

Studiengang: Softwaretechnik

Prüfer: Prof. Dr. P. Levi

Betreuer: Dipl. Inf. T. Buchheim

begonnen am: 15. Mai 2003

beendet am: 11. November 2003

CR-Klassifikation: I.2.3, I.2.6, I.2.9

Diplomarbeit Nr. 2103

**Einsatz probabilistischer
Verfahren zur
Entscheidungsfindung im
RoboCup**

Jörg Rüdener

Institut für Parallele
und Verteilte Systeme
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Danksagung

Ich danke Thorsten Buchheim für seine zahlreichen Hilfestellungen und Ratschläge, allen am CoPS-Team Beteiligten für ihre Unterstützung und ihre wertvolle Arbeit und Professor Levi, der diese Diplomarbeit ermöglicht hat. Ich bedanke mich außerdem bei Martin Kunzelnick für das Roboterbild auf Seite 2 und bei Daniel Minder für die Begutachtung der Arbeit. Weiterhin gilt mein Dank dem Decision Systems Laboratory an der University of Pittsburgh für ihre Bibliothek „smile“.

Inhaltsverzeichnis

1	Motivation und Einleitung	1
1.1	RoboCup	1
1.2	Das Stuttgarter RoboCup-Team	2
1.3	Bayessche Netze zur Entscheidungsfindung	4
1.4	Aufbau der Arbeit	5
1.5	Verwandte Arbeiten	6
2	Bayessche Netze	7
2.1	Überblick und Begriffserläuterungen	7
2.1.1	Unabhängigkeit	9
2.1.2	Berechnungen	10
2.1.3	Spezielle Wahrscheinlichkeitsknoten	12
2.2	Entscheidungsnetzwerke	13
2.3	Dynamische Bayessche Netze	14
2.4	Modellierung	15
2.5	Inferenz	16
2.6	Lernverfahren	17
2.7	Bewertung	19
3	Bibliothek für Bayessche Netze	23
3.1	Anforderungen und vorhandene Bibliotheken	23
3.2	Architektur	24
3.3	Inferenz	27
3.3.1	Inferenz in Polybäumen	27
3.3.2	Inferenz in Cliquenbäumen	30
3.4	Lernen	36

3.4.1	Einleitung	36
3.4.2	Herleitung der Gradientenabstiegsformel	37
3.4.3	Berechnung des Gradienten	38
3.4.4	Adaption des Gradienten	40
3.4.5	Architektur des Frameworks	41
4	Anwendung Bayesscher Netze im RoboCup	43
4.1	Besonderheiten des Szenarios	43
4.2	Modellierungstechniken	44
4.2.1	Diskretisierung	44
4.2.2	“Sensorknoten”	44
4.2.3	Innere Knoten	46
4.2.4	Baumstrukturen	46
4.2.5	Entscheidungen und Nützlichkeit	47
4.2.6	Bewertungsknoten	49
4.3	Ballbesitz	50
4.3.1	Aufbau des Netzes	51
4.3.2	Training	53
4.4	Schussentscheidung	55
4.4.1	Aufbau des Netzes	55
4.4.2	Training	57
4.5	Passspiel	64
4.5.1	Aufbau des Netzes	64
4.5.2	Evaluation	66
5	Zusammenfassung und Ausblick	67
5.1	Beitrag und Abgrenzung der Arbeit	67
5.2	Ausblick	67
A	Benutzung der Software	69
A.1	Bibliothek für Bayessche Netze	69
A.1.1	Grundlegende Benutzung	69
A.1.2	Adapter zum Weltmodell	71
A.1.3	Trainieren eines Bayesschen Netzes	73

A.1.4	Dateiformat	74
A.1.5	Hilfsprogramme	75
A.1.6	Verzeichnis- und Projektstruktur	77
A.2	Neue Spielerarchitektur	77
A.3	Spezielle Software für die hier vorgestellten Netze	79
A.3.1	Ballbesitz	79
A.3.2	Schussentscheidung	80
A.3.3	Passspiel	82
B	Verwendete Netzwerke	85
B.1	Ballbesitz	85
B.2	Schussentscheidung	93
B.2.1	Netzwerk mit einem Zielknoten, für statischen Torwart	93
B.2.2	Netzwerk mit einem Zielknoten, für sich regelförmig bewegend- Torwart	101
B.2.3	Netzwerk mit Bewertungsknoten, für statischen Torwart	105
B.2.4	Netzwerk mit Bewertungsknoten, für sich regelförmig bewegen- den Torwart	110
B.3	Passspiel	114
B.3.1	Netzwerk für den Nutzen eines Passes zu einer Position	114
B.3.2	Netzwerk für den Nutzen des Dribblings	118
	Glossar	121
	Abbildungsverzeichnis	123
	Tabellenverzeichnis	125
	Symbolverzeichnis	127
	Legende zu den Klassendiagrammen	129
	Literaturverzeichnis	131

Kurzfassung

Der RoboCup ist ein standardisiertes Szenario für die Entwicklung autonomer mobiler Roboter. Hier ist es nützlich, die Entscheidungen über auszuführende Aktionen durch probabilistische Berechnungen zu unterstützen. In dieser Arbeit wird untersucht, wie Bayessche Netzwerke zu diesem Zweck eingesetzt werden können. Es werden relevante Modellierungstechniken vorgestellt und mittels einiger so erstellter Netzwerke gezeigt, wie diese – eventuell nach Anwendung eines geeigneten Lernverfahrens – die ihnen gestellten Aufgaben im RoboCup lösen können. Außerdem wird die während der Arbeit entstandene Bibliothek zur Verwendung und zum Training Bayesscher Netze beschrieben.

Abstract

RoboCup is a standardized scenario to develop autonomous mobile robots. In this scenario it is useful to support decision making processes by probabilistic computations. This thesis examines the usage of bayesian (belief) networks for that purpose. It introduces relevant modelling techniques and demonstrates how thus constructed networks can perform different tasks after being trained by an appropriate learning algorithm. Additionally, it describes the developed library for the usage and training of bayesian networks.

Kapitel 1

Motivation und Einleitung

Diese Diplomarbeit beschäftigt sich mit dem Einsatz Bayesscher Netze zur Entscheidungsfindung im RoboCup-Szenario. In den folgenden Abschnitten dieses Kapitels werden ein Überblick über die Rahmenbedingungen der Arbeit gegeben, das Szenario erläutert und die weitere Gliederung der Arbeit vorgestellt.

1.1 RoboCup

Das RoboCup-Szenario [rbc] bietet seit 1997 eine standardisierte Problemstellung im Bereich der Robotik und der autonomen Agenten. Ziel ist es, Roboter zu entwickeln, die Fußball spielen. Dabei teilt sich der RoboCup in mehrere Ligen auf:

- In der Simulations-Liga spielen keine physischen Roboter, sondern es spielen Mannschaften aus je 11 in reiner Software realisierten autonomen Agenten gegeneinander, während ein Server die Umwelt (Spielfeld, Ball, Interaktionen, Schiedsrichter, ...) simuliert.
- In der Sony Four Legged-Liga spielen Aibo-Hunde von Sony ([aib]) gegeneinander.
- In der Small Size-Liga spielen je vier kleine Roboter gegeneinander auf einem Feld von der Größe einer Tischtennisplatte. Sie werden zentral von einem Rechner gesteuert, der Daten von einer über dem Feld angebrachten Deckenkamera erhält.
- In der Middle Size-Liga spielen je vier Roboter von ca. 50 cm Durchmesser und bis zu 80 cm Höhe vollkommen autonom auf einem Feld der Größe von ca. fünfeinhalb mal achteinhalb Metern. Um die Orientierung der Roboter zu erleichtern, sind an den Ecken des Spielfelds farbig markierte Pfosten aufgestellt; auch die Tore und der Ball haben definierte Farben, damit sie von den Kameras der Roboter leichter erkannt werden können.

- In einer zukünftig geplanten Liga sollen humanoide Roboter gegeneinander spielen; großes Ziel ist es, im Jahr 2050 ein Team aus Robotern zu haben, das gegen menschliche Fußballer gewinnen kann.
- Schließlich gibt es noch weitere spezielle Ligen für Schüler oder für Rettungsroboter, die der RoboCup-Initiative angeschlossen sind.

Das Stuttgarter RoboCup-Team (s.u.) spielt in der Middle Size-Liga, für welche auch die in dieser Arbeit betrachteten Szenarien gelten. Die Ergebnisse dieser Arbeit sind aber problemlos auf andere Ligen und allgemein auf die Thematik autonomer Agenten, die in Echtzeit Entscheidungen treffen müssen, übertragbar.

1.2 Das Stuttgarter RoboCup-Team

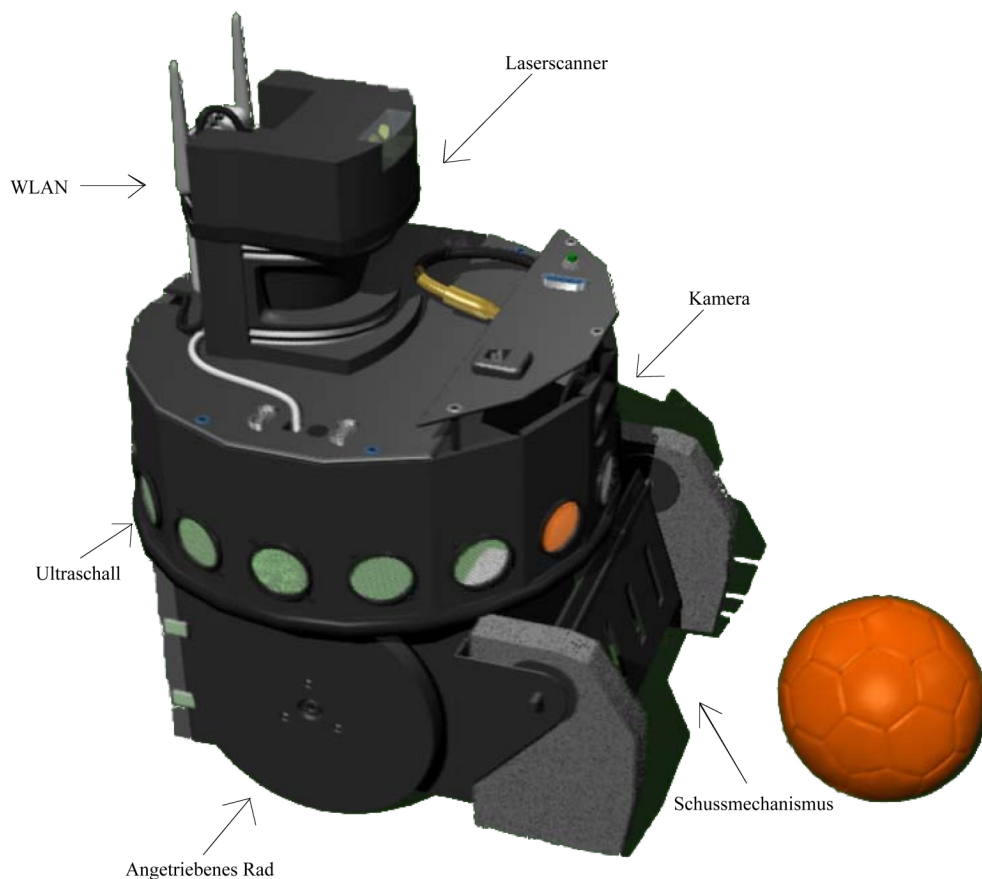


Abbildung 1.1: *Roboter des Teams CoPS Stuttgart*

Das Stuttgarter Team CoPS (**co**operative **soc**cer **p**laying **r**obots) nimmt seit vielen Jahren erfolgreich am RoboCup in der Middle Size-Liga teil (so wurde etwa bei den

Weltmeisterschaften 2001 in Seattle ein vierter Platz erreicht). Die Roboter des Teams sind alle auf dem Grundmodell eines Nomad Scout-Roboters aufgebaut (siehe Abbildung 1.1 auf der vorherigen Seite). Diese haben zwei von Elektromotoren angetriebene Räder an den Seiten und ein Stützrad hinten. Sie bringen auch eine Reihe von Ultraschallsensoren mit, die aber wegen ihrer Störanfälligkeit kaum mehr benutzt werden. Die Roboter wurden erweitert durch einen magnetischen Schussmechanismus vorne (beim Torwart an der Seite), einen Laserscanner zur Hinderniserkennung und Selbstlokalisierung und eine oder (beim Torwart) drei Kameras, die hauptsächlich zur Ballerkennung eingesetzt wird. Durch zwei Schaumstoffteile neben dem Schussmechanismus können sie im Rahmen der Regeln eine gewisse Kontrolle über den Ball ausüben (ein Schuss ist aber nur geradeaus möglich). Über ein drahtloses Netzwerk sind die Roboter in der Lage, untereinander Informationen wie beispielsweise ihre Positionsdaten oder die des Balles auszutauschen.

