fast and accurate reasoning.

Network Structure

The network structure can be described with an adjacency matrix, adding the requirement that the matrix represents a DAG and the columns are always numbered in topological order, i.e., ancestors before descendants. An example of such a description can be seen in Figure 3.2.

We will be using data for the Terran race in StarCraft, so for constructing the adjacency matrix representation for the network structure we need domain specific information about the StarCraft Terran tech tree that can be found on the BWAPI repository [25]. The game log data we are using is on tabular format where each column represents a unit or building for the Terran race and each row represents the chosen information extracted from one game log. To construct the adjacency matrix we begin by writing down the order of the units and buildings for each column in the data set and then look up the tech tree requirements to see how to write the corresponding line in the adjacency matrix for each unit or building. A part of the network structure for the game log data used is shown in Figure 3.3.

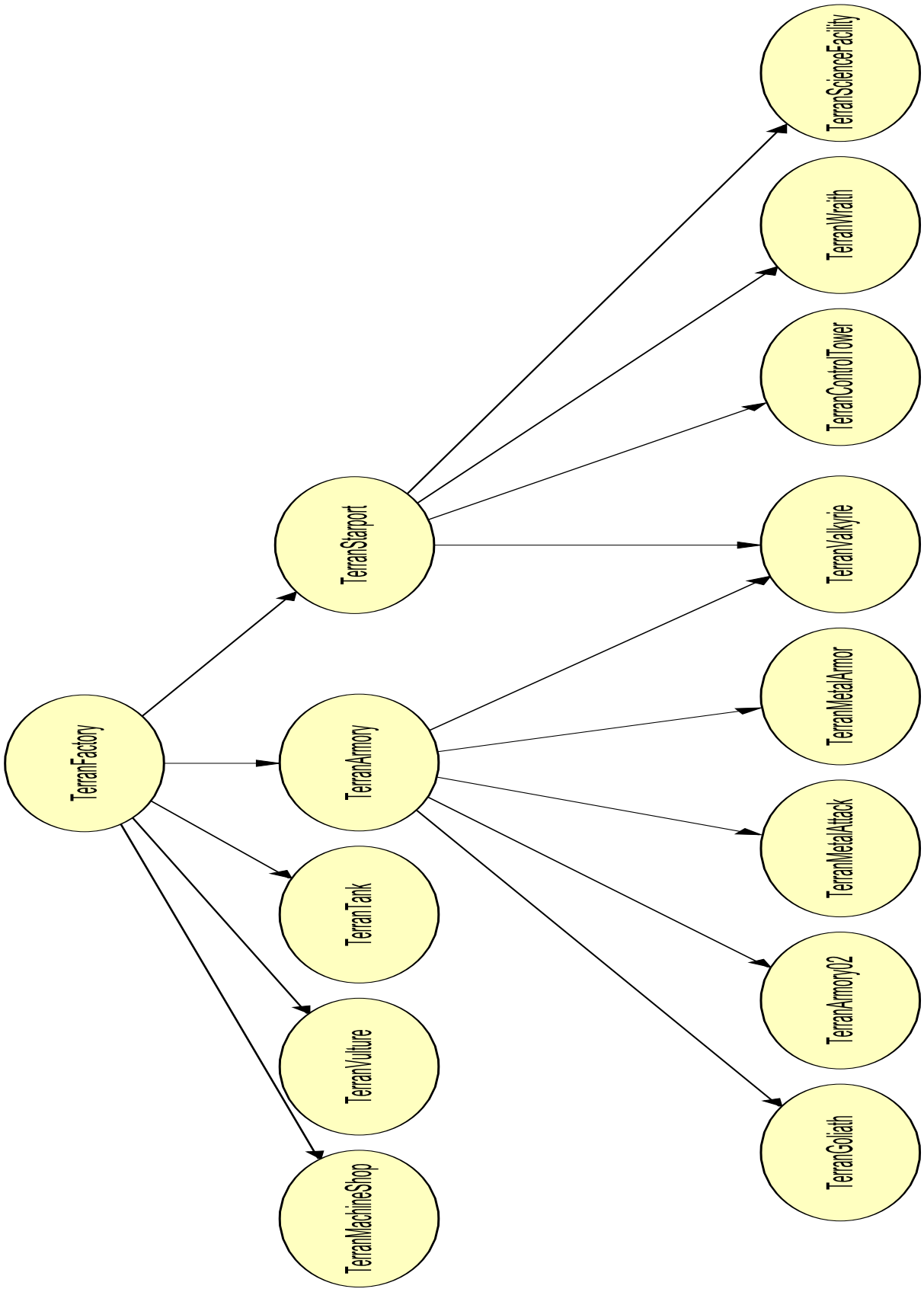


Figure 3.3: A part of the network structure for the StarCraft Terran race

Table 3.1: A subset of StarCraft training data. Creation time values are in game frames

TerranCommandCenter	TerranSupplyDepot	TerranRefinery	TerranRefinery02
1	1398	2806	12830
1	1435	2805	10405
1	1465	2550	22220
1	1395	2565	14405
1	1520	2170	18485
1	1485	2635	15855
1	1638	2470	16492
1	1455	2665	15755
1	1450	5165	14700

Training the Network

StarCraft and most RTS games provide a tool to record game logs into replays that can be re-simulated by the game engine. This replay feature can be used to extract players actions and process them into data to train a BN with. The replays have to be converted from a proprietary format (that contains only user interface actions performed by each player but no game state) into a log of game actions, or simulated in the game engine, to get the actual training data. Several websites, such as GosuGamers.net, TeamLiquid.net and ICCup.com are dedicated to collecting and sharing StarCraft replays. We use the data set provided by Weber et al. [37]. The focus in the data set is on one-versus-one matches and the goal was to gather enough matches to be able to develop an opponent model that is not limited to a single type of player, set of maps or style of gameplay.

Simple preprocessing is needed to transform the game log data into our tabular format required for training the network. The algorithm chosen to be used for learning the network parameters is the EM algorithm. The conditional probability distribution learned for each node is represented with a CPT that has a number of parameters that is exponential in the number of parents of the node. So for example, if a node has three parents each with 80 states and the node also has 80 states then the size of the table will be $80^4 = 4096 \times 10^4$. Parts of the training data are shown in Table 3.1.

Figure 3.4 shows the UML class diagram for the BNManager class, which encapsulates the functionality of the BN. The code for the BN component is open source and available online [22], so any modifications or additions that are needed can easily be implemented and added to the component. After creating an instance of the BNManager class the Bayesian Network is constructed and ready to be used. Connecting the network to a game engine and the game AI is done by using the exposed methods on the BNManager class in the appropriate game loop events or AI decision processes.

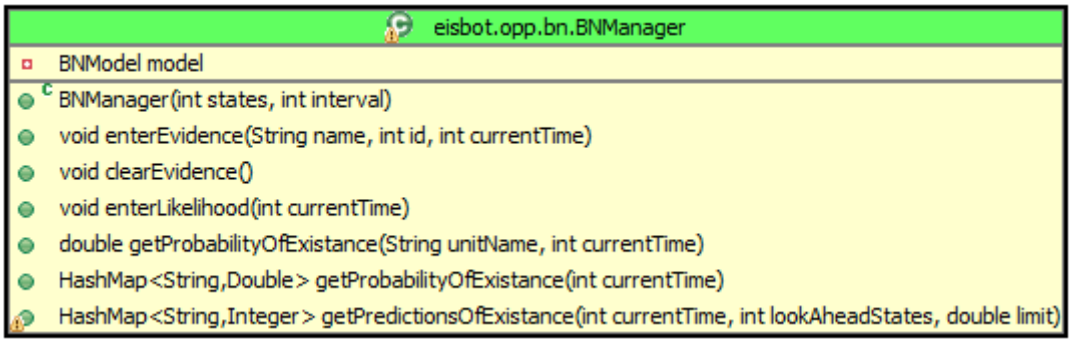


Figure 3.4: UML class diagram for the BN Manager class

3.2.1 Provided Game Logs to Train the Network

As a part of this project we implemented a web crawler and downloaded 8364 game logs from previously played professional StarCraft games. We simulated these game logs in the StarCraft game engine using the BroodWar API and extracted 1.2 TB of data. Although not used, as part of this work, the data may prove useful for future research purposes. It can be downloaded from an online repository [27]. In Table 3.2 the information that was recorded for each game is described, the data is provided in text format.

Table 3.2: Information in provided StarCraft simulation data

Information	Specific timings	Sample timings (2x per sec)
Map	Unit production time	Position of each unit/building
# players	Building construction time	State of each unit/building
Rep ID	Research timings	Hit points of each unit/building
Races	Destruction timings	Status of special abilities
Players	Morph timings	Type and side of each unit
		Resource status for players

3.3 Evaluation

We can evaluate our implementation by setting up an experiment where we use game logs from StarCraft matches (a disjoint set of game logs to the set used to train the network), where we remove information about the creation time of n units or buildings, enter the remaining units and building creation times into the network and return the n units and buildings from the network that have the highest probability of existing at the current time. We can then compare the units and buildings returned from the network with the actual units and buildings removed from the original game log to get a precision value for the

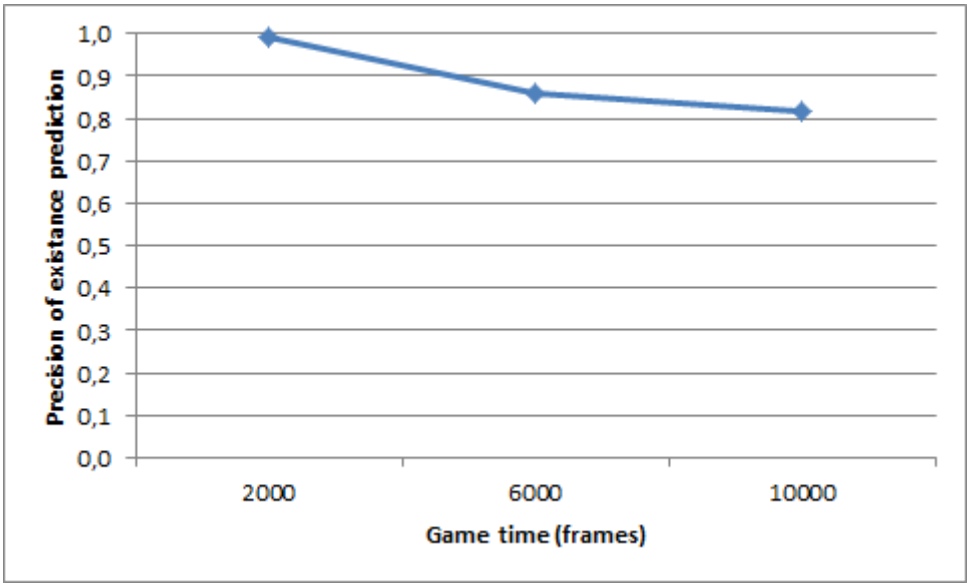


Figure 3.5: Precision of existance prediction for unobserved units and buildings

BN estimations. Figure 3.5 shows the results of our experiment using a subset of the game logs (Terran vs. Protoss matches) from [37] in a ten-fold cross validation setup for game times set at 2,000, 6,000 and 10,000 frames. The results show that using the BN, one can expect high precision rates for estimations about which unobserved opponent units and buildings exist for the first 10,000 frames. The precision values will depend on the origins of the game logs used as well as other factors. The results should only be viewed as indicative of the capabilities of the implemented BN. Our main results are presented in Chapter 5.

Chapter 4

Bayesian Networks in RTS Games

In this chapter we discuss possible applications for a Bayesian Network (BN) constructed from game logs in RTS games. We investigate three uses; in-game commentary assistance, extending case-based reasoning and improving strategy prediction. In summary, we believe that a BN can be used in all of the above suggested applications. Providing information that assist human players in their decision-making, decreasing the noise in the retrieval stage of the case-based reasoning process when opponent information is included in the case representation and making AIs more effective and enjoyable to play against by improving strategy prediction in RTS games.

4.1 In-Game Commentary Assistance

Intelligent commentary assistance software can be used in several ways to assist human players in competitive games. The trend in on-line poker, for example, is to use various poker commentary assistance software to model your opponents and prevent them from being able to model you [33]. In StarCraft, our BN can allow players an overview of likely units and buildings that an opponent has constructed in an ongoing competitive game. Assisting them in their strategic decision making. To show an example of this we embedded our BN in a open source StarCraft bot called EISBot [21] that connects to the StarCraft game engine through the BroodWar API library [23]. Figure 4.1 shows a graphical user interface with information from the BN on the right of the screen. By seeing the probabilities of different units and buildings existing at a given time and for the observed evidience in the on-going game, an expert player can predict the most possible strategies that his opponent is executing and react accordingly. Another possible use for the same BN would be as commentary assistance to spectators of StarCraft tournaments.

Giving the spectators hints as to what are the most likely possible future situations for the game they are watching.

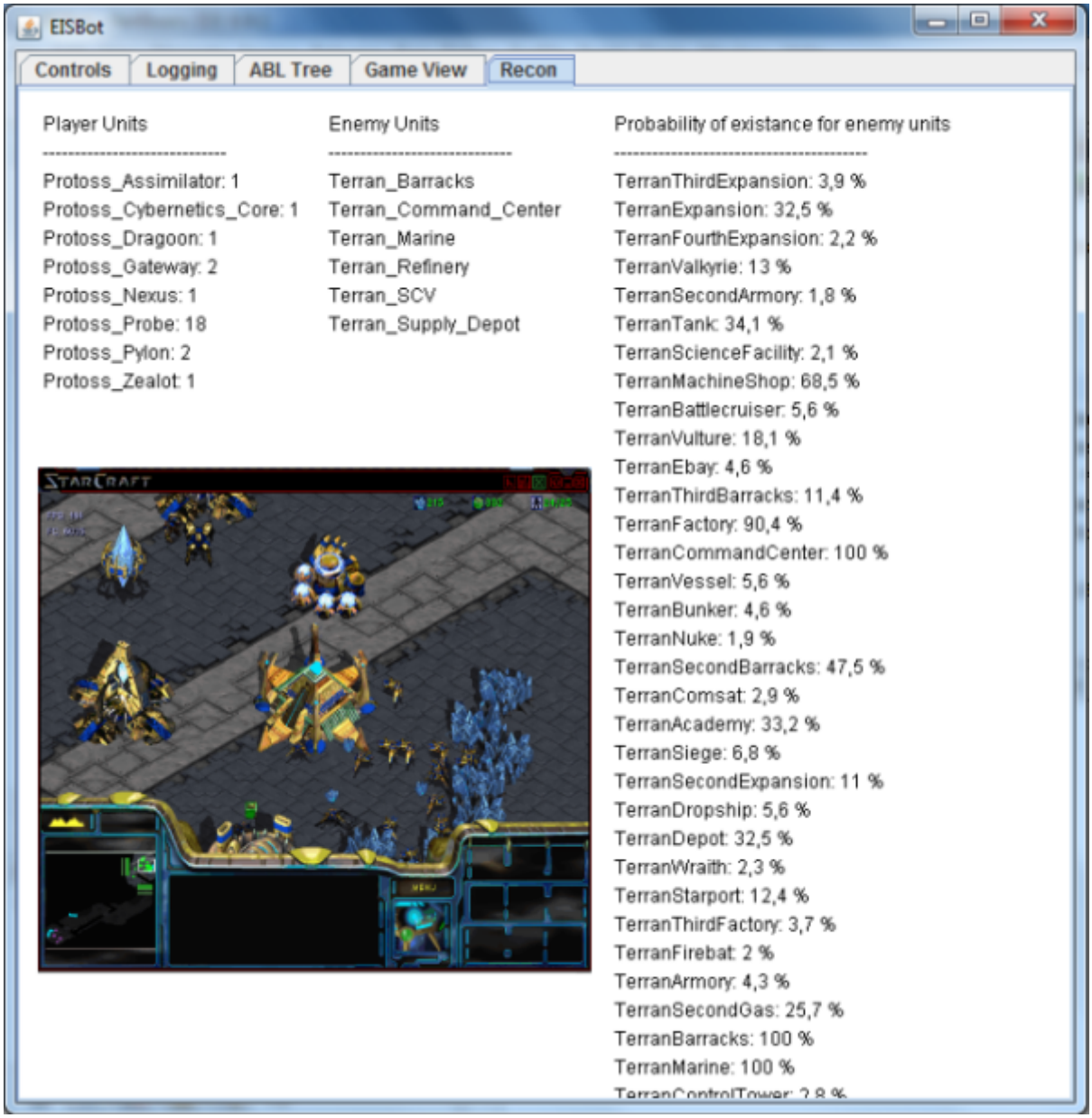


Figure 4.1: A screenshot of our GUI implementation of BNs for commentary assistance in RTS games

4.2 Extending Case-Based Reasoning with Bayesian Networks

In this chapter we will discuss Case-bsed Reasoning techniques, their application in RTS games and how Bayesian networks can possibly be used to increase their effectiveness in imperfect information environments.