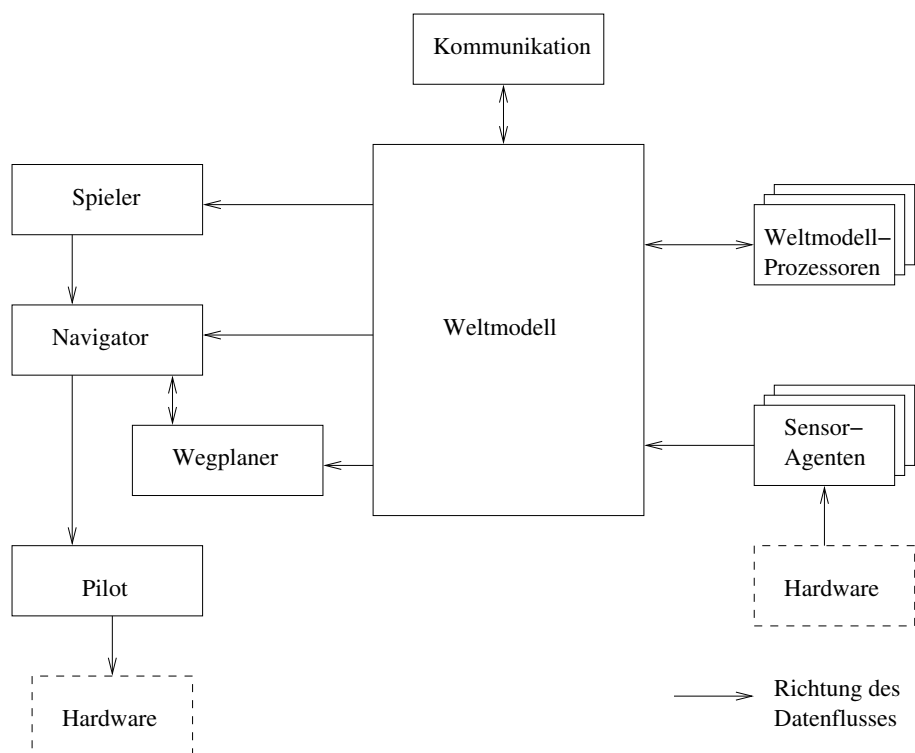


Abbildung 1.2: Softwarearchitektur im Team CoPS

Die Softwarearchitektur der Roboter stellt ein Multi-Agenten-System dar und ist vereinfacht in Abbildung 1.2 gezeigt. Die Agenten greifen alle gemeinsam auf die Daten eines Weltmodells zu, die von einer Reihe unabhängiger Algorithmen berechnet werden (sog. *Weltmodellprozessoren*, dieses Prinzip ist beschrieben in [BKLL03]). Die Agenten teilen sich einerseits auf in Sensorauswertungskomponenten, die beispielsweise die Daten des Laserscanners auslesen, und andererseits mit den Aktionen des Roboters beschäftigte Agenten. Diese bauen schichtenartig aufeinander auf: der Pilot auf unterster Ebene bietet eine Hardwareabstraktionsschicht und sorgt für die intelligente Ansteuerung der Räder. Der Navigator darüber führt einzelne Handlungselemente aus, wie

z.B. Anfahren des Balles oder einer Position, einen Torschuss oder das Dribbling mit dem Ball. Er bedient sich dabei der Funktionen des Wegplaners, der einen Fahrtweg um die vorhandenen Hindernisse herum berechnet. Der Spieler schließlich ist die Entscheidungsinstanz: er wertet die im Weltmodell gespeicherte Situation aus und gibt dem Navigator die dafür passenden Befehle.

Die Bayesschen Netze in dieser Diplomarbeit waren daher im Bereich des Weltmodells und des Spielers angeordnet. Wie in Kapitel 3 erläutert, war es jedoch auch nötig, eine Bibliothek zur Verwendung der Bayesschen Netze zu entwickeln. Diese ist zwar eine eigene Komponente, als reine Hilfsbibliothek jedoch kein eigener Agent, sodass sie in Abbildung 1.2 nicht gezeigt wird.

1.3 Bayessche Netze zur Entscheidungsfindung

Im RoboCup-Szenario ist es offensichtlich, dass eine Rechnung mit absoluten Wahrheitswerten schwere Nachteile hat – insbesondere wegen der mit Unsicherheiten behafteten Sensordaten, die zudem nicht immer eindeutig interpretiert werden können. Weitere Unsicherheit entsteht durch fehlendes Wissen, da die Sensoren eines Roboters nicht das gesamte Spielfeld abdecken. Deshalb wird auch bisher stets mehr oder weniger implizit mit Wahrscheinlichkeiten gearbeitet. Dies geschieht etwa dadurch, dass Schwellwerte festgelegt werden, bei deren Über- oder Unterschreitung eine logische Variable ihren Wert wechselt (beispielsweise wird angenommen, dass ein Roboter den Ball kontrolliert, sobald der Abstand zwischen ihm und dem Ball kleiner als ein bestimmter Wert ist). Hierbei ist ein großer Nachteil der sprunghafte Übergang an der Grenze. An anderen Stellen werden den booleschen Variablen Wahrscheinlichkeiten zugeordnet (in manchen Fällen über ausgefeilte Algorithmen, in manchen Fällen einfach umgekehrt proportional zur verstrichenen Zeit von der letzten Messung an), die dann ebenfalls mit Schwellwerten versehen werden.

Damit ein Roboter rationale Entscheidungen treffen kann, ist es dagegen häufig notwendig, eine Reihe wahrscheinlichkeitsbehafteter Variablen miteinander zu verknüpfen und aus ihren Werten Schlüsse zu ziehen, die ebenfalls eine gewisse Wahrscheinlichkeit haben. Zwar müssen am Ende, wenn der Roboter eine Aktion auswählen muss, natürlich immer noch Schwellwerte verwendet werden. Jedoch können diese je nach Situation flexibel gehandhabt werden, und vor allem geht beim probabilistischen Schließen keine Information verloren: Schwellwerte werden nicht am Anfang, sondern am Ende eingesetzt, sodass die ursprünglichen Wahrscheinlichkeiten stets in der Rechnung erhalten bleiben.

Probabilistische Entscheidungsfindung ist ein in der Künstlichen Intelligenz relativ junges, aber inzwischen etabliertes Feld der Forschung. Während für manche einfachere Berechnungen Methoden wie Hidden Markov Models ausreichen, haben sich für komplexere Sachverhalten die sog. *Bayesschen Netzwerke* durchgesetzt, die mit einer Erweiterung als *Entscheidungsnetzwerke* sehr gut für die Aktionsauswahl eignen. Bayessche Netze, deren Aufbau und Funktionsweise in Kapitel 2 ausführlich erläutert werden, bieten u.a. folgende Vorteile:

- Sie ermöglichen (nicht nur kausale, sondern auch diagnostische und interkausale) Schlussfolgerungen unter stetiger Einbeziehung von Wahrscheinlichkeiten
- Sie sind von Menschen relativ einfach zu verstehen und von im Anwendungsgebiet erfahrenen Experten auch gut zu modellieren
- Sie können vom Rechner anhand von Beispielen automatisch erlernt werden
- Sie bieten eine Möglichkeit zur Strukturierung komplexer Zusammenhänge
- Sie sind üblicherweise ausreichend schnell

Nachteile der Bayesschen Netze sind ein gewisser Aufwand, um sie an veränderte Umweltbedingungen anzupassen, und eine im schlechtesten Fall äußerst geringe Geschwindigkeit. Außerdem sind derzeit noch wenige Bibliotheken zur Verwendung Bayesscher Netze frei verfügbar (siehe Abschnitt 3.1), sodass anfangs häufig Arbeit investiert werden muss, um die nötige Software-Infrastruktur zu schaffen.

1.4 Aufbau der Arbeit

Die weitere Arbeit ist gemäß der behandelten Themenschwerpunkte gegliedert. Kapitel 2 gibt eine Einführung in die Funktion und den Aufbau Bayesscher Netze und ihrer Erweiterungen. Es befasst sich auch mit grundlegenden Berechnungen und allgemeinen Modellierungstechniken und gibt eine Übersicht über bekannte Algorithmen zur Durchführung von Inferenz und zum automatischen Lernen.

Kapitel 3 beschäftigt sich mit der Erstellung einer Bibliothek für die Verwendung Bayesscher Netze. Es werden die Anforderungen an solch eine Bibliothek im RoboCup-Szenario dargestellt und erläutert, warum eine Eigenentwicklung notwendig war. Dann werden die Algorithmen detailliert vorgestellt, die in der Bibliothek verwendet werden. Außerdem gibt das Kapitel eine Beschreibung der Architektur der Bibliothek.

In Kapitel 4 wird ausführlich auf die Anwendung Bayesscher Netze im RoboCup-Szenario eingegangen. Zunächst werden eine Reihe von Modellierungstechniken vorgestellt, die häufig verwendet werden können, um für das Szenario besonders passende Netze zu entwickeln. Danach werden einige Beispiele Bayesscher Netze vorgestellt, die in verschiedenen Szenarien die Anwendbarkeit der Verfahren demonstrieren.

Das abschließende Kapitel 5 fasst die wichtigsten Ergebnisse der Arbeit zusammen und gibt einen Ausblick auf zukünftige Anwendungen.

Im Anhang ist noch eine Anleitung zur Benutzung der entstandenen Software, insbesondere der Bibliothek, enthalten.

1.5 Verwandte Arbeiten

Bayessche Netze wurden bisher im RoboCup noch nicht intensiv eingesetzt. H. Younes hat in seiner Diplomarbeit ([You98]) anhand einiger sehr einfacher Netze immerhin gezeigt, dass die Standardinferenzverfahren bekannter Bibliotheken für die Anwendung im RoboCup schnell genug sind.

Die Anwendung Bayesscher Netze in autonomen mobilen Systemen ist dagegen schon etwas häufiger. Nikovski et al. ([NMR]) benutzen z.B. ein Dynamisches Bayessches Netzwerk (DBN) in einem Roboter, der Museumsführungen macht; dabei wurden gewisse Markov-Modelle auch trainiert ([NN99]), u.a. mit einer Variante des EM-Algorithmus. Ein großes Forschungsprojekt zum Einsatz Bayesscher Netze in einem autonomen Fahrzeug war das „BATmobile“ von Forbes et al. ([FHKR95]). Hier wurden DBNs u.a. dazu verwendet, die Entscheidung über einen Fahrstreifenwechsel auf einer (simulierten) Autobahn zu erleichtern. In diesem Zusammenhang hat Forbes auch Verstärkungslernen eingesetzt ([For02], allerdings wurden nicht die DBNs trainiert). In [Ham01] wurde die Verwendung von DBNs in Verbindung mit und im Vergleich zu anderen temporalen Verfahren zur Vorhersage von Verkehrssituationen erörtert. Sahami ([Sah]) schlägt vor, die a-priori Wahrscheinlichkeiten eines Bayesschen Netzwerks dynamisch aus den Sensordaten zu errechnen, und demonstriert dies an einem kleinen Netzwerk zur Kollisionsvermeidung.

Die generelle Verwendung probabilistischer Methoden ist im RoboCup schon viel weiter verbreitet, sie werden dabei hauptsächlich zur Weltmodellierung eingesetzt (z.B. zur Selbstlokalisierung über Laserdaten, siehe [FBT99]). In [Sto98] wird auch eine Variante des Verstärkungslernens eingesetzt, um Verhaltensweisen von Agenten in der Simulationsliga zu erlernen; hierbei werden Entscheidungsbäume verwendet.

Kapitel 2

Bayessche Netze

2.1 Überblick und Begriffserläuterungen

Bayessche Netze sind schon seit vielen Jahren ein etabliertes Mittel, um unsichere Daten, Zusammenhänge und mit Wahrscheinlichkeiten behaftete Variablen zu modellieren sowie um Schlüsse aus unsicherem Wissen zu ziehen und Entscheidungen unter unsicherem Wissen zu treffen. Einführungen in das Prinzip der Bayesschen Netze finden sich inzwischen in den meisten Lehrbüchern über künstliche Intelligenz (wie z.B. [RN95]) oder am Anfang von Arbeiten, die sich mit speziellen Themen befassen (sehr gut z.B. bei [Kra98]). Auch Bücher, die sich allein mit Bayesschen Netzen beschäftigen, sind vorhanden – der „Klassiker“ ist von Pearl [Pea88], eine gute Einführung auch zur Modellierung von Bayesschen Netzen bietet Jensen in [Jen01].

Ein Bayessches Netzwerk (auf englisch auch *belief network* genannt) besteht aus zwei Teilen: einer meist grafisch dargestellten Struktur und einer dahinterliegenden mathematischen Semantik. Die Struktur ist ein gerichteter azyklischer Graph (DAG). Dabei steht jeder Knoten für eine mit einer Wahrscheinlichkeit behaftete Variable, während die Kanten die Abhängigkeiten zwischen den Variablen modellieren. Klassische Bayessche Netzwerke enthalten nur diskrete Variablen¹. Das bedeutet (wenn man einen Knoten mit einer Variable identifiziert), dass jeder Knoten eine Menge an Zuständen besitzt, die er einnehmen kann.

Jedem dieser Wahrscheinlichkeitsknoten ist weiterhin eine Tabelle zugeordnet, in der zu jeder Kombination der Zustände seiner Elternknoten im Netz die bedingten Wahrscheinlichkeiten dafür, dass der Knoten einen seiner Zustände annimmt, angegeben sind. Diese Tabelle heißt *Conditional Probability Table (CPT)*; sie kann für einen Knoten X auch als Funktion $\psi : \text{parents}(X) \cup \{X\} \rightarrow [0, 1]$ angesehen werden. Durch die Struktur des Netzwerks und die CPTs ist das Bayessche Netzwerk komplett bestimmt.

¹Es gibt auch Ansätze zur Verwendung von kontinuierlichen Variablen, die dann über Wahrscheinlichkeitsverteilungen modelliert werden; auf diese Art der Bayesschen Netze wird hier aber nicht weiter eingegangen, da ihre Anwendung bis jetzt nur eingeschränkt möglich ist.