4.2.1 Case-Based Reasoning

Case-based reasoning is a problem solving paradigm that in many respects is fundamentally different from other major AI approaches. Instead of relying solely on general knowledge of a problem domain, or making associations along generalized relationships between problem descriptors and conclusions, CBR is able to utilize the specific knowledge of previously experienced, concrete problem situations (cases). A new problem is solved by finding a similar past case, and reusing it in the new problem situation. A second important difference is that CBR also is an approach to incremental, sustained learning, since a new experience is retained each time a problem has been solved, making it immediately available for future problems. The CBR field has grown rapidly over the last few years, as seen by its increased share of papers at major conferences, available commercial tools, and successful applications in daily use.

Excerpted from Aamodt, A., & Plaza, E. (1994).

Case-based reasoning (CBR) techniques revolve around the idea of solving new problems based on the solutions of similar past problems. Popular examples of humans using case based reasoning are:

- An auto mechanic who fixes an engine by recalling another car that exhibited similar symptoms
- A lawyer who advocates a particular outcome in a trial based on legal precedents

Case-based reasoning has been formalized for purposes of computer reasoning as a four-step process [1]:

- **Retrieve** the cases from the case-base whose problem is most similar to the new problem
- **Reuse** the solutions from the retrieved cases to create a proposed solution for the new problem

- **Revise** the proposed solution to take account of the problem differences between the new problem and the problems in the retrieved cases
- **Retain** the new problem and its revised solution as a new case for the case-base if appropriate

The retrieval process is the most relevant part of the CBR cycle, for our purposes, since it is in that part of the cycle that a Bayesian Network could potentially be used to increase the effectiveness of case-based reasoning techniques for RTS game AI. The CBR cycle can be seen in Figure 4.2.

4.2.2 The Retrieval Process

The purpose of the retrieval process is to retrieve the most effective/similar case or cases for the current situation from a case library. Typically it consists of nearest neighbour approaches using for example the euclidian distance between all, or a subset of all case variables as a distance metric. Retrieving a case starts with a (possibly partial) problem description and ends when a best matching case has been found. The subtasks involve:

- identifying a set of relevant problem descriptors;
- matching the case and returning a set of sufficiently similar cases (given a similarity threshold)
- selecting the best case from the set of cases returned.

Nearest neighbor approaches utilize cases with a feature vector representation. Given a query, nearest neighbor retrieves the nearest case within the feature space. Similarity between cases is computed using a distance function where the general form for computing

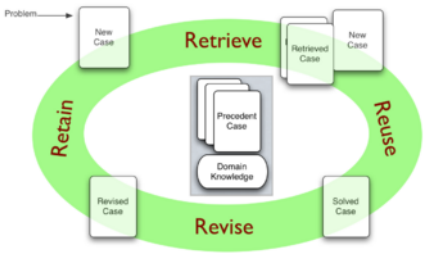


Figure 4.2: The Case-based reasoning cycle

the distance between a query q, and a case c, is defined by the L_p norm:

$$d(q,c) = \left(\sum_{j=1}^{n} |q_j - c_j|^p \right)^{1/p} \tag{4.1}$$

where q_j and c_j are features and n is the number of features in the case description. Common L_p norms are L_1, the Manhattan distance, and L_2, the Euclidean distance.

4.2.3 Applications in RTS Games

CBR is one of the most popular technique used in connection with RTS game AI research [3, 35, 28]. Almost all the work related to case based reasoning in RTS games report success rates and methods for perfect information environments, if information about the opponent is included in the case representation . In fact, most work does not even include any information about the opponent in the case representations, since this increases the noise when performing retrieval in imperfect information environments [36].

In general, when the case base is constructed from RTS game logs, we have perfect information about the opponent. We can therefore use any information we like in the case representation of the cases to be stored in our case library. An example of this kind of information would be to include variables like number of existing buildings of type X or unit type Y for the opponent. At retrieval time we only have partial information about the opponent. If this possibly missing information was included in the case representations, retrieving the most similar cases for the situation from the case library becomes much harder from a similarity measurement point of view. On the other hand, not including information about the opponent in case representations could result in retrieval and adoption of a case that is far from being the best case in the case library for the situation at hand. In [36] it is stated that increasing the number of features that describe the opponent increases noise when performing retrieval. Using a BN to estimate the values of the opponent features could decrease this noise and allow for many more opponent features to be included in the case representations used.

In summary, when a RTS game AI that is using CBR related techniques is faced with the problem of retrieving the most effective case for the current situation from a case library that includes opponent information in case representations and the AI is operating in an imperfect information environment, we believe that using BNs for state estimations can increase the effectiveness of the AI and the reported success rates. The general idea is that including estimations from a BN for unobserved variables that are included in case representations will make case based reasoning related techniques more effective by de-

creasing the noise in similarity computations during the retrieval stage of the CBR cycle. We corresponded with the authors of Darmok 2 (D2), an on-line case-based planning framework, to incorporate a BN into D2. The initial design is ready, but not implemented as it is outside the scope of this project.

4.3 Strategy Prediction

Strategy prediction in RTS games like StarCraft can be described as the process of identifying the order in which opponents will expand their tech tree. At the beginning of a StarCraft game, each player decides a build order (strategy) to use based on the opponents race, map features and other preferences. Build orders are sequences of actions that are executed and expand the tech tree of the player in a particular order. As a StarCraft game progresses, players need to adapt their strategy to counter the strategy being executed by the opponent. The sooner a player can predict the strategy being executed by the opponent, the sooner he can react and adapt his strategy in the best possible manner. Not only does strategy prediction allow the AI to counteract with known effective counter strategies, making it more competetive, but it also opens up the possibility of more interesting gameplay for human opponent matches. As an example, we can consider a RTS game AI that is capable of strategy prediction an has a set of strategies it can execute along with data about how effective they are against commonly observed strategies. This would allow the AI to pick a counter strategy to the strategy being exectuted by the human opponent based on the level of challenge desired, adapt poorly for week players and effectively for strong players. If the strategy prediction is extended to work in imperfect information environments, handling the partial observability in RTS games, a human player can try to hide units and buildings and influence the information available to the AI opponent. This would hinder the AI in its ability to make precise predictions and allow the human player to use the essential game elements of "bluffing" and "surprise". In summary, predicting strategies of opponents effectively is an essential part of a competetive RTS game AI and also has the potential to increase the entertainment value of playing against the computer in RTS games.

In the next chapter we examine in more detail how BNs can be used to improve strategy prediction in StarCraft.

Chapter 5

Case-Study: Improving Strategy Prediction in StarCraft

In this chapter we will show how a Bayesian Network (BN) can be used to improve strategy prediction under uncertainty in the RTS game StarCraft. We show this by comparing classification results for the task of predicting strategies in StarCraft, with and without estimations from a BN for uncertain (unobserved) features. By training the network on a large collection of game logs we can use probabilistic inference algorithms to infer the probabilities of unobserved units and buildings, given observed evidence for a particular game, and use this information to augment the feature vectors used in the strategy prediction with the corresponding estimations from the network. As we will show, this greatly increases the effectiveness of strategy prediction in imperfect information environments.

5.1 Background

One of the main challenges for game designers is creating adaptive AI opponents that react to player actions. For RTS games, an intelligent AI opponent should be capable of responding to a player's strategy with a counter-strategy. For this to occur, the AI opponent must first identify the players strategy. RTS games are imperfect information environments, where the AI has partial information about the game state. In order to be adaptive, an RTS game AI therefore has to recognize strategies under uncertainty before effective counter-strategies can be executed.