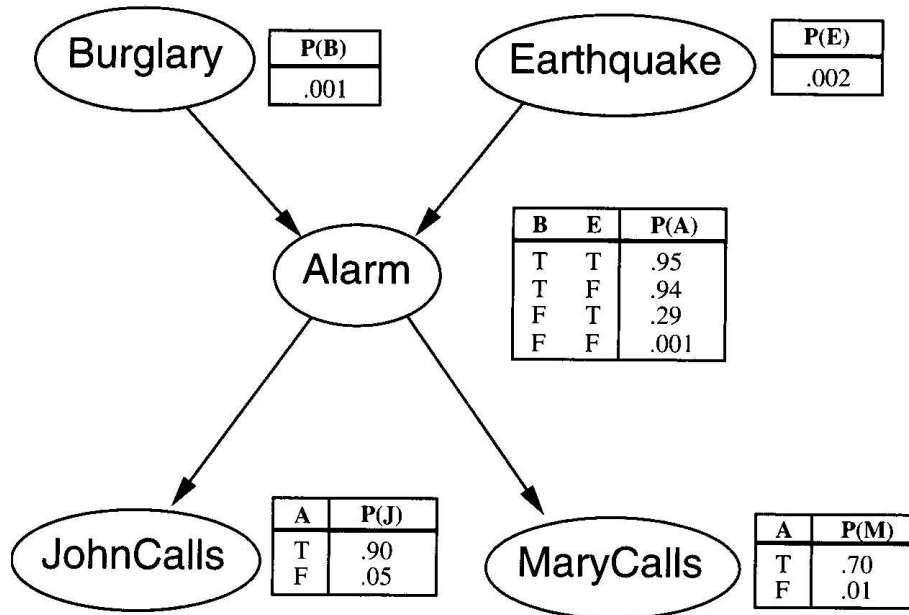


Abbildung 2.1: Beispiel eines Bayesschen Netzwerkes

Abbildung 2.1 zeigt beispielhaft ein Bayessches Netz mit fünf Knoten (aus [RN95]). Es modelliert, ob eine Alarmanlage in einem Haus läutet, während der Besitzer verreist ist, und ob dann einer seiner Nachbarn (Mary und John) den Besitzer anruft, um ihn davon zu informieren. In diesem Netzwerk sind nur boolesche Variablen enthalten, jeder Knoten hat also genau zwei Zustände. In den CPTs sind die Wahrscheinlichkeiten für den Zustand *false* jeweils der Übersicht wegen weggelassen, da stets $P(X = false|parents(X)) = 1 - P(X = true|parents(X))$ gilt. Zur Interpretation des Netzwerkes und der CPTs einige Beispiele:

- Ob der Alarm losgeht, hängt davon ab, ob es einen Einbruch (Burglary, B) gab und ob es ein Erdbeben (Earthquake, E) gab
- Falls es keinen Alarm gibt, ruft John mit einer Wahrscheinlichkeit von 5% an
- Ein Erdbeben tritt mit einer Wahrscheinlichkeit von 0.2% auf
- Falls es keinen Einbruch, aber ein Erdbeben gibt, wird der Alarm mit einer Wahrscheinlichkeit von 71% nicht losgehen

Falls der Zustand eines oder mehrerer Knoten bekannt ist (dies wird als *Evidenz* bezeichnet), so kann man daraus auf die Wahrscheinlichkeiten für die Zustände der anderen Knoten schließen. Diesen Vorgang nennt man *Inferenz*.

Sofern sie richtig modelliert wurden, sind Bayessche Netzwerke intuitiv verständlich und in vielen Bereichen nutzbar. Hauptsächlich wurden sie bisher in der Wissensverarbeitung eingesetzt, u.a. für medizinische Diagnosen und *data mining*. Aber auch in der Robotik oder im Bereich der autonomen Fahrzeuge gab es schon Untersuchungen

und Projekte, die Bayessche Netzwerke eingesetzt haben (siehe Abschnitt 1.5). Mit einer einfachen Erweiterung können sie praktisch direkt als Entscheidungsverfahren eingesetzt werden (s.u.).

2.1.1 Unabhängigkeit

Eine wichtige Eigenschaft Bayesscher Netze ist, dass sie die Unabhängigkeit von Wahrscheinlichkeitsvariablen direkt modellieren – d.h, man kann anhand der Struktur des Netzes sofort bestimmen, ob eine Menge von Variablen unabhängig von einer anderen Menge ist, falls eine Reihe von Knoten Evidenz trägt. Es gibt eine Eigenschaft von Knotenmengen, *d-Separation*² genannt, die genau äquivalent zu dieser Unabhängigkeit ist. Zwei Knotenmengen A und B werden von der Menge E der Knoten mit Evidenz *d-separiert*, wenn jeder ungerichtete Pfad von einem Knoten in A zu einem Knoten in B durch E *geblickt* wird. Ein Pfad wird von E geblickt, wenn es einen Knoten Z auf ihm gibt, für den eine der drei folgenden Bedingungen zutrifft (siehe Abbildung 2.2):

1. $Z \in E$ und eine der Kanten auf dem Pfad führt zu Z hin, die andere führt von Z weg
2. $Z \in E$ und beide Kanten auf dem Pfad führen von Z weg
3. Weder Z noch irgendein Nachkomme von Z sind in E und beide Kanten auf dem Pfad führen zu Z hin

Diese Definition hört sich zunächst etwas kompliziert an, ist aber im Prinzip einleuchtend: es muss verhindert werden, dass sich geänderte Evidenz in A entlang der Kanten zu B auswirkt (sonst wären die Knotenmengen eben nicht mehr unabhängig); und dazu muss es auf jedem möglichen Ausbreitungspfad mindestens einen Knoten geben, der die Ausbreitung verhindert.

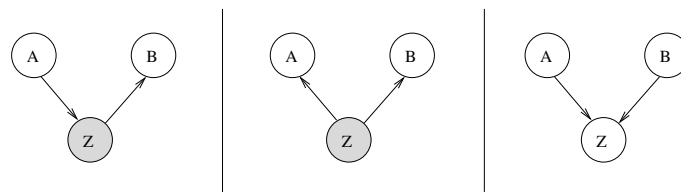


Abbildung 2.2: Beispiele zur *d-Separation*: Knoten A und B sind jeweils *d-separiert*. Schattierte Knoten haben Evidenz.

Einige Beispiele für *d-Separation* anhand des Beispielnetzwerks aus Abbildung 2.1:

- Falls *Alarm* Evidenz trägt, so sind *MaryCalls* und *Earthquake* *d-separiert* und somit unabhängig voneinander
- Ebenso sind in diesem Fall *MaryCalls* und *JohnCalls* *d-separiert*

²von direction-dependent separation

- Trägt *Alarm* dagegen keine Evidenz, so sind *MaryCalls* und *JohnCalls* abhängig: Ruft Mary an, so könnte das bedeuten, dass es Alarm gab, und in dem Fall steigt die Wahrscheinlichkeit für einen Anruf von John
- *Burglary* und *Earthquake* sind nur dann d-separiert, wenn keiner der anderen Knoten Evidenz trägt. Gibt es nämlich einen Hinweis auf den Alarm und man erfährt, dass ein Erdbeben stattgefunden hat, so verringert dies die Wahrscheinlichkeit eines Einbruchs wieder (sog. „wegerklären“)

Bei Berechnungen sollte man sich in Erinnerung behalten, dass für unabhängige Variablen A und B $P(A, B|E) = P(A|E)P(B|E)$ gilt.

2.1.2 Berechnungen

Vereinigte Wahrscheinlichkeitsfunktion

Jedes Bayessche Netzwerk ist direkt eine Repräsentation einer Wahrscheinlichkeitsfunktion seiner Variablen (auf englisch häufig *Joint Probability Distribution* oder *JPD* genannt). Diese Funktion erhält man, indem man alle Knotenwahrscheinlichkeiten miteinander multipliziert:

$$P(\mathbf{x}|E) = \prod_{i=1}^n P(x_i | \text{Parents}(X_i)) \quad (2.1)$$

Um beim Beispiel zu bleiben: Die Wahrscheinlichkeit dafür, dass der Alarm losgeht, John und Mary anrufen, aber weder ein Einbruch noch ein Erdbeben auftraten, ist $P(J, M, A, \bar{B}, \bar{E}) = P(J|A)P(M|A)P(A|\bar{B}, \bar{E})P(\bar{B})P(\bar{E}) = 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 = 0.00062$.

Bayessche Regel

Äußerst wichtig beim Berechnen der Wahrscheinlichkeiten und für die Inferenzalgorithmen ist die *Bayessche Regel*. In der einfachen Form erhält man sie aus der Rechnung $P(A|B)P(B) = P(A, B) = P(B|A)P(A)$ als

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (2.2)$$

In der erweiterten Form unter Berücksichtigung zusätzlicher Evidenz E lautet sie

$$P(A|B, E) = \frac{P(B|A, E)P(A|E)}{P(B|E)} \quad (2.3)$$

Die Bayessche Regel bietet nicht nur eine Möglichkeit, unbekannte bedingte Wahrscheinlichkeiten zu berechnen, sondern führt auch häufig zu Vereinfachung dadurch, dass die Abhängigkeit zwischen den beiden Variablen in zwei der drei neuen Terme nicht mehr vorhanden ist.

Normalisierung

Bei der Anwendung der Bayesschen Regel kann der Nenner auf der rechten Seite der Gleichung 2.2 häufig als Konstante betrachtet werden, was zur Form

$$P(A|B) = \alpha P(B|A)P(A) \tag{2.4}$$

führt. Berechnet man zuerst $P(B|A)P(A)$ für alle Zustände von A und B und multipliziert dann die erhaltenen Werte mit α , so wird dieser Vorgang als *Normalisierung* bezeichnet, weil erst dadurch die Summe der berechneten Werte auf 1 gebracht wird. Es gilt daher auch

$$\alpha = 1 / \sum_B P(B|A)P(A) \tag{2.5}$$

Besonders wichtig ist diese Eigenschaft hinsichtlich der CPTs, bei denen α genau dem Kehrwert der Summe über eine Dimension entspricht. Ein Beispiel zur Normalisierung zeigt Abbildung 2.3.

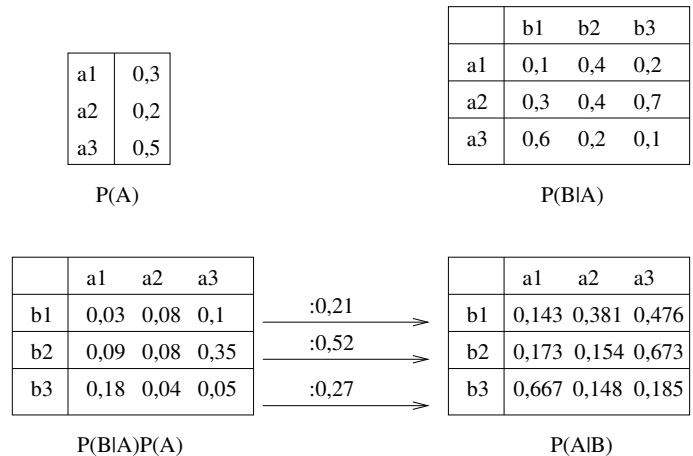


Abbildung 2.3: Beispiel für eine Normalisierung. Aus $P(B|A)P(A)$ geht $P(A|B)$ hervor, indem durch die jeweilige Zeilensumme dividiert wird. Diese Zeilensummen sind genau $P(B)$.

Aussummieren von Variablen

Die Rechnung $P(A|E) = \sum_X P(A, X|E)$ wird als *Aussummieren* der Variable X bezeichnet. Die Notation bedeutet einfach, dass sich $P(a_1^i, \dots, a_n^i)$ als $\sum_X P(a_1^i, \dots, a_n^i, x^j)$ berechnet (i bezeichnet den Index der Zustandskombination der Variablen in A). Nach dem gleichen Prinzip lässt sich auch eine ganze Variablenmenge aus einer Wahrscheinlichkeitsfunktion aussummieren.

Multiplikation von CPTs

Für CPTs lässt sich eine besondere Art der Multiplikation definieren, durch die zwei CPTs zu einer gemeinsamen CPT über der Vereinigungsmenge der betrachteten Knoten zusammengefasst werden. Dabei entsteht jeder Eintrag der neuen CPT aus der

Multiplikation der korrespondierenden – also zu den gleichen Zuständen der jeweiligen Knoten gehörenden – Einträge in den beiden alten CPTs.

Multipliziert man z.B. die CPTs der Knoten *Alarm* und *Earthquake* des Beispielnetzwerks aus Abbildung 2.1, so erhält man eine neue CPT der gleichen Größe wie der CPT von *Alarm*, u.a. mit dem Eintrag $P_{AE}(A = \text{true}|B = \text{true}, E = \text{false}) = P_A(A = \text{true}|B = \text{true}, E = \text{false})P_E(E = \text{false}) = 0.94 \times 0.998 = 0.93812$.

Multipliziert man die CPTs der Knoten *MaryCalls* und *Alarm*, so erhält man eine CPT mit 8 Zeilen für die Variablen *B*, *E*, *A* und *M*, u.a. mit dem Eintrag $P_{AM}(M = \text{true}|B = \text{true}, E = \text{false}, A = \text{true}) = P_A(A = \text{true}|B = \text{true}, E = \text{false})P_M(M = \text{true}|A = \text{true}) = 0.94 \times 0.70 = 0.658$.

Wie leicht zu ersehen ist, ist diese Art der Multiplikation kommutativ und assoziativ.

2.1.3 Spezielle Wahrscheinlichkeitsknoten

Manchmal lässt sich für eine CPT eine ganz bestimmte Form angeben, was die Anzahl freier Variablen einschränkt und die Berechnungen vereinfacht und beschleunigt. Besonders häufig benutzte Formen werden hier kurz angesprochen.

Deterministische Knoten

In einem deterministischen Knoten hat die CPT nur Nullen und Einsen als Einträge. Die Variable ist dabei häufig eine logische Verknüpfung der Variablen der Elternknoten; sie kann aber auch eine einfache mathematische Verknüpfung sein wie beispielsweise das Maximum der Elternknoten.