Weber et al. [37] present a data mining approach to strategy prediction, where expert gameplay was learned by applying machine learning techniques to a large collection of game logs. This approach enables domain independent supervised learning algorithms to acquire StarCraft domain knowledge and allows for the classification of feature vectors representing the game state into different strategies before they are executed. The approach was evaluated in perfect and imperfect information environments and the results showed that it had higher predictive capabilities then both state lattice and exact rule based classifiers. We improve upon their approach by augmenting the feature vectors with estimations from our BN for uncertain features, before classification.

5.2 Game Logs

We use the data set extracted from StarCraft game logs by Weber et al. [37]. This data will be used to train both the classifiers responsible for strategy prediction and the BNs used to infer the probabilities of unobserved units and buildings.

5.2.1 Encoding Replays as Feature Vectors

Weber's approach encodes the game logs as feature vectors, where each feature describes when a unit or building was first produced (see Table 5.1). The goal of this representation is to capture the timing aspects of a players strategic decisions in a game, enabling the prediction of distinct strategies. Each feature vector contains temporal features, in the unit of frames, that record when a player expands different branches of the tech tree and represents a single players actions for an entire game.
Formally, the feature vector for a player P is defined as follows:

$$F(x) = \begin{cases} t & : \text{time when x is first produced by P} \\ 0 & : \text{x was not (yet) produced by P} \end{cases} \tag{5.1}$$

where x is a unit type, building type or unit upgrade.

5.2.2 Labeling Replays

The game logs are labeled with a strategy using rules based on analysis of expert game play. Each rule set was designed to capture middle game strategies and different rule sets are used for each race in StarCraft. The rule sets label logs with strategies based on the

Table 5.1: A subset of an example feature vector

Feature	Frame nr.
Terran Command Center	1
Terran Supply Depot	1398
Terran Refinery	2806
Terran Barracks	2012
Terran Marine	3264
Terran Medic	0
Terran Firebat	0
Terran Bunker	6212
Terran Academy	14170
Terran Stem	0
Terran Comsat Station	15992
Terran Engineering Bay	7218

order in which building types are produced. Six strategies were defined for each race, and if a game does not fit into any of them, it is labeled as unknown. The rule set for the Terran race is shown below. The six strategies are Standard, Vulture Harrass, Siege Harrass, Fast Dropship, Bio and Unknown.

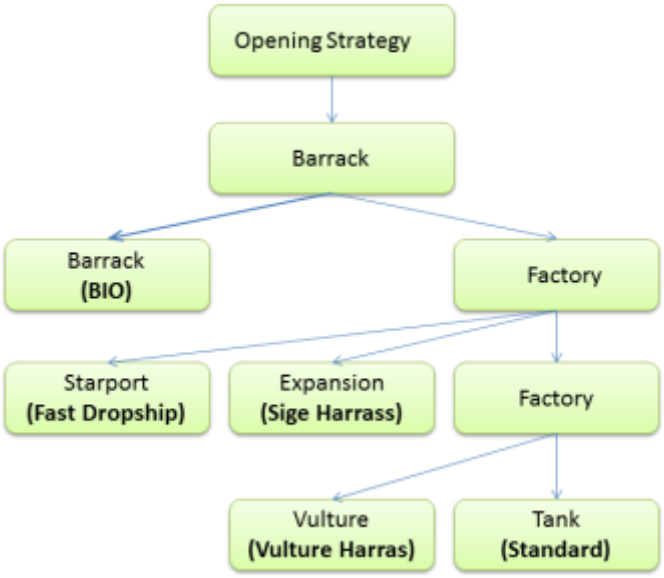


Figure 5.1: Rule set for labelling Terran strategies. The tree is traversed by selecting the first building produced in the child nodes.

5.2.3 Representing Cases for Training the Bayesian Network

To learn the parameters of the BN described, we structure the feature vectors extracted according to the specifications described in Chapter 3, where each game log represents a single case used for learning in the network.

5.2.4 Overview of the Data Used

Using the game logs gathered we have prepared both labeled and unlabeled data sets to be used by supervised learning methods for strategy prediction and by unsupervised learning methods to infer the probabilities of the existence of unobserved units and buildings.

5.3 Strategy Prediction Under Uncertainty

5.3.1 Methodology Used for Classification

We represent strategy prediction in StarCraft as a multi-class classification problem. The classification algorithms used were:

- J48 - C4.5 decision tree [31]
- NNge - Non-nested generalized exemplars [20]

We used the implementations of these algorithms provided in the Weka machine learning library [40], using the default settings and performing ten-fold cross validation for all our experiments.

The algorithms were evaluated at different time steps throughout each game, where different time steps where simulated by setting all features with a value greater then the current game time to 0. This transformation was applied to the training data and test data. To test the effects of imperfect information on the classification algorithms, features in the feature vector to be classified from the test set where randomly set to 0, based on a missing feature probability. This follows the exact methodology used by Weber et al. [37] and simulates a player that is unable to scout some part of the environment in StarCraft. Setting unknown entries to 0 instead of marking them as unknown may potentially adversely affect some classifiers that treat it as any other numerical value. We did, however,

not experiment with alternative ways of representing unknown values as for comparison purposes we chose to follow the exact representation used by Weber et al. [37].

5.3.2 Methodology Used for Bayesian Feature Estimations

For the following discussion, we refer to our implementation of the BN described in Chapter 3, and focus specifically on the additions and methodology needed for our strategy prediction case study. To estimate the time value for buildings and units that correspond to features that where set to 0, as a part of the classification methodology to simulate the fog of war in RTS games, we have to map the probabilities in the BN into time estimations for these features. We do this by using the expected value of the random variables in the network, after propagating the evidence for the current game, that correspond to missing features in the feature vector to be classified.

$$E[X] = \sum x_i \times P(X = x_i) \tag{5.2}$$

Since the variables in our network are discrete, the number of states and interval size for the states in the variables will effect the accuracy of the estimated time values. We did not fine tune these parameters for our evaluations, since our main focus was on proof of concept. For the same reason, we begin by only putting time estimations for features, in the feature vector to be classified, that we know where set to 0 to simulate imperfect information about the environment. The alternative is to get the probability that the variable in the network that corresponds to the feature in question represents a unit or building that has been constructed at the given game time, and then decide upon a threshold value for setting an estimation for that feature in the feature vector. Evaluation of this approach is discussed specifically in Section 5.4.4.

For each classification where Bayesian feature estimations are used, we perform the following steps in the network:

1. Clear all potentials and evidence in the network to start a new inference cycle
2. Enter as evidence into the network the currently non-blank features in the test vector to be classified
3. Propagate the supplied evidence and infer the new marginal distributions for variables in the network
4. Calculate the expected values for variables corresponding to blank features in the test vector and put the inferred expected values into the test vector to be classified

Table 5.2: Precision of strategy prediction for games at 5 minutes and 10 minutes

	5 minutes			10 minutes		
	NNge	J48	Boosting	NNge	J48	Boosting
Protoss vs. Protoss	0.50	0.44	0.49	0.81	0.82	0.85
Protoss vs. Terran	0.69	0.63	0.56	0.89	0.92	0.91
Protoss vs. Zerg	0.63	0.63	0.66	0.85	0.88	0.88
Terran vs. Protoss	0.76	0.66	0.61	0.82	0.81	0.92
Terran vs. Terran	0.82	0.76	0.54	0.85	0.82	0.90
Terran vs. Zerg	0.92	0.89	0.90	0.95	0.92	0.89
Zerg vs. Protoss	0.54	0.56	0.60	0.84	0.85	0.83
Zerg vs. Terran	0.53	0.50	0.46	0.87	0.91	0.84
Zerg vs. Zerg	0.83	0.83	0.85	0.94	0.96	0.94

Since we are typically using hundreds of test vectors, ten-fold cross validation and both multiple data sets and classifiers in our case-study, this process is repeated many hundred thousand times in our evaluations.

5.4 Evaluation

To show that BN can be used to improve the accuracy of strategy prediction under uncertainty in RTS games, we compare the precision of the classification algorithms versus game time and ratio of missing features, for feature vectors with and without estimations from a BN. Effectively comparing the method used in [37], with our proposed method using BNs.