Noisy-OR Knoten

Noisy-OR ist eine Möglichkeit, kausale Beziehungen zu multiplizieren. Man geht davon aus, dass der Grund dafür, dass eine (boolsche) Variable „wahr“ wird, stets mindestens eine wahre Elternvariable ist. Anders gesagt, modellieren die Elternknoten genau alle Gründe für den Kindknoten. Die Beziehung ist aber eben nicht deterministisch, sondern „verrauscht“ („noisy“): mit einer bestimmten Wahrscheinlichkeit bleibt die Variable „falsch“, selbst wenn die Elternvariable „wahr“ ist. Diese Wahrscheinlichkeit wird als „noise parameter“ bezeichnet, verursacht wird sie konzeptionell von einem Inhibitor. Man nimmt weiter an, dass die Inhibitoren voneinander unabhängig sind; dadurch errechnet sich die Wahrscheinlichkeit für false im Falle mehrerer auf true stehender Elternknoten als Produkt der einzelnen Inhibitorwahrscheinlichkeiten.

Als einfaches Beispiel (nach [RN95]) kann man betrachten, dass Fieber nur von ein paar wenigen Ursachen erzeugt werden könne, darunter Erkältung und Grippe. Ist dann $P(\text{Fieber}|\text{Erk.}) = 0.4$, so ist der noise parameter für Fieber also 0.6. Ist weiterhin $P(\text{Fieber}|\text{Grippe}) = 0.8$, so kann man errechnen: $P(\text{Fieber}|\text{Erk.}, \text{Grippe}) =$

$$1 - P(\neg \text{Fieber} | \text{Erk.}, \text{Grippe}) = 1 - P(\neg \text{Fieber} | \text{Erk.})P(\neg \text{Fieber} | \text{Grippe}) = 1 - 0.6 \times 0.2 = 0.88.$$

Noisy-AND Knoten

Ganz ähnlich sind die *Noisy-AND* Knoten aufgebaut. Hier geht man davon aus, dass normalerweise alle Elternvariablen true sein müssen, damit die Kindvariable true wird (vgl. logische UND-Verknüpfung). Der „noise parameter“ gibt dann die Wahrscheinlichkeit an, dass die Kindvariable true wird, obwohl ein Elternknoten false ist. Die Wahrscheinlichkeit für true im Falle mehrerer auf false stehender Elternknoten errechnet sich wiederum als Produkt der Einzelwahrscheinlichkeiten.

2.2 Entscheidungsnetzwerke

Bayessche Netze lassen sich durch recht einfache Art und Weise erweitern, um Entscheidungen zu unterstützen oder gar Entscheidungsprozesse zu repräsentieren. Hierfür fügt man dem Netz zwei Typen von Knoten hinzu:

- *Entscheidungsknoten* repräsentieren die zur Wahl stehenden Möglichkeiten bei einer Entscheidung. Elternknoten eines Entscheidungsknoten können nur andere Entscheidungsknoten sein. Diese Beziehung besagt dann, dass die vom Knoten modellierte Entscheidung nach den Entscheidungen bei seinen Eltern zu fallen hat. Entscheidungsknoten zeigen nur die Möglichkeiten auf; sie geben keinen Hinweis auf die Wahrscheinlichkeit einer bestimmten Wahl. Sie werden graphisch durch Rechtecke dargestellt.
- *Nützlichkeitsknoten* geben den Nutzen einer bestimmten Entscheidung an und werden graphisch durch auf der Spitze stehende Rauten dargestellt. Sie haben mindestens einen Entscheidungsknoten und mindestens einen Wahrscheinlichkeitsknoten unter ihren Vorfahren. In einer ihnen zugeordneten Tabelle steht für jede Kombination der Zustände ihrer Eltern die Nützlichkeit dieser Kombination. Normalerweise tragen die Wahrscheinlichkeitsknoten, die direkt Eltern der Nützlichkeitsknoten sind, keine Evidenz; das bedeutet, dass nach einem Inferenzdurchgang ein Nützlichkeitsknoten für jede mögliche Entscheidung den *mittleren erwarteten Nutzen* angibt, den diese Entscheidung hat. Dadurch kann dann eine Entscheidung getroffen werden, sinnvollerweise diejenige mit dem höchsten erwarteten Nutzen.

Abbildung 2.4 zeigt ein einfaches Beispiel für ein Entscheidungsnetzwerk. Ein Roboter könnte es für die Entscheidung benutzen, ob er hindernisbedingt bremsen soll, während er zum Ball fährt. Von seiner Entscheidung (und anderen Daten wie der Hindernisdistanz, der Ballposition usw.) hängt ab, ob es zu einer Kollision mit einem anderen Roboter kommt und ob der Roboter in den Ballbesitz gelangt. Diese Daten wiederum bestimmen den Nutzen, den der Roboter von seiner Entscheidung hat.

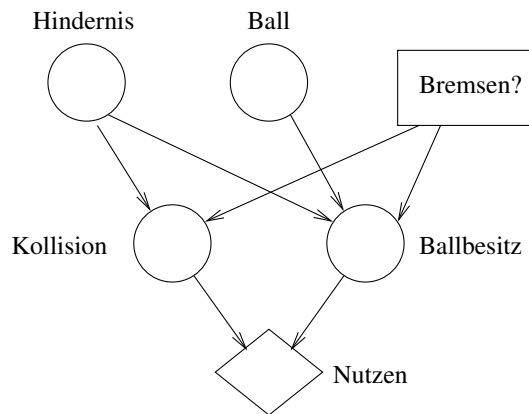


Abbildung 2.4: Beispiel eines Entscheidungsnetzwerks

Inferenz in einem Entscheidungsnetzwerk wird üblicherweise so durchgeführt, dass man die Evidenz aller Wahrscheinlichkeitsknoten berücksichtigt und dann für jede mögliche Entscheidung (bzw. Kombination von Entscheidungen) die Wahrscheinlichkeiten der abhängigen Knoten mit einem bekannten Inferenzalgorithmus bestimmt. Die Nützlichkeiten erhält man danach durch Summation.

2.3 Dynamische Bayessche Netze

Bayessche Netze lassen sich ebenfalls einsetzen, um zeitbehaftete Modelle zu erstellen und Abhängigkeiten von zeitlich früheren Daten zu berücksichtigen. Die entsprechende Erweiterung nennt sich „Dynamische Bayessche Netze“ (DBN). Sie funktionieren, indem man das Netz in Zeitscheiben aufteilt und die neue Inferenz nach und nach hinzufügt. Üblicherweise geht man dabei davon aus, dass die *Markov-Eigenschaft* erfüllt ist, dass also nur höchstens einen Zeitschritt zurückliegende Werte die aktuellen Werte direkt beeinflussen. Die generelle Struktur eines solchen DBN zeigt Abbildung 2.5.

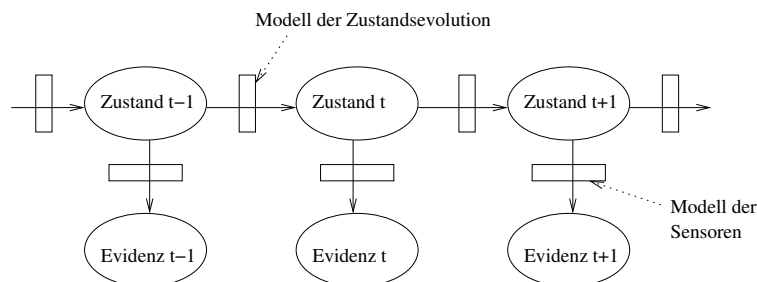


Abbildung 2.5: Generelle Struktur eines DBN (nach [RN95])

Inferenz führt man dann folgendermaßen durch:

1. Sei die Evidenz für Zeitpunkt $t - 1$ schon vorhanden. Durch normale Inferenz erhält man Wahrscheinlichkeiten $\hat{P}_t(X)$.

2. Nun entfernt man die Zeitscheibe von $t - 1$ und gibt den Zustandsvariablen in t statt der bedingten Wahrscheinlichkeiten a-priori-Wahrscheinlichkeiten - diese sind genau $\hat{P}_t(X)$.
3. Daraufhin legt man die aus den Sensoren gewonnene neue Evidenz (in Zeitscheibe t) fest und führt erneut Inferenz durch, dadurch erhält man die echten Wahrscheinlichkeiten $P_t(X)$.
4. Schließlich fügt man die nächste Zeitscheibe hinzu und kann beim ersten Schritt einen neuen Zyklus starten.

DBNs können genau wie normale Bayessche Netze erweitert werden zu Dynamischen Entscheidungsnetzwerken (DDNs). Dabei werden die selben Entscheidungen in jeder Zeitscheibe erneut getroffen. Sowohl DBNs als auch DDNs können gerade in der Robotik sehr nützlich sein. Beispielsweise könnte man modellieren, dass ein defekter Sensor sehr viel wahrscheinlicher ist, wenn er schon in der vorhergehenden Zeitscheibe defekt war, oder dass es wahrscheinlicher ist, im Ballbesitz zu sein, wenn man in der vorhergehenden Zeitscheibe im Ballbesitz war (wobei allerdings Rückkopplungs- und Verstärkungseffekte berücksichtigt werden müssen). Die in dieser Arbeit betrachteten Netzwerke kommen jedoch ohne zeitliche Abhängigkeiten aus.

2.4 Modellierung

Sehr wichtig für die optimale Nutzung Bayesscher Netze ist die Modellierung ihrer Struktur. Bei falscher Modellierung geschieht es häufig, dass das entstehende Netz zu viele Kanten hat (und daher auch viel zu viele Parameter) oder dass die Bedeutung der bedingten Wahrscheinlichkeiten nur schwer zu ergründen ist (und daher auch die Angabe sinnvoller Werte äußerst schwierig wird).

Eine sehr gute Methode, Bayessche Netze zu konstruieren, ist es, auf die Kausalitätsbeziehungen zwischen den Variablen zu achten. Wenn man stets eine Verbindung von einem Knoten zu einem anderen hinzufügt, falls er ihn *direkt* kausal beeinflusst, so kann man damit viele Probleme vermeiden. U.a. werden transitive Kanten vermieden und die Angabe der CPT-Werte wird vereinfacht. Zudem repräsentiert ein solches Netzwerk meistens auch die Unabhängigkeitsbeziehungen unter den Variablen sehr gut. Falls man dagegen Verbindungen von Effekten zu Ursachen erstellt, so führt dies häufig dazu, dass auch ansonsten unabhängige Ursachen oder Effekte verbunden werden müssen, um die korrekten Einflüsse zu erhalten.

Würde man im Netzwerk auf Seite 8 die Richtung der Kanten invertieren, so würde beispielsweise die Unabhängigkeit von *Burglary* und *Earthquake* für den Fall, dass keine weitere Evidenz vorliegt, verlorengehen. Umgekehrt müsste man eine zusätzliche Kante zwischen *JohnCalls* und *MaryCalls* einführen, um zu modellieren, dass sich die Wahrscheinlichkeit für einen Anruf von Mary ändert, falls John anruft und sonst nichts bekannt ist (weil die Wahrscheinlichkeit für einen Alarm steigt). Zudem wäre die Bestimmung der CPTs deutlich schwieriger – während die Wahrscheinlichkeit für einen

Alarm im Falle eines Erdbebens wohl vom Hersteller der Alarmanlage angegeben wird, muss die Wahrscheinlichkeit eines Erdbebens im Falle eines Alarms erst anderweitig berechnet werden.

Natürlich kann nicht jedes Bayessche Netz vollkommen aus Kausalbeziehungen bestehen. Man sollte jedoch auch ansonsten immer darauf achten, möglichst wenig Parameter zu erzeugen. Falls das Netzwerk ein Polybaum bleiben kann, beschleunigt dies außerdem die Berechnung der Inferenz. Glücklicherweise ergibt eine intuitive Anordnung der Knoten nach dem Kriterium „welche Variable beeinflusst direkt welche andere Variable“ häufig bereits eine gute Anfangsstruktur.

Falls man schon von Anfang an die Variablen fest vorliegen hat und ihre Abhängigkeiten kennt, so kann man die Struktur eines korrespondierenden Netzwerks bestimmen, indem man stets einzelne Knoten hinzufügt und die Elternmenge als eine minimale Knotenmenge bestimmt, so dass die Gleichung

$$\mathbf{P}(X_i|X_{i-1}, \dots, X_1) = \mathbf{P}(X_i|\text{parents}(X_i))$$

erfüllt bleibt. Sinnvollerweise beginnt man dabei mit den Variablen, die nicht von anderen Variablen beeinflusst werden, fährt mit den Variablen fort, die durch sie beeinflusst werden, usw. bis zu den Variablen, die keine weiteren Variablen mehr beeinflussen (den Blättern im Netz). Diese Methode liefert auf jeden Fall ein korrektes Bayessches Netz, das zudem häufig einen geringen Verzweigungsgrad aufweist.

2.5 Inferenz

Bayessche Netze können eine Reihe von Fragestellungen beantworten, indem man ausgehend von bestimmter Evidenz die Wahrscheinlichkeiten der Zustände der anderen Knoten berechnet, was als *Inferenz* bezeichnet wird. Je nach Richtung des Informationsflusses unterscheidet man verschiedene Arten Inferenz, obwohl die Algorithmen stets die gleichen sind:

1. Diagnostische Inferenz ermittelt Gründe für bestimmte Effekte, läuft also entgegen der Kantenrichtung
2. Kausale Inferenz dagegen schließt aus Gründen auf Effekte, läuft also mit der Kantenrichtung
3. Interkausale Inferenz („wegerklären“) verläuft zwischen Gründen eines bestimmten Effektes (z.B. sinkt im Netz aus Abbildung 2.1 bei gegebenem Alarm die Wahrscheinlichkeit für einen Einbruch, wenn man weiß, dass es ein Erdbeben gab)
4. Gemischte Inferenz schließlich beinhaltet mehrere der obigen Arten

Wie bereits erwähnt, enthält ein Bayessches Netzwerk die Wahrscheinlichkeiten für alle seine Variablen. Sie können theoretisch über Formel 2.1 folgendermaßen bestimmt werden:

$$P(x_i|x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = \frac{P(x_1, \dots, x_n)}{P(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)} = \frac{P(x_1, \dots, x_n)}{\sum_{X_i} P(x_1, \dots, x_n)}$$

Dies wird für größere Netze jedoch rasch zu aufwändig, sodass bessere Algorithmen benötigt werden. Leider hat sich herausgestellt, dass das Inferenzproblem in Bayesschen Netzwerken im allgemeinen Fall NP-hart ist [Coo90]. Dies liegt hauptsächlich an Problemen mit ungerichteten Zyklen im Netzwerk (damit sind Zyklen gemeint, die der ungerichtete Graph hat, der aus dem Bayesschen Netzwerk hervorgeht, wenn man die Richtung der Kanten nicht mehr beachtet). Für Netzwerke ohne solche Zyklen – also in Polybaumstruktur – hat Pearl schon früh einen Algorithmus entwickelt, der linear in der Anzahl der Knoten ist (siehe [Pea88]). Eine Variante dieses Algorithmus ist in Abschnitt 3.3.1 detailliert beschrieben.

Aber auch in Netzwerken, die keine Polybaumstruktur haben, kann exakte Inferenz häufig noch sehr rasch durchgeführt werden. Der derzeitige Standardalgorithmus wurde von Lauritzen und Spiegelhalter entwickelt (siehe [LS88]) und später noch weiter verfeinert. Er beruht auf der Umwandlung des Netzes in einen Baum aus den Cliques eines Hilfsgraphen; das Verfahren ist genauer beschrieben in Abschnitt 3.3.2.

Des Weiteren gibt es für Netzwerke mit ungerichteten Zyklen auch eine Reihe von Algorithmen, die nur eine näherungsweise Berechnung der Wahrscheinlichkeiten durchführen. Diese beruhen üblicherweise auf stochastischer Simulation (*logic sampling*, *likelihood weighting* und *backward sampling* sind unterschiedliche Vertreter). Das Prinzip ist, sehr viele Tests durchzuführen, bei denen zu jedem Knoten im Netzwerk nacheinander ein Zustand festgelegt wird. Im einfachsten Fall beginnt man bei den Wurzelknoten und legt deren Zustand zufällig fest, wobei jeder Zustand genau seine a-priori-Wahrscheinlichkeit hat, genommen zu werden. Dann kann man bei den direkten Kindern der Wurzelknoten weitermachen, wobei jeder Zustand nun mit der Wahrscheinlichkeit genommen wird, die in der CPT des Knotens für die bereits festgelegten Elternzustände angegeben ist. Auf diese Weise fährt man im restlichen Netzwerk fort. Nach genügend vielen Durchläufen kann man dann die Wahrscheinlichkeiten der einzelnen Knoten unter bestimmter Evidenz durch einfaches Zählen annähernd bestimmen. Stochastische Verfahren sind normalerweise ziemlich aufwändig, liefern keine reproduzierbaren Ergebnisse und benötigen insbesondere für unwahrscheinliche Zustände sehr viele Durchläufe. Daher sind sie im RoboCup nicht zu gebrauchen.

2.6 Lernverfahren

Das automatisierte Erlernen Bayesscher Netzwerke durch den Rechner teilt sich je nach bekannten Informationen in vier große Teilgebiete auf:

1. Erlernen der Parameter eines Netzwerks mit bekannter Struktur, in dem alle Knoten direkt beobachtbar sind

2. Erlernen der Parameter eines Netzwerks mit bekannter Struktur, in dem einige sog. „versteckte“ Knoten nicht beobachtbar sind
3. Erlernen sowohl der Parameter als auch der Struktur eines Netzwerks, in dem alle Knoten beobachtbar sind
4. Erlernen sowohl der Parameter als auch der Struktur eines Netzwerks mit „versteckten“ Knoten

Die Parameter sind dabei die Werte in den CPTs. „Beobachtbar“ bedeutet, dass in der vorgegebenen Beispielsammlung oder Datenbank für jedes Beispiel der konkrete Zustand des Knotens vorhanden ist.

Der erste Fall ist – da alle Informationen vorliegen – der einfachste, und ein direkter statistischer Ansatz führt hier häufig bereits zum Erfolg. So kann man beispielsweise eine Dirichlet-Funktion für die Parameter ansetzen. Unter der Annahme, dass die einzelnen Parameter unabhängig voneinander sind, berechnet man dann die neuen Schätzungen für die Parameter durch Zählen der Vorkommen ihrer Zustandskombinationen (beschrieben beispielsweise in [Hec95]). Auch Verfahren wie lineare Regression oder probabilistische Entscheidungsbäume werden zum Lernen der Parameter eingesetzt.

Der letzte Fall ist umgekehrt der schwierigste, und bis heute sind noch keine guten universellen Algorithmen bekannt, die das Problem lösen. Der dritte Fall ist gut untersucht und weiterhin aktuelles Forschungsthema; eine sehr gute Übersicht über vorhandene Algorithmen bieten [Kra98] und [Hec95]. Weitere Literatur zu speziell diesem Fall ist in [Bun96] angesprochen.

Der zweite Fall – erlernen der Parameter eines Netzwerks mit „versteckten“ Knoten – ist derjenige, der für die Anwendung im RoboCup-Szenario am interessantesten ist. Dies liegt an mehreren Gründen:

- Es ist (mit etwas Anwendungswissen) für einen Menschen nicht allzu schwer, eine sinnvolle Netzwerkstruktur bereits am Anfang vorzugeben, solange die Komplexität des Netzes überschaubar ist. Wie in Abschnitt 2.4 angesprochen, gibt es einige Techniken, die diese Aufgabe erleichtern; zusätzlich lassen sich spezielle Techniken finden, die im RoboCup-Szenario besonders nützlich sind (erläutert in Kapitel 4 ab Seite 44).
- Soll Verstärkungslernen angewendet werden, so ist es sogar erforderlich, von Anfang an eine Netzwerkstruktur vorliegen zu haben, da sonst die ersten Entscheidungen gar nicht getroffen werden können.
- Praktische Gründe der Implementierung sprechen ebenfalls dafür, die Struktur von Anfang an vorzugeben. Andernfalls müsste eventuell nach jedem Lernvorgang die Einbindung des Netzes in die vorhandene Architektur geändert werden.
- Schließlich bringen sog. innere Knoten große Vorteile hinsichtlich Effizienz und Übersichtlichkeit, wie in Abschnitt 4.2.3 erläutert wird. Gerade diese inneren Knoten sind aber im RoboCup üblicherweise nicht beobachtbar.

Für das Problem wurden eine Reihe von Algorithmen entwickelt (für eine Übersicht siehe z.B. [Kra98]). Die wichtigsten sind stochastische Lernverfahren, insbesondere Monte-Carlo Methoden, der *Expectation Maximisation (EM)*-Algorithmus und Gradientenabstieg. Monte-Carlo Methoden, deren wichtigster Vertreter das *Gibbs-Sampling* ist (siehe z.B. [Bun94]), haben den Vorteil, dass sie theoretisch das globale Optimum finden; jedoch sind sie eher langsam und benötigen zudem vergleichsweise große Datenmengen. Der EM-Algorithmus und Gradientenabstiegsverfahren können dagegen in lokalen Optima hängenbleiben. In beiden Fällen wird eine *maximum a posteriori* Schätzung für die Parameter bestimmt, und in beiden Fällen können die benötigten Daten in einem Bayesschen Netzwerk knotenlokal berechnet werden. Der EM-Algorithmus ([DLR77]) geht in zwei konzeptionell (wenn auch nicht unbedingt implementierungstechnisch) getrennten Schritten vor, eben der Berechnung von erwarteten Werten („expectation“) und der darauf folgenden Maximierung der Parameter („maximisation“). Er ist für viele Probleme gut geeignet, hat aber Schwierigkeiten mit kleinen Datenmengen – insbesondere, falls für manche Fälle gar keine Beispiele vorhanden sind – und ist langsam nahe des Optimums. Gradientenabstiegsverfahren sind vor allem aus dem Bereich der Neuronalen Netze bekannt und dort sehr gut untersucht. Sie erscheinen für die Anwendung im RoboCup und insbesondere für das Verstärkungslernen besonders geeignet, weil man sehr einfach Anfangsschätzungen für die Parameter angeben kann und sie auch mit kleinen Datenmengen zurecht kommen. Details zum Gradientenabstieg folgen in Abschnitt 3.4.

Schließlich gibt es auch noch deterministische Lernverfahren (untersucht wurden einige Varianten z.B. in [CDS96]), beispielsweise das sog. *fractional updating*. Sie nehmen an, dass die einzelnen Parameter voneinander unabhängig sind und einer Dirichlet-Verteilung folgen. Diese Verfahren werden seltener benutzt, haben jedoch in [CDS96] gute Ergebnisse erzielt.

2.7 Bewertung

Bayessche Netze sind ein exzellentes Mittel, um in Situationen mit unsicherem Wissen Schlussfolgerungen zu ziehen und Entscheidungen zu treffen. Ihre Vorteile insbesondere in der Robotik sind:

- Inhärente Modellierung der Unsicherheiten und Wahrscheinlichkeiten
- Kompakte und verständliche Repräsentation von komplexen Wahrscheinlichkeitsfunktionen
- Modellierung von Kausalbeziehungen und probabilistischer Unabhängigkeit
- Einfache Modellierung von nicht idealen Sensoren (siehe Abschnitt 4.2.2 in Kapitel 4)
- Als DBNs flexible Beherrschung auch zeitlich abhängiger Zustände

- Möglichkeit des automatischen Lernens, mit einigen Verfahren auch Verstärkungslernen

Dem stehen freilich auch einige Nachteile entgegen. Die wichtigsten davon sind:

- Inferenz kann – insbesondere in großen Netzwerken mit vielen ungerichteten Zyklen – lange dauern.
- Eine direkte Verarbeitung kontinuierlicher Werte ist derzeit mit Bayesschen Netzwerken noch schwierig.
- Die Konstruktion von Netzwerken mit guter Struktur ist nicht trivial. Schlecht konstruierte Netzwerke sind jedoch eventuell nicht in der Lage, die gestellte Aufgabe zu lösen.
- Es ist eher aufwändig, die Struktur eines bestehenden Netzwerks zu ändern, falls sich äußere Gegebenheiten geändert haben.

Insbesondere beim Lernen Bayesscher Netze mit Hilfe von Gradientenabstiegsverfahren drängt sich ein Vergleich mit Neuronalen Netzen auf, da die mathematische Methode und die graphische Struktur sehr ähnlich erscheinen. Dennoch bestehen große Unterschiede zwischen den beiden (u.a. angesprochen in [RN95]):

- Bayessche Netze sind eine *lokale* Repräsentation der Wahrscheinlichkeitsfunktion. Das bedeutet, dass man in sie „hineinschauen“ kann: man kann die CPTs der einzelnen Knoten betrachten und überlegen, ob die Werte sinnvoll sind, und man kann erkennen, wodurch bestimmte Ausgaben erzeugt werden. Neuronale Netze dagegen repräsentieren ihre Funktion als Ganzes, und die Werte bei einzelnen Neuronen sind für sich genommen bedeutungslos.
- Inferenz bei Bayesschen Netzen ist deutlich langsamer und ist im allgemeinen Fall sogar NP-hart. Andererseits können Bayessche Netze manchmal kleiner sein als die entsprechenden Neuronalen Netze.
- Vor dem Training lassen sich in Bayesschen Netzen sinnvolle Vorgaben für die CPTs machen und so Startwerte festlegen, die die Anzahl der benötigten Beispiele stark reduzieren. Hinsichtlich des Verstärkungslernens kann ein Bayessches Netz so auch im untrainierten Zustand schon halbwegs vernünftige Ergebnisse liefern.
- Da das Training Bayesscher Netze beim Gradientenabstieg auf Inferenz beruht, braucht es pro Beispiel mehr Zeit.
- Bayessche Netze haben eine zweidimensionale Ausgabe: einerseits die Zustände, andererseits die Wahrscheinlichkeiten für jeden Zustand.
- Bayessche Netze können Inferenz in jede Richtung durchführen, während bei Neuronalen Netzen Ein- und Ausgabevariablen fest sind.

Die Vorteile der Bayesschen Netze sind daher insbesondere dann groß, wenn sich schon die Problemstellung in lokale Probleme aufteilen lässt. Dies ist z.B. häufig beim automatischen Schlussfolgern und Entscheiden der Fall. Neuronale Netze dagegen sind besser für Probleme geeignet, die als Ganzes betrachtet werden müssen, wie beispielsweise Bilderkennung und -verstehen, und für Probleme mit sehr vielen Eingabevariablen.