5.4.1 Replication of Previously Reported Results

To verify our implementation and experimental setup we begin by replicating the results from [37]. Table 5.2 shows the precision for strategy prediction with perfect information for games at 5,000 and 10,000 frames. We use frames instead of minutes as we do not know the exact conversion rate used in [37] and to eliminate the possibility of errors resulting from game logs being played at different frame rates per second (this is a public setting in StarCraft). Figure 5.2 shows the results for strategy prediction, with perfect information, of Protoss versus Terran games for NNge, Boosting and J48 from 0 to 12,000 frames. Figure 5.3 shows precision in an imperfect information environment for ratios of missing attributes from 0 to 0.5.

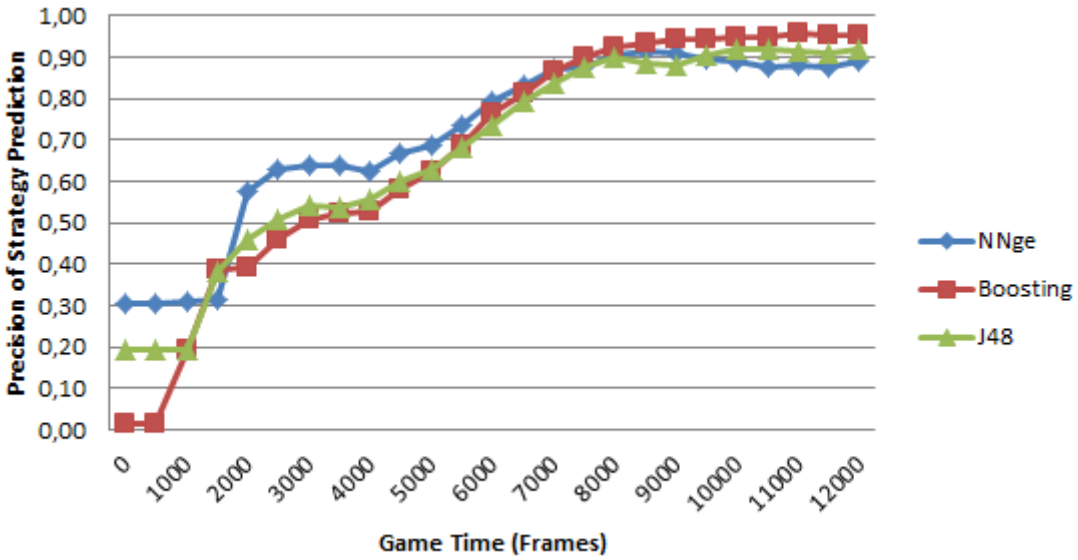


Figure 5.2: Precision of strategy prediction for Protoss vs. Terran in perfect information environments

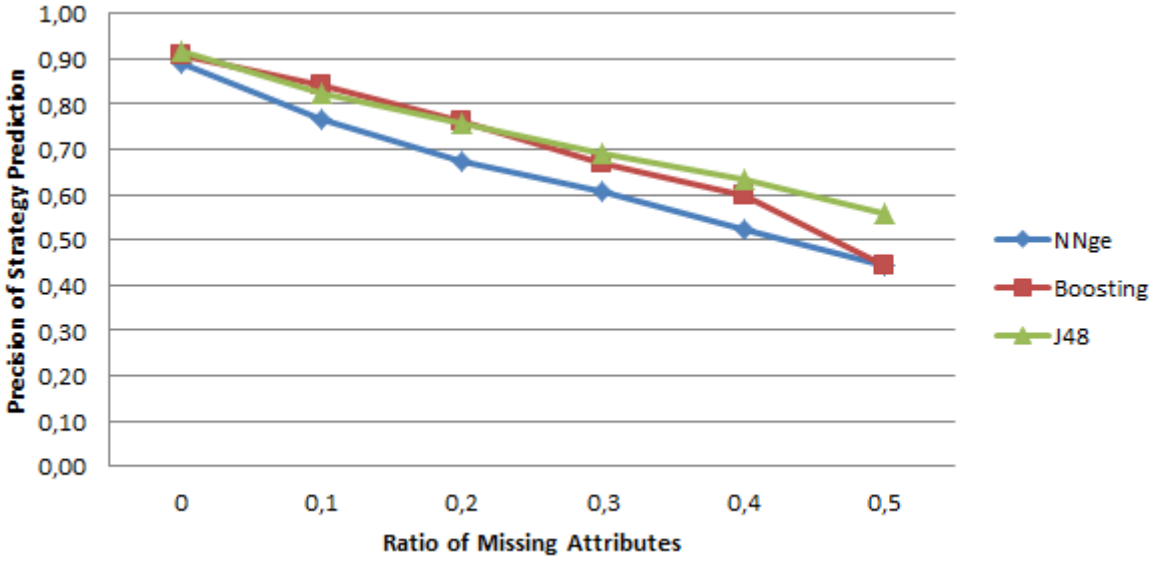


Figure 5.3: Precision of strategy prediction in an imperfect information environment. The missing attribute ratio specifies the probability that an attribute will be set to 0. The results are for Protoss vs. Terran games at 10,000 frames.

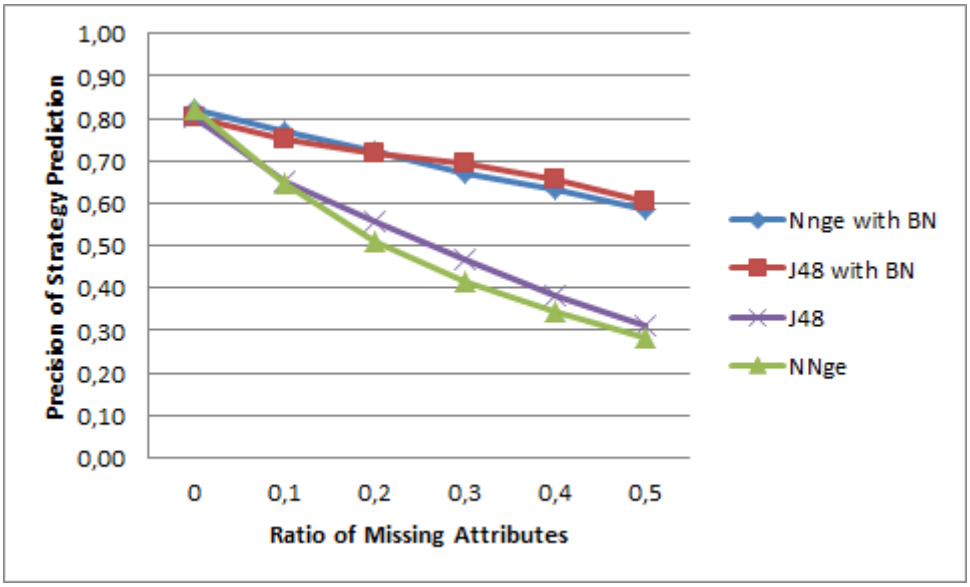


Figure 5.4: **Known Opponent:** Precision results when using and not using a BN for different values of the ratio of missing attributes. The results are for Terran vs. Protoss games at 10,000 frames.

We see that precision decreases linearly with increased ratio of missing attributes for imperfect information environments, as expected. This result replicates that of Weber et al. [37] and serves the purpose of verifying that our implementation is consistant with theirs. This is important as we use their results as our baseline in further experiments, where we use a BN to predict values for some of the missing attributes.

5.4.2 Predicting a Known Opponent

In this section we show results for predicting a known opponent, using the same data set of 1139 previously played StarCraft games from Terran vs. Protoss matches to train both our BN and classifiers. This is justifiable since other opponent AI bots are often available to train on. Many of these bots use a combination of predefined strategies that can be learned from playing enough matches against the bot before a competition match. Although the simulation for a known opponent is idealistic, there is no chance of overfitting to the training data and it has a valid appeal to a real task, as described above. We use J48 and NNge for our comparison. Figure 5.4 shows the precision of strategy prediction for both classifiers with ratio of missing attributes from 0.0 to 0.5, with and without using estimations from the BN. In both cases, the precision rate clearly improves when using BN estimations and the precision values decreases at a slower rate, with increasing ratio of missing attributes, then when BN estimations are not used.