Kapitel 3

Bibliothek für Bayessche Netze

3.1 Anforderungen und vorhandene Bibliotheken

Um Bayessche Netze für den RoboCup nutzen zu können, ist es unabdingbar, eine Bibliothek zur Verfügung zu haben, die gewisse Grundvoraussetzungen erfüllt. Dazu gehört nicht nur der Umgang mit Bayesschen Netzen allgemein: wie in Abschnitt 4.1 erläutert, stellt auch der RoboCup als Szenario besondere Anforderungen. Die wichtigsten Anforderungen, die die zu benutzende Bibliothek auf jeden Fall erfüllen muss, lauten:

- Erstellen, Verändern, Laden und Speichern von Bayesschen Netzen
- Exakte und schnelle Inferenz, also Unterstützung zumindest des Cliquesbaum-Algorithmus (siehe Abschnitt 2.5)
- Lernen von Parametern mittels Gradientenabstieg (siehe Abschnitt 2.6)
- Sicherheitsmechanismen für die Verwendung mit mehreren Threads
- C/C++ - API, verwendbar unter Linux mit gcc 2.95 und gcc 3.2, um kompatibel zur schon vorhandenen Software zu sein

Die generelle Anwendbarkeit von Bayesschen Netzen in Echtzeit-Anwendungen hinsichtlich ihrer Inferenzgeschwindigkeit wurde in [You98] untersucht und für gegeben erachtet. Im Internet sind eine ganze Reihe von Bibliotheken für Bayessche Netze frei verfügbar (einen Überblick gibt z.B. [bns]). Jedoch sind die meisten wie beispielsweise Pulcinella [pul] oder JavaBayes [jvb] für andere Programmiersprachen oder andere Betriebssysteme; Testversionen kommerzieller Systeme wie Netica [ntc] sind eingeschränkt bezüglich Benutzungsdauer oder Anzahl verwendbarer Knoten. Von der University of Pittsburgh existiert eine *smile* genannte Bibliothek [smi], die viele Anforderungen erfüllt und die für die meisten Untersuchungen hier verwendet wurde, bis die eigene Bibliothek fertiggestellt war (danach konnte sie noch als Referenz für Testergebnisse dienen). Sie unterstützt alle wichtigen Inferenzalgorithmen, und es gibt auch

einen einfachen grafischen Editor für Netzwerke dazu (diesen allerdings nur unter Windows). Smile hat aber keine Unterstützung für Lernverfahren und keine Threadsicherheit, sodass zunächst einige Eigenimplementierungen und Proxys notwendig wurden. Sie hat aber auch einige schwerwiegendere Nachteile: der Source-Code ist nicht verfügbar, und die verwendete Version hat ein größeres Speicherleck im Inferenzalgorithmus (es existiert eine neuere Version, die allerdings nicht mehr mit gcc 2.95 kompatibel ist). Daher wurde im Rahmen dieser Diplomarbeit eine eigene Bibliothek implementiert.

In den folgenden Abschnitten werden nun die Architektur dieser Bibliothek sowie die implementierten Inferenz- und Lernalgorithmen genauer erläutert.

3.2 Architektur

Zunächst wurde eine kleine Bibliothek entwickelt, die fast nur Proxy für die Smile-Bibliothek ist. Um später auf einfache Weise zwischen den Bibliotheken wechseln zu können, wurden für alle Klassen abstrakte Schnittstellen definiert und das Muster „Abstrakte Fabrik“ (siehe [GHJV96], wo auch viele andere hier erwähnte Entwurfsmuster beschrieben werden) verwendet, um die konkreten Instanzen für die jeweilige Implementierung zu erzeugen.

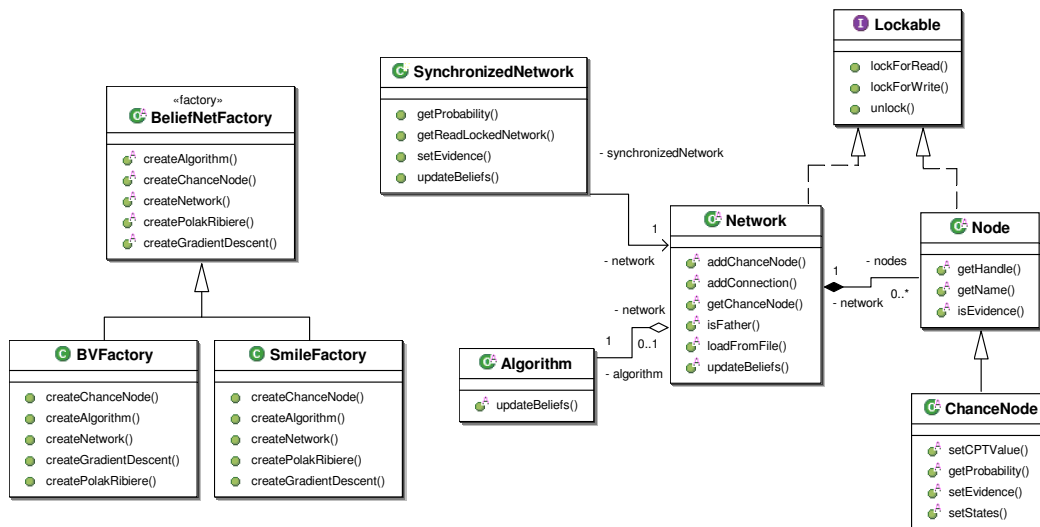


Abbildung 3.1: Basisarchitektur der Bibliotheken

Abbildung 3.1 zeigt die Basisarchitektur der Bibliotheken. Hierbei sind von den jeweiligen Klassen die Methoden nur auszugsweise aufgeführt, um einen Eindruck von der Funktion zu geben. Zu den abstrakten Klassen wie `Network` und `ChanceNode` gibt es dann jeweils eine konkrete Klasse in jeder Bibliothek, die von der entsprechenden Fabrik instantiiert wird. Um die Bibliothek auf einfache Weise thread-sicher zu machen, wurde die Klasse `SynchronizedNetwork` erstellt, die Netzwerk oder Knoten richtig sperrt und für die während des Spiels am häufigsten benutzten Funktionen Abkürzungsmethoden bereitstellt. Knoten können nicht unabhängig von Netzwerken

existieren, sondern sind stets einem Netzwerk zugeordnet, was die Verwaltung vereinfacht. Neben ChanceNode sind problemlos weitere Knotenarten z.B. für Entscheidungs- oder Nützlichkeitsknoten denkbar.

Die Implementierung der eigenen „BV“¹ - Bibliothek, deren wichtigste Klassen Abbildung 3.2 zeigt, beruht maßgeblich auf der generischen Klasse *DynamicMatrix*. In ihren Instanzen werden die Tabellen der CPTs gespeichert, wobei fast alle Zugriffsmethoden von der konkreten Speicherart abstrahieren². Für diese Speicherart wurde ein lineares Array gewählt, in dem die Felder der Matrix seriell abgespeichert sind. Da bei den Inferenzalgorithmen ein komplettes seriell Durchlaufen der Matrizen gegenüber häufigen wahlfreien Zugriffen bei geschickter Implementierung sehr viel häufiger vorkommt, ist diese Wahl hinsichtlich der Laufzeit optimal.

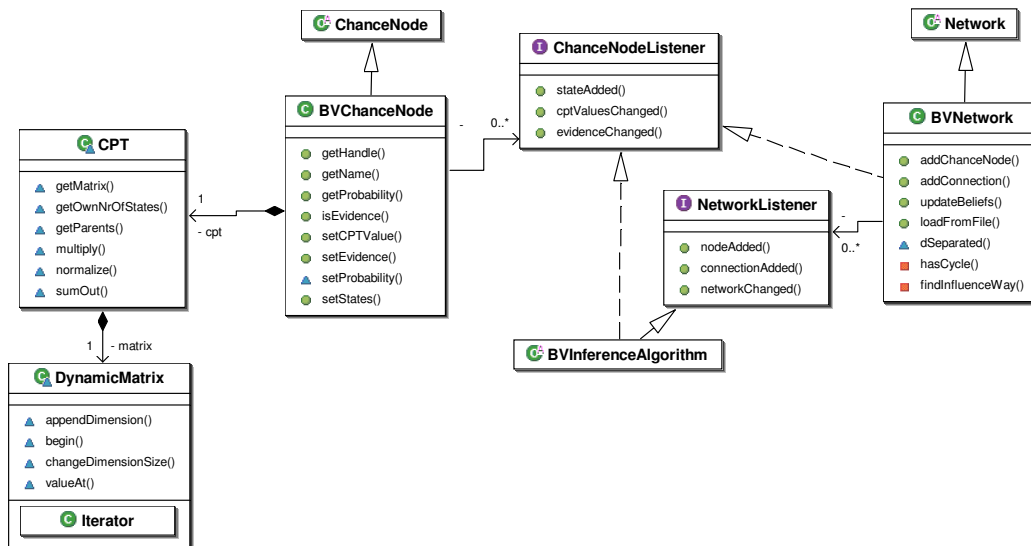


Abbildung 3.2: Struktur der „BV“ - Bibliothek

Abfragen der Netzstruktur sind zur Verwendungszeit eher ungewöhnlich. Daher wurde darauf verzichtet, eine beschleunigte Speicherstruktur wie z.B. eine Adjazenzmatrix zu verwenden, um den Graphen des Netzwerks zu repräsentieren. Statt dessen halten die Objekte des Typs *BVNetwork* einfach eine Liste der vorhandenen Knoten und zwei *maps*, in denen zu jedem Knoten eine Liste seiner Eltern und eine Liste seiner Kinder abgelegt sind (entspricht doppelten Adjazenzlisten). Das Beobachter-Muster (siehe [GHJV96]) wurde zweifach verwendet, um das Framework erweiterbar zu machen und *BVNetwork* sowie *BVChanceNode* von den Klassen zu entkoppeln, die von Änderungen an den entsprechenden Daten informiert werden müssen. Prominente Beispiele für solche Beobachter sind die Inferenzalgorithmen; z.B. muss der Cliquesbaum neu gebaut werden, wenn sich die Netzwerkstruktur ändert.

¹Für die Abteilung Bildverstehen

²Intern gibt es ein paar direkte Zugriffsmöglichkeiten, da diese die Inferenz drastisch beschleunigen - siehe Abschnitt 3.3.2.4

Zur Verbindung eines Netzwerkes mit Daten aus dem Weltmodell wurden Adapterklassen erstellt, die auf der einen Seite Weltmodelldaten (falls nötig) diskretisieren und einem Knoten als Evidenz übergeben, auf der anderen Seite Wahrscheinlichkeiten eines Knotens oder den Index seines wahrscheinlichsten Zustands in ein Objekt im Weltmodell schreiben können. Abbildung 3.3 zeigt die Klassen und ihr Funktionsprinzip. *WoMoObjectNode* stellt die 1:1-Beziehung zwischen Knoten und Weltmodell-Objekt her, mit der von einem Netz berechnete Daten im Weltmodell gespeichert werden können. *GeneralSensorNode* kann allgemein für die umgekehrte Verbindung von Weltmodell zu Wahrscheinlichkeitsknoten benutzt werden, während *RobotVelocity* und eine Reihe ähnlicher Klassen spezielle Ausprägungen für bestimmte Weltmodelldaten sind. Adapter können auch auf mehr als ein Objekt im Weltmodell zugreifen, wobei es allerdings meist keinen Sinn hat, sich als Observer für mehr als eines (dasjenige, das häufiger geändert wird) anzumelden.

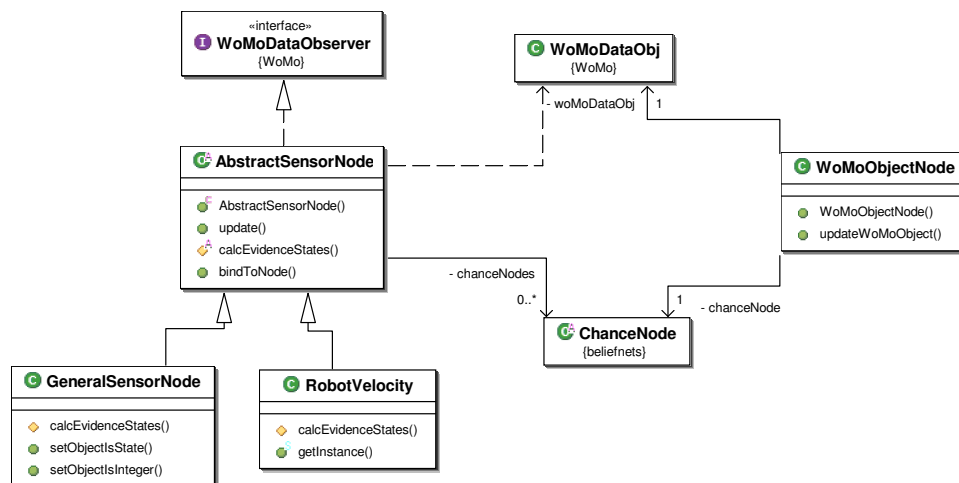


Abbildung 3.3: Adapter zwischen Weltmodell und Bayesschen Netzen

Um den Berechnungsaufwand in der update-Schleife des Weltmodells gering zu halten, wurden mehrere Maßnahmen getroffen. Zunächst berechnen die Adapter teilweise aus den gleichen Weltmodell-Daten mehrere diskrete Werte (beispielsweise aus den Hindernisinformationen den Abstand des nächsten Hindernisses in X- und in Y-Richtung). Bei der Verbindung des Adapters mit einem Wahrscheinlichkeitsknoten wird dann angegeben, welcher Wert als Evidenz für den Knoten gelten soll. Außerdem sollte es für jede Berechnung nur eine Instanz des Adapters geben, an der alle Wahrscheinlichkeitsknoten, die diese Daten modellieren, angebunden sind; und schließlich wird nur die Evidenz im Knoten geändert, aber noch keine Inferenz durchgeführt.

Eine detaillierte Anleitung zur Benutzung der Bibliothek findet sich in Anhang A ab Seite 69.

3.3 Inferenz

3.3.1 Inferenz in Polybäumen

Falls das Bayessche Netz die Form eines Polybaumes hat – also keine ungerichteten Zyklen enthält –, kann Inferenz mit Hilfe eines rekursiven Algorithmus durchgeführt werden, der eine Abart des ursprünglich von Pearl [Pea88] in nachrichtenbasierter Form (d.h., jeder Knoten schickt bestimmte Nachrichten an seine Nachbarn) entwickelten Verfahrens ist und in [RN95] beschrieben wird.