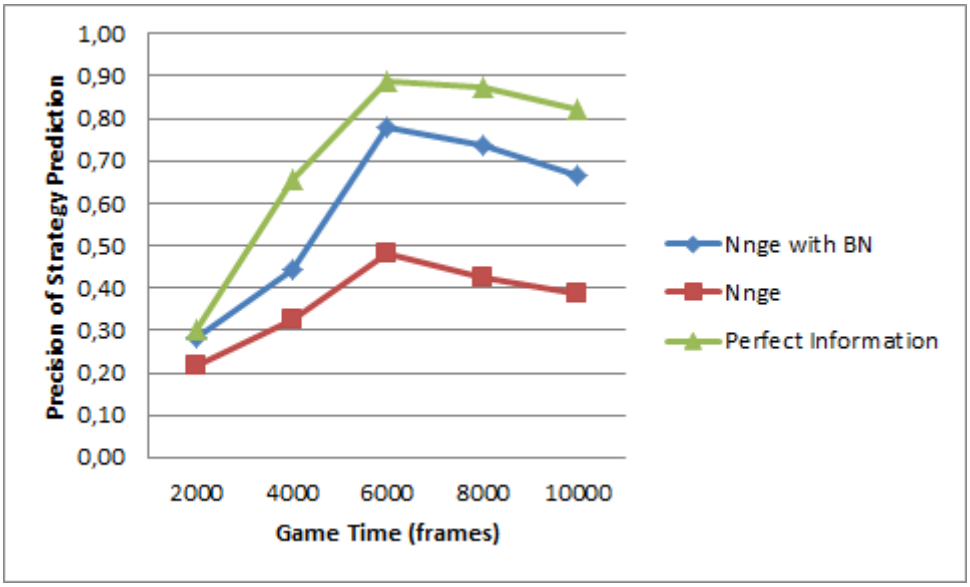


Figure 5.5: **Known Opponent:** Comparison of results for NNge precision with and without using a BN when the ratio of missing attributes equals 0.3

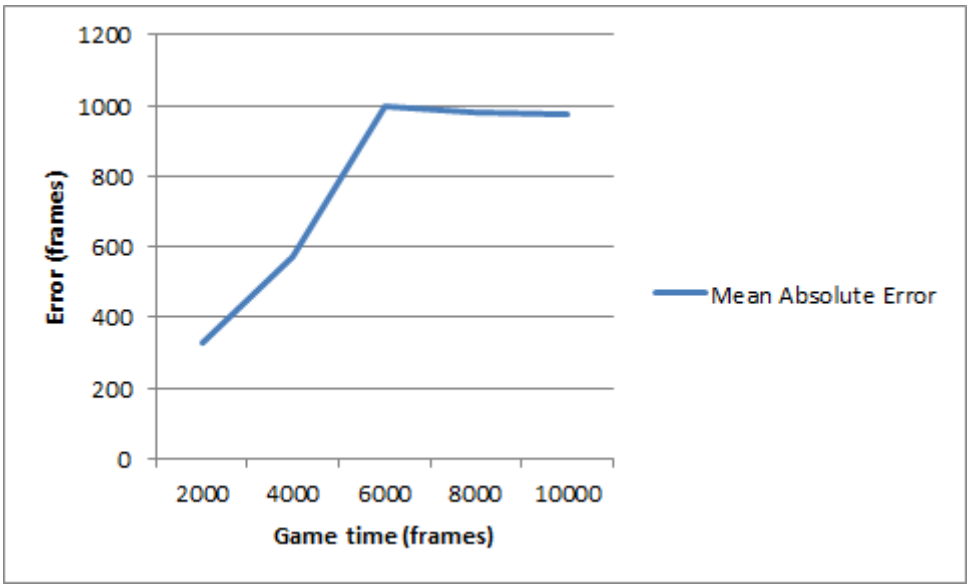


Figure 5.6: **Known Opponent:** Mean absolute error for BN estimations used to get the results for NNge precision with a BN when the ratio of missing attributes equals 0.3 (Figure 5.5)

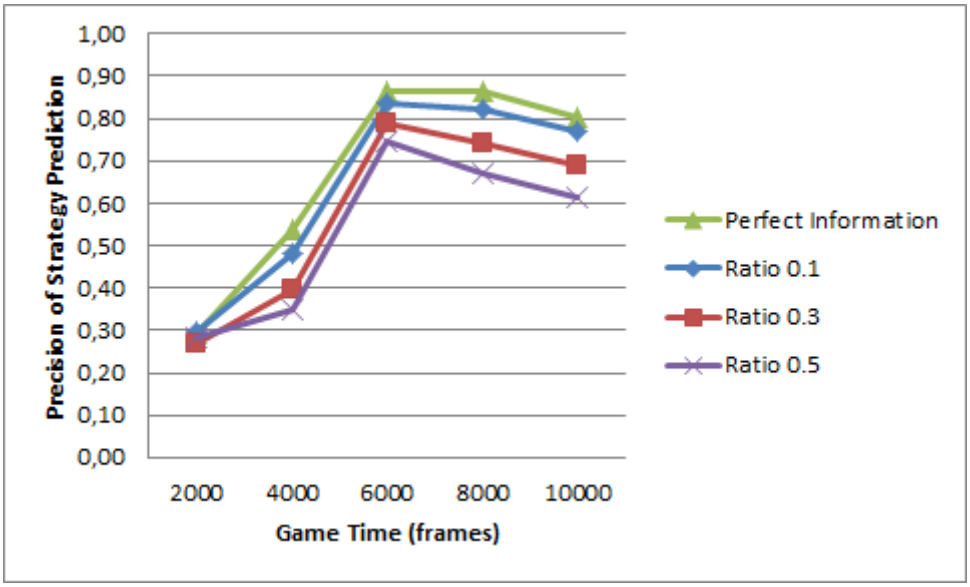


Figure 5.7: **Known Opponent:** Precision results **using a BN** for J48 and different values of the ratio of missing attributes

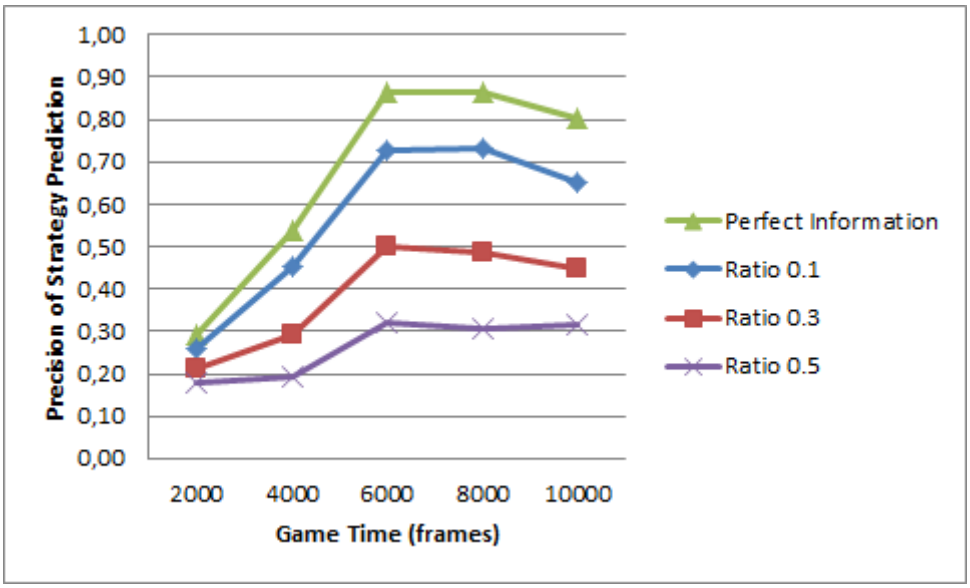


Figure 5.8: **Known Opponent:** Precision results **without using a BN** for J48 and different values of the ratio of missing attributes

It is intuitive to look at how the difference between using and not using a BN evolves over time. Figure 5.5 shows the evolution of strategy prediction precision from 2,000 frames to 10,000 frames. The precision of strategy prediction increases linearly with increasing information about the opponent, up to around 6,000 frames. After 6,000 frames it starts to decrease slightly. This is because the feature vector used to classify includes a greater total number of features and consequently has more blank features. Using the BN increases the precision of strategy prediction for all time points (frame counts), with a maximum difference of 0.3 at 6,000 frames. This corresponds to a 63% increase in the precision value. It is also clear from this that the best time to predict strategies is at around 6,000 frames. Figure 5.6 shows the Mean absolute error for BN estimations used with the same experimental setup as was used to get the precision results in Figure 5.5. The state interval size used for the discrete nodes in the network was 480 frames, so the estimations from the network typically fall within two states from the correct value. As mentioned before, this was not fine tuned.

Figure 5.7 shows the evolution of precision of strategy prediction over time, for different values of missing attributes and using estimations from the BN. We see how the trend on results with a BN is persisted with increasing number of missing attributes in the feature vector to classify. The trend is less obvious in Figure 5.8 from using ratio = 0.1 to ratio = 0.5. Comparing Figure 5.7 and Figure 5.8, we see that using a BN improves the precision of strategy prediction for both all time and ratio of missing attribute values.

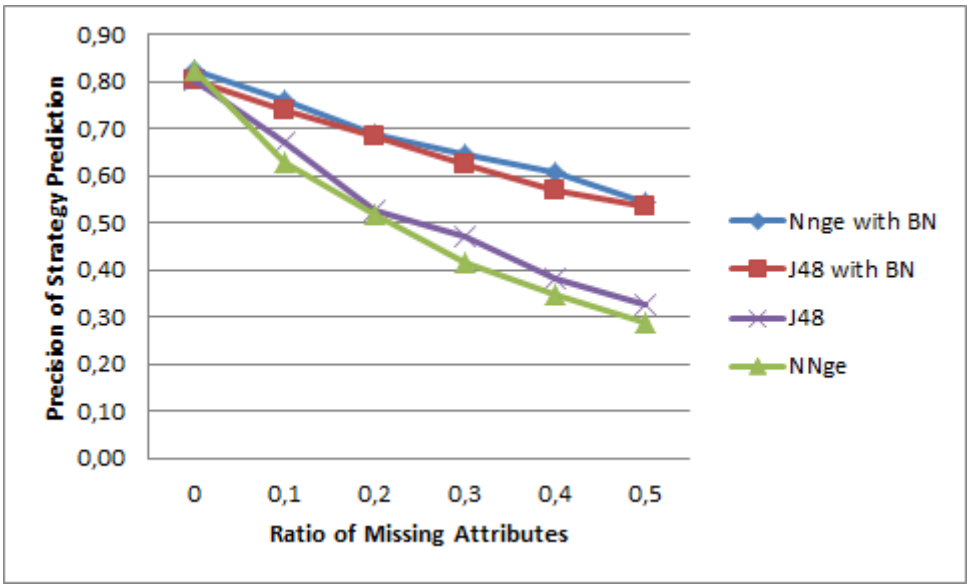


Figure 5.9: **Unknown Opponent:** Precision results when using and not using a BN for different values of the ratio of missing attributes. The results are for Terran vs. Protoss games at 10,000 frames.

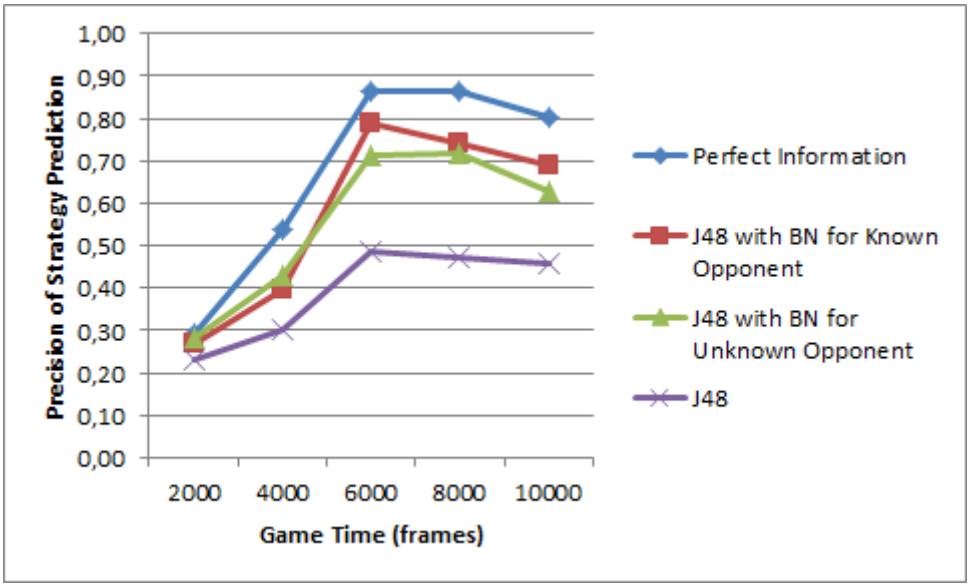


Figure 5.10: **Unknown Opponent:** Comparison of J48 results for simulations of a known and unknown opponent with ratio of missing attributes set at 0.3

5.4.3 Predicting an Unknown Opponent

In this section we run selected experiments as in the previous section, except that now we train the BN on a different data set then used to train the classifiers. We use the Terran vs. Terran and Terran vs. Zerg data sets to train the Bayesian Network and as before we use the Terran vs. Protoss data set for classification. This simulates an opponent that the AI has never played against before and is therefore less likely to have seen the exact build order used. If we compare the results in Figure 5.9 with those for simulating a known opponent (Figure 5.4) we see that the precision of strategy prediction decreases only slightly in the case of an unknown opponent. Figure 5.10 compares the results over time and shows the same slight decrease in precision values. In general the interpretations for the results of simulating a known opponent are all valid for the simulations for an unknown opponent.

5.4.4 Selecting Features to Estimate

In previous experiments, we used time estimations from the BN to replace the exact features that where set to zero when transforming the test set to simulate an imperfect information environment. In this section we generalize and do not use knowledge of which features where set to zero. This means that the method is suited to be used as is for in-game AI, given unit creation times. The problem of selecting which features to use estimations for introduces both the possibility of wrong estimations, where a feature is

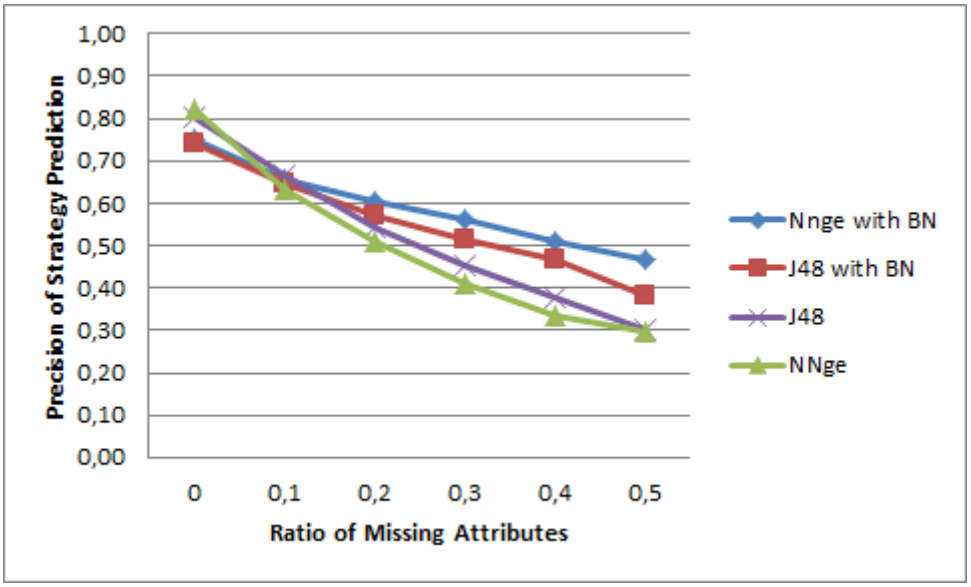


Figure 5.11: **Known Opponent, Selected Estimations:** Precision results when using and not using a BN for different values of the ratio of missing attributes. The results are for Terran vs. Protoss games at 10,000 frames.

selected for estimation that was not set to zero when transforming the test set, and missed estimations, where a feature where set to zero when transforming the test set but not selected for estimations. As described in Chapter 3, we can use the BN to get the probabilities of a unit or building corresponding to a feature in the feature vector existing at the particular time. We can then decide a threshold value for which to put estimations from the BN into the feature vector. In our evaluation we used 0.8 as the threshold value. This value was selected after iterating over different values for Terran vs. Protoss games at 10.000 frames. It is not fine tuned and the value that gives the best results will generally vary depending on the data used and particular game time. For each feature vector, the features that correspond to units or buildings in the network, that have a higher probability then the threshold value of existing, are estimated.