3.3.1.1 Herleitung

Das grundlegende Prinzip ist, die auf einen bestimmten Zustand eines Knotens X hindeutende Evidenz in zwei Teilevidenzen aufzuspalten: einerseits den „causal support“ E_X^+ , der von allen Knoten kommt, die im Baum oberhalb von X liegen, und andererseits den „evidential support“ E_X^- , der von allen Knoten kommt, die unterhalb von X liegen. Da das Netz ein Polybaum ist, sind diese beiden Mengen d-separiert (veranschaulicht in Abbildung 3.4). Somit gilt (siehe Abschnitt 2.1):

$$P(x|E_X) = P(x|E_X^+, E_X^-) = \frac{P(E_X^-|x, E_X^+)P(x|E_X^+)}{P(E_X^-|E_X^+)} = \alpha P(E_X^-|x)P(x|E_X^+) \quad (3.1)$$

Man betrachte zunächst $P(x|E_X^+)$. Sei \mathbf{U} der Vektor der Elternknoten von X und \mathbf{u} eine bestimmte Kombination von Zuständen der Eltern. Dann kann die Wahrscheinlichkeit eines Zustandes von X durch Mittelung über die Elternzustände – gewichtet mit dem jeweiligen CPT-Eintrag – bestimmt werden:

$$P(x|E_X^+) = \sum_{\mathbf{u}} P(x|\mathbf{u}, E_X^+) P(\mathbf{u}|E_X^+)$$

Da X unabhängig von $E_X^+ \setminus \mathbf{U}$ ist, sofern die Elternzustände festliegen, und da wegen der Polybaumstruktur auch die Elternknoten alle voneinander unabhängig sind, wenn nur E_X^+ betrachtet wird, lässt sich dies vereinfachen zu:

$$P(x|E_X^+) = \sum_{\mathbf{u}} P(x|\mathbf{u}) \prod_i P(u_i|E_{u_i \setminus X}) \quad (3.2)$$

Dabei ist $P(x|\mathbf{u})$ ein Eintrag in der CPT von X , und $P(u_i|E_{u_i \setminus X})$ ist eine rekursive Formulierung der Berechnung für einen Elternknoten³.

Die Berechnung von $P(E_X^-|x)$ ist etwas komplizierter. Sei Y die Menge der Kinder von X (bei gegebenem x sind diese voneinander unabhängig) und Z_i die Menge der Eltern

³*Notation:* Die Menge $E_{A \setminus B}$ sei die Evidenz, die mit A verbunden ist, aber nicht über B .

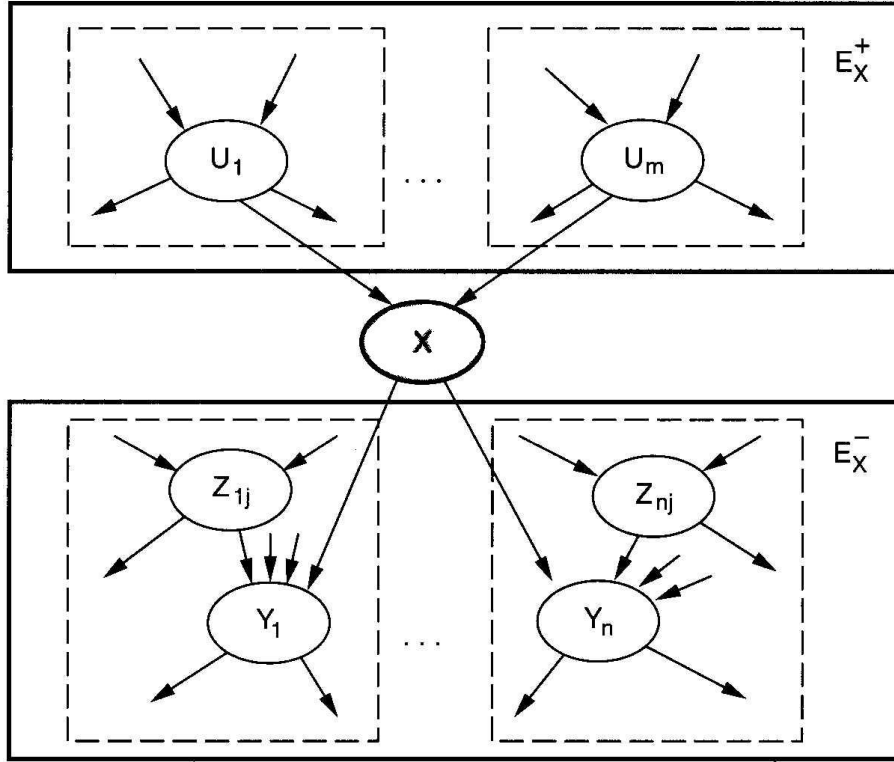


Abbildung 3.4: Ausschnitt aus einem Bayesschen Netz in Polybaumstruktur (aus [RN95])

von Y_i ohne X . Dann lässt sich die gesuchte Wahrscheinlichkeit als Mittel über die Wahrscheinlichkeiten von Y_k errechnen.

$$P(E_X^-|x) = \prod_Y P(E_{Y_k \setminus X}|x) = \prod_Y \sum_{Y_k} \sum_{\mathbf{z}_k} P(E_{Y_k \setminus X}|x, y_{k_i}, \mathbf{z}_k) P(y_{k_i}, \mathbf{z}_k|x)$$

bei Mittelung über die Zustände der Kinder und alle Kombinationen der Zustände von deren Elternknoten. $E_{Y_k \setminus X}$ spaltet man wiederum auf in die Komponenten $E_{Y_k \setminus X}^+$ und $E_{Y_k \setminus X}^-$. Dabei ist $E_{Y_k \setminus X}^+$ unabhängig von X und Y_k , und $E_{Y_k \setminus X}^-$ ist unabhängig von X und Z_k .

$$P(E_X^-|x) = \prod_Y \sum_{Y_k} P(E_{Y_k \setminus X}^-|y_{k_i}) \sum_{\mathbf{z}_k} P(E_{Y_k \setminus X}^+|\mathbf{z}_k) P(y_{k_i}, \mathbf{z}_k|x)$$

Mit einigen einfachen Umrechnungen und Normalisierung führt dies zur endgültigen Formel:

$$P(E_X^-|x) = \beta \prod_Y \sum_{Y_k} P(E_{Y_k \setminus X}^-|y_{k_i}) \sum_{\mathbf{z}_k} P(y_{k_i}|x, \mathbf{z}_k) \prod_{\mathbf{z}_k} P(z_{k_l}|E_{Z_{k_l} \setminus Y_k}) \quad (3.3)$$

Dabei ist $P(E_{Y_k}^-|y_{k_i})$ eine rekursive Berechnung für einen Kindknoten, $P(y_{k_i}|x, \mathbf{z}_k)$ ist ein Eintrag in der CPT von Y_k , und $P(z_{k_i}|E_{Z_{k_i}\setminus Y_k})$ ist wiederum eine rekursive Formulierung des ursprünglichen Problems.

Falls ein Kind Y_k von X Evidenz trägt (bei Zustand e), so ist $P(E_{Y_k}^-|y_{k_i})$ Null für $i \neq e$ und Eins für $i = e$.

3.3.1.2 Algorithmus

Der Inferenz-Algorithmus für Polybäume geht direkt aus den oben angegebenen Formeln 3.2 und 3.3 hervor. Er besteht aus zwei Funktionen:

- `supportExcept(X, V)` berechnet für alle Zustände von X $P(x|E_{X\setminus V})$
- `evidenceExcept(X, V)` berechnet für alle Zustände von X $P(E_{X\setminus V}^-|x)$

Diese beiden Funktionen werden im Folgenden als Pseudo-Code angegeben. Eine deutliche Beschleunigung kann durch Zwischenspeichern schon berechneter Ergebnisse geschehen. Um die Wahrscheinlichkeiten für das gesamte Netzwerk zu bestimmen, wird für jeden Knoten `supportExcept(X, null)` aufgerufen.

```
function supportExcept(X, V) :  $\mathbf{P}(X|E_{X\setminus V})$ 
  if X.hasEvidence return  $\mathbf{P}(X)$ ;
   $\mathbf{P}(E_{X\setminus V}^-|\mathbf{X}) := \text{evidenceExcept}(X, V)$ ;
  U := parents(X);
  if U.isEmpty return  $\alpha \mathbf{P}(E_{X\setminus V}^-|x) \mathbf{P}(X)$ ;
  for each  $U_i$  in U do  $\mathbf{P}(U_i|E_{U_i\setminus X}) := \text{supportExcept}(U_i, X)$ ;
  return  $\alpha \mathbf{P}(E_{X\setminus V}^-|\mathbf{X}) \sum_{\mathbf{u}} \mathbf{P}(X|\mathbf{u}) \prod_i P(u_i|E_{u_i\setminus X})$ 
end function
```

```
function evidenceExcept(X, V) :  $\mathbf{P}(E_{X\setminus V}^-|\mathbf{X})$ 
  Y := children(X) \ V;
  if Y.isEmpty return Gleichverteilung;
  for each  $Y_i$  in Y do
    if not  $Y_i$ .hasEvidence  $\mathbf{P}(E_{Y_i}^-|\mathbf{Y}_i) := \text{evidenceExcept}(Y_i, \text{null})$ ;
    else  $\mathbf{P}(E_{Y_i}^-|\mathbf{Y}_i) = \mathbf{P}(Y_i)$ ; // 1 bei Evidenzzustand, sonst 0
     $Z_i := \text{parents}(Y_i) \setminus X$ ;
    for each  $Z_{i_j}$  in  $Z_i$  do  $\mathbf{P}(Z_{i_j}|E_{Z_{i_j}\setminus Y_i}) = \text{supportExcept}(Z_{i_j}, Y_i)$ ;
  end for;
  for each x do
     $P(E_{X\setminus V}^-|x) := \beta \prod_Y \sum_{Y_k} P(E_{Y_k}^-|y_{k_i}) \sum_{\mathbf{z}_k} P(y_{k_i}|x, \mathbf{z}_k) \prod_{\mathbf{z}_k} P(z_{k_i}|E_{Z_{k_i}\setminus Y_k})$ 
  end for;
  return Vektor aus berechneten Werten
end function
```

3.3.1.3 Aufwandsabschätzung

Sowohl `supportExcept` als auch `evidenceExcept` werden, falls ihre Ergebnisse zwischengespeichert werden, bei der Inferenz je einmal pro Knoten (mit dem zweiten Argument null) und einmal pro Kante (mit den beteiligten Knoten als Argumenten) aufgerufen. In `supportExcept` werden dabei für die Summation so viele Rechenschritte durchgeführt, wie die CPT des Knotens Einträge besitzt, und für die Berechnung der Produkte nochmals so viele, aber geteilt durch die Anzahl der Zustände des Knotens. Sei die Knotenzahl n , die Kantenzahl e , die Anzahl Zustände des Knotens X s_x und die Anzahl der Einträge in der CPT des Knotens X c_x , so liegt der Aufwand für `supportExcept` insgesamt also bei

$$O\left(\sum_{X=X_1}^{X_n} \left(c_x + \frac{c_x}{s_x}\right) + \sum_E \left(c_{x_e} + \frac{c_{x_e}}{s_{x_e}}\right)\right) = O((n + e) * c)$$

mit $c = \sum_i c_i$ als der Gesamtanzahl der Variablen im Netzwerk.

Die abschließende Berechnung in `evidenceExcept` hat den gleichen Aufwand noch einmal, allerdings multipliziert mit der Summe der Zustandszahlen der Kinder des Knotens (siehe Formel 3.3). Sei r die größte Summe dieser Zustandszahlen für alle Elternknoten. Dann ist der Aufwand des Algorithmus insgesamt (da das Netzwerk ein Polybaum sein muss)

$$O((n + e) * c + (n + r * e) * c) = O((n + r * e) * c) = O(r * n * c)$$

Auf einem Rechner mit Intel Xeon Prozessor mit 3 GHz Taktfrequenz, Linux-Betriebssystem (Kernel 2.4.20) und gcc-Compiler (Version 2.95) benötigte der Algorithmus bei einem Netz mit $n = 3$, $c = 10$ und $r = 2$ durchschnittlich 19.2 Mikrosekunden für einen Inferenzdurchgang. Für ein Netzwerk mit $n = 17$, $c = 996$ und $r = 10$ benötigte er durchschnittlich 1550 Mikrosekunden.

3.3.2 Inferenz in Cliquesbäumen

3.3.2.1 Prinzip

Der Algorithmus zur exakten Inferenz mittels Cliquesbäumen wurde von Lauritzen und Spiegelhalter [LS88] gefunden und ist in [CGH97] und [Nea90] detailliert erläutert. Einen exzellenten Einblick in Optimierungstechniken für eine schnelle Implementierung gibt [HD96].

Das Prinzip des Algorithmus liegt darin, das Bayessche Netzwerk in eine andere Repräsentation zu überführen, den *Cliquesbaum* (*Junction Tree*), in der dann die Inferenz durchgeführt wird. Jeder Clique ist dabei wiederum eine CPT zugeordnet, die (nach der Inferenz) die vereinigte Wahrscheinlichkeitsfunktion aller Knoten in der Clique enthält. Bestimmte mathematische Eigenschaften des Cliquesbaumes ermöglichen es, alle Berechnungen während der Inferenz mit den lokalen Informationen einer Clique

auszuführen, wobei zusätzlich Nachrichten von jeder Clique an ihre Nachbarcliquen geschickt werden. Durch die Baumstruktur werden die Probleme der ungerichteten Zyklen umgangen, an denen der Algorithmus von Pearl scheitert (andererseits können die CPTs der einzelnen Cliques unter ungünstigen Umständen sehr groß werden).

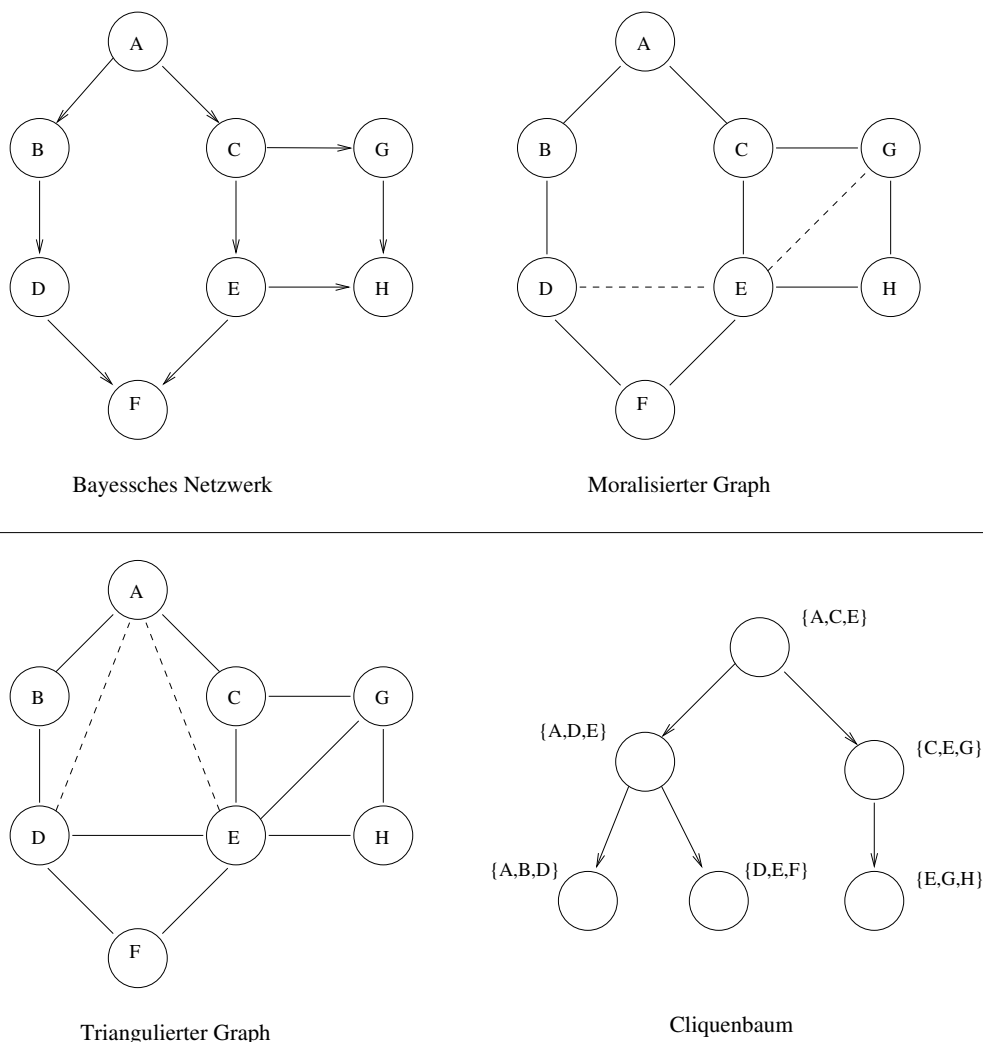


Abbildung 3.5: Erzeugung eines Cliquenbaumes

Um den Cliquenbaum zu erstellen, werden nacheinander folgende Schritte durchgeführt, die in Abbildung 3.5 an einem Beispiel verdeutlicht sind:

1. Das Netzwerk wird „moralisiert“, d.h., für jeden Knoten werden seine Elternknoten untereinander mit einer Kante verbunden
2. Das Netzwerk wird in einen ungerichteten Graphen überführt, indem die Richtung der Kanten nicht mehr beachtet wird
3. Der Graph wird „trianguliert“, d.h. dass im neuen Graphen jeder Zyklus aus mehr als 3 Knoten zumindest eine Kante zwischen zwei seiner Knoten besitzt, die nicht zum Zyklus gehört (eine sog. *Sehne*).

4. Die Cliques des Graphs werden bestimmt
5. Aus den Cliques wird ein Baum gebildet, und zwar so, dass jeder Knoten, der zu zwei verschiedenen Cliques gehört, auch zu jeder Clique auf dem Pfad zwischen diesen beiden Cliques gehört.

Danach wird jeder Wahrscheinlichkeitsknoten des Netzwerks genau einer der Cliques zugeordnet, die ihn enthalten. Für jede Clique C_i bestimmt man eine CPT $\psi_i(C_i) = \prod \{\psi(X) | X \text{ ist } C_i \text{ zugeordnet}\}$ (mit $\psi(X)$ als der CPT des Knotens X). Nun können die Wahrscheinlichkeiten für die Zustände jedes Knotens bestimmt werden, wie im Folgenden erläutert wird.

3.3.2.2 Durchführung von Inferenz

Es sei ein Cliquesbaum wie oben beschrieben vorhanden mit den Cliques C_1 bis C_n und der jeweiligen CPT $\psi_i(C_i)$. Sei $S_{ij} = C_i \cap C_j$ der *Separator* zwischen zwei Cliques C_i und C_j . Sei C_{ij} die Menge der Cliques im Teilbaum von C_i , wenn die Verbindung zwischen C_i und C_j weggelassen wird. Sei weiterhin $R_{ij} = C_{ij} \setminus S_{ij}$. Dann können die Wahrscheinlichkeiten für die Zustände jedes Knotens folgendermaßen bestimmt werden:

1. Berücksichtige die Evidenz in den einzelnen CPTs. Dies kann entweder dadurch geschehen, dass alle Einträge in einer CPT, die zu einem Zustand gehören, der mit der Evidenz unvereinbar ist, auf 0 gesetzt werden, oder dadurch, dass die Evidenzknoten aus den Cliques entfernt und die CPTs entsprechend angepasst werden.
2. Jede Clique sendet an all ihre Nachbarcliques Nachrichten der Form

$$M_{ij}(S_{ij}) = \sum_{R_{ij}} \prod_{C_k \in C_{ij}} \psi_k(C_k)$$

Diese Nachrichten sind ebenfalls Matrizen von gleicher Art wie die CPTs.

3. In jeder Clique wird die neue CPT berechnet als

$$P(C_i) = \psi_i(C_i) \prod_k M_{ki}(S_{ki})$$

4. Die Wahrscheinlichkeiten eines Knotens können nun durch Aussummieren der anderen Knoten bestimmt werden:

$$P(X|e) = \sum_{Y \in C_i \setminus X} P(C_i)$$

Für den Beweis der Korrektheit dieses Verfahrens wird auf die angegebene Literatur verwiesen.

Die Nachrichten M_{ij} lassen sich, wie beispielsweise in [CGH97] angegeben, aus der CPT einer Clique und den von den Nachbarn empfangenen Nachrichten berechnen als

$$M_{ij}(S_{ij}) = \sum_{C_i \setminus S_{ij}} \psi_i(C_i) \prod_{k \neq j} M_{ki}(S_{ki})$$

Das bedeutet, eine Nachricht kann genau dann an einen Nachbarn geschickt werden, wenn die Clique schon selbst alle Nachrichten von ihren anderen Nachbarcliquen erhalten hat. Unter Ausnutzung der Baumstruktur lässt sich die Inferenz also in folgende Schritte aufteilen:

1. Blattcliquen schicken ihre Nachrichten an ihre Mütter. Da Blattcliquen nur einen Nachbarn haben, können diese Nachrichten von Anfang an bestimmt werden.
2. Hat eine Clique alle Nachrichten von ihren Kindern erhalten, so schickt sie eine Nachricht an ihre Mutterclique.
3. Hat die Wurzel alle Nachrichten von ihren Kindern erhalten, so schickt sie ihre Nachrichten an ihre Kinder.
4. Erhält eine Clique eine Nachricht von ihrer Mutterclique, so berechnet sie die Wahrscheinlichkeiten und schickt ihre Nachrichten an ihre Kinder.

Zur einfachen Unterscheidung werden die Nachrichten, die sich im Baum auf die Wurzel zu bewegen, λ_i und die Nachrichten, die sich von der Wurzel weg bewegen, π_i genannt.

3.3.2.3 Der Algorithmus

Der endgültige Algorithmus besteht aus zwei Phasen: einer Baumaufbauphase, die nur einmalig nach einer Änderung der Netzwerkstruktur durchgeführt werden muss, und einer Aktualisierungsphase, die nach jeder Evidenzänderung durchgeführt wird. Da sich bei den Anwendungsszenarien im RoboCup die Netzwerkstruktur während eines Spiels nicht ändert und die Geschwindigkeit der Baumaufbauphase daher nicht besonders kritisch ist, wurde für die Erzeugung des Cliquenbaumes als relativ einfacher Algorithmus eine Variante des *maximum cardinality search fill-in* implementiert, das in [CGH97] beschrieben ist und hier um die direkte Erstellung der Cliquen während der Triangulation erweitert wurde. Dieser Algorithmus lautet im Pseudocode:

```

procedure buildTree
Undirected_Tree tempTree;
for all X in Network do
  insert X into tempTree;
  for all Y in parents(X) do

```

```

insert (X, Y) into tempTree;
for all Z in parents(X) \ Y do
  -- Moralisierung
  if not connected(Z, Y) insert (Z, Y) into tempTree;
od
od
od
list cliqueList;
Clique C;
while es gibt noch nicht zu einer Clique gehörende Knoten do
  X := ein Knoten mit maximaler Anzahl bereits zu
     Cliquen gehörenden Nachbarn;
  N := bereits zu Cliquen gehörende Nachbarn von X;
  if N ist vollständig verbunden then
    if C \ N =  $\emptyset$  then
      C := C  $\cup$  {X};
    else
      cliqueList.append(C);
      C := N  $\cup$  {X};
    fi
  else
    -- Triangulierung
    Mache N zu einer vollständig verbundenen Menge;
    cliqueList :=  $\emptyset$ ;
    C :=  $\emptyset$ ;
  fi
od
cliqueList.append(C); -- letzte Clique
for all C in cliqueList do
  D := diejenige Clique aus den Cliquen in der Liste vor C,
     mit der C die größte Schnittmenge hat;
  parent(C) := D;
od
for all X in Network do
  clique(X) := eine Clique, die X und parents(X) enthält;
od
for all C do
   $\psi(C) := \prod_{\{X \mid \text{clique}(X)=C\}} \psi(X)$ ;
od
end procedure

```

Der Algorithmus für die Inferenz selbst besteht dann wie beschrieben aus mehreren Schritten (die Funktionen propagateLambda und propagatePi sind weiter unten aufgeführt):

1. Für alle Cliques C_i : $\tilde{\psi}_i = \psi_i$ unter Berücksichtigung der Evidenz
2. Für alle Blattcliquen C_i führe `propagateLambda(i)` aus
3. Wenn eine Clique C_i eine Nachricht λ_j erhält:
 - (a) $\tilde{\psi}_i(C_i) = \lambda_j(S_{ji})\tilde{\psi}_i(C_i)$
 - (b) Falls C_i nun λ -Nachrichten von allen Kindern erhalten hat:
 - Falls C_i die Wurzel ist, führe die unten angegebenen Schritte 4 (b) und (c) aus
 - Sonst: $P(C_i) = \tilde{\psi}_i(C_i)$; führe `propagateLambda(i)` aus
4. Wenn eine Clique C_i eine Nachricht π_j erhält:
 - (a) $P(C_i) = \pi_j(S_{ji})\tilde{\psi}_i(C_i)$
 - (b) Berechne $P(X) = \sum_{Y \in C_i \setminus X} P(C_i)$ für alle X mit $\text{clique}(X) = C_i$
 - (c) Falls C_i nicht Blattclique ist, führe `propagatePi(i)` aus

Funktion `propagateLambda(i: Integer)`:

1. $\lambda_i(S_{ij}) = \sum_{C_i \setminus S_{ij}} \tilde{\psi}(C_i)$ (die Mutterclique ist C_j)
2. $\tilde{\psi}_i(C_i) = \frac{\tilde{\psi}_i(C_i)}{\lambda_i(S_{ij})}$ (s. [Nea90])
3. Sende λ_i an C_j

Funktion `propagatePi(i: Integer)`:

Für alle Kinder C_j von C_i :

1. $\pi_i(S_{ij}) = \sum_{C_i \setminus S_{ij}} P(C_i)$
2. Sende π_i an C_j

3.3.2.4 Laufzeit

Der Algorithmus kann mit verschiedenen Maßnahmen stark beschleunigt werden. Zum Einen ist es möglich, schon berechnete Werte weiterzuverwenden. In [Nea90] wird eine Variante vorgestellt, mit der der Berechnungsaufwand bei hinzukommender oder entfernter Evidenz reduziert wird; in leichter Abwandlung wurde diese hier auch dann benutzt, wenn sich die Evidenz eines Knotens geändert hat. Dies ist sehr typisch für die Anwendungsszenarien im RoboCup, denn häufig verändern sich zwischen zwei Abfragen des Netzes nur einige Sensorinformationen leicht, und nur ein oder zwei Knoten wechseln ihren Evidenzzustand.

Die Variante sieht so aus, dass am Anfang Blattcliquen, in deren Knoten es keine Veränderung gab, nur dummy-Nachrichten an ihre Mütter schicken. Haben sich die Knoten in einer Mutter nicht geändert und empfängt diese nur dummy-Nachrichten von ihren Kindern, so sendet sie ebenfalls eine dummy-Nachricht weiter. Empfängt die Wurzel nur dummy-Nachrichten und gab es keine Änderungen in den Knoten der Wurzel, so gab es insgesamt keine Änderungen und der Algorithmus wird beendet. Beim Empfang einer dummy-Nachricht werden keine Berechnungen durchgeführt. Hat eine Clique eine dummy-Nachricht versendet, erhält jetzt allerdings eine π -Nachricht von ihrer Mutter, so muss sie ihre CPT anpassen; es ist dann $P(C_i) = \pi_j(S_{ji})\tilde{\psi}_i(C_i)$ mit der ursprünglichen $\tilde{\psi}_i$ direkt nach Einbeziehen der Evidenz.

Zum Anderen kann man auf Implementierungsebene optimieren, wie es [HD96] erwähnt. Die CPTs werden wie bereits beschrieben in einem Array gespeichert, und während der Inferenz sind die Multiplikation von CPTs wie auch das Aussummieren einzelner Knoten aus der CPT sehr häufige Operationen. Hier lässt sich viel Zeit sparen, indem