Using expert domain knowledge about StarCraft and trial and error experience of the selection process, it is in our opinion possible to increase the precision of strategy prediction considerably from our results. For example, not using estimations for units or buildings that are known to give a bad estimate or given the time at which the prediction is to be made, have predifined rules about what features to use estimations for. It was our choice not to use any specific rule based domain information in our evaluations. We can see from Figure 5.11 that adding the complexity of selecting which features to use estimations for decreases the precision of the strategy prediction when using BN. We can also see that for the ratio of missing attributes higher then 0.1 it is better to use a BN for strategy prediction

then not to use it. With a maximum difference of 0.17 at a ratio of 0.5 missing attributes for NNge, a 56% increase in precision value.

Chapter 6

Conclusions

We have proposed, implemented and tested a Bayesian Network (BN) to represent uncertain knowledge for the real-time strategy game StarCraft. Using game logs to find prior probabilities and in-game observations as evidence, we show that a BN is an appropriate model to represent uncertain knowledge in RTS games. By constructing the network to mimic the structure of the game-specific technology tree, we have found that useful probabilistic inferences can be made about the RTS domain. This approach also results in BNs that are easy to construct and intuitive to work with. Our evaluations of using the BN to improve strategy prediction showed an increase in the prediction precision for almost all combinations of the ratio of missing attributes and different game times used in our experimental setup. The precision of strategy prediction generally decreased linearly with increased ratio of missing attributes, but at a slower rate than when not using a BN. The increase in precision values was up to 56% over previously published work [37]. This clearly shows that BNs can be used to improve other AI techniques and that they are a promising approach to represent and reason with uncertainty in RTS games.

There are many ways in which this work can be refined and extended. We limit our presentation to the Terran race in StarCraft and the first 10,000 game frames (around 10 minutes of game time). This can be extended to capture full StarCraft games, using other races or tech trees from other RTS games, including more units and buildings in the network and using more game logs to train on. Although not necessary for our evaluations, an effective method to estimate creation times for observed units is needed. Domain knowledge can be incorporated into both the BN and the techniques used for classification to get higher precision values for strategy prediction in different situations. Investigating related approaches such as Influence Diagrams and Tree Augmented Naive Bayes Classifiers for the RTS game domain is an interesting direction for future research and so is extending

Case-based Reasoning techniques with BNs, as described in Section 4.2. Finally, a Brood War Bayesian Network (BWBN) library similar to the Terrain Analyzer (BWTA) library [24], that abstracts away most of the details of constructing and updating the BN and allows game AI developers to use BNs in their bots could be created.

Bibliography

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] G. Agnarsson and R. Greenlaw. *Graph Theory: Modeling, Applications, and Algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[3] David Aha, Matthew Molineaux, and Marc Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In Héctor Munoz-Avila and Francesco Ricci, editors, *Case-Based Reasoning Research and Development*, volume 3620 of *Lecture Notes in Computer Science*, pages 5–20. Springer Berlin / Heidelberg, 2005.

[4] David W. Albrecht, Ingrid Zukerman, and An E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8(1):5–47, 1998.

[5] M. Buro. Real-time strategy games: A new ai research challenge. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 485–486, 2003.

[6] M. Buro. Call for ai research in rts games. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, pages 139–142, 2004.

[7] N. M. Dempster, A. P. Maximum likelihood from incomplete data via the em algorithm. *The Royal Statistical Society*, 39(1):1–38, 1977.

[8] E. Dereszynski, J. Hostetler, A. Fern, T. Dietterich, T.T. Hoang, and M. Udarbe. Learning probabilistic behavior models in real-time strategy games. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[9] C. Fairclough, M. Fagan, B. Mac Namee, and P. Cunningham. Research directions for ai in computer games. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 333–344, 2001.

[10] David Ferrucci. Build watson: an overview of deepqa for the jeopardy! challenge. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 1–2, 2010.

[11] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2):131–163, 1997.

[12] H. Guo and W. Hsu. A survey of algorithms for real-time bayesian network inference. In *AAAI/KDD/UAI02 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*. Edmonton, Canada, July 2002.

[13] J. Hagelbäck and S.J. Johansson. Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 631–638, 2008.

[14] J.L. Hsieh and C.T. Sun. Building a player strategy model by analyzing replays of real-time strategy games. In *IEEE International Joint Conference on Neural Networks, IJCNN 2008.(IEEE World Congress on Computational Intelligence).*, pages 3106–3111, 2008.

[15] F. V. Jensen, S. Lauritzen, and K. Olesen. Bayesian updating in causal probabilistic networks by local computation. In *Computational Statistics Quarterly*, chapter 4, pages 269–282. Citeseer, 1990.

[16] M. D Jensen. Minirts: Strategy identification using bayesian grid models theme. Master's thesis, Aalborg University, Aalborg, Denmark, 2008.

[17] F. Kabanza, P. Bellefeuille, F. Bisson, A.R. Benaskeur, and H. Irandoust. Opponent behaviour recognition for real-time strategy games. In *AAAI Workshops*, 2010.

[18] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, Cambridge, Massachusetts, 2009.

[19] S.L. Lauritzen and D.J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988.

[20] B. Martin. *Instance-based learning: nearest neighbour with generalisation*. Masters thesis, University of Waikato, Hamilton, New Zealand, 1995.

[21] J. McCoy and M. Mateas. An integrated agent for playing real-time strategy games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 1313–1318, 2008.

[22] Bayesian network code repository. "http://code.google.com/p/sc-bw-bn/", n.d. [Online; accessed 10-January-2012].

[23] Brood war api. "http://code.google.com/p/bwapi/", n.d. [Online; accessed 17-Desember-2011].

[24] Brood war terrain analyzer api. "http://code.google.com/p/bwta/", n.d. [Online; accessed 17-Desember-2011].

[25] Brood war unit types. "http://code.google.com/p/bwapi/wiki/UnitTypes", n.d. [Online; accessed 17-Desember-2011].

[26] Starcraft game logs (ucsc). "http://eis.ucsc.edu/sites/default/files/", n.d. [Online; accessed 10-January-2012].

[27] Starcraft simulation data. "http://code.google.com/p/sc-bw-bn/", n.d. [Online; accessed 10-January-2012].

[28] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram. On-line case-based planning. *Computational Intelligence*, 26(1):84–119, 2010.

[29] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

[30] M. Ponsen, P. Spronck, H. Munoz-Avila, and D.W. Aha. Knowledge acquisition for adaptive game ai. *Science of Computer Programming*, 67(1):59–75, 2007.

[31] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.

[32] F. Schadd, S. Bakkes, and P. Spronck. Opponent modeling in real-time strategy games. In *8th International Conference on Intelligent Games and Simulation (GAME-ON 2007)*, pages 61–68, 2007.

[33] E. Schuijtvlot. Application of ai in poker. Master's thesis, VU University Amsterdam, De Boelelaan, HV Amsterdam, 2011.

[34] G. Synnaeve and P. Bessiere. A bayesian model for opening prediction in rts games with application to starcraft. In *Proceedings of 2011 IEEE CIG*, Seoul, South Korea, September 2011.

[35] B.G. Weber and M. Mateas. Case-based reasoning for build order in real-time strategy games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference, AAAI Press*, pages 106–111, 2009.

[36] B.G. Weber and M. Mateas. Conceptual neighborhoods for retrieval in case-based reasoning. In Lorraine McGinty and David Wilson, editors, *Case-Based Reasoning Research and Development*, volume 5650 of *Lecture Notes in Computer Science*, pages 343–357. Springer Berlin / Heidelberg, 2009.

[37] B.G. Weber and M. Mateas. A data mining approach to strategy prediction. In *In Proceedings of Computational Intelligence and Games, CIG 2009*, pages 140–147. IEEE, 2009.

[38] B.G. Weber, M. Mateas, and A. Jhala. Building human-level ai for real-time strategy games. In *2011 AAAI Fall Symposium Series*, 2011.

[39] S. Wintermute, J. Xu, and J.E. Laird. Sorts: A human-level approach to real-time strategy ai. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*, volume 1001, pages 55–60, 2007.

[40] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Mateo, California, 2005.

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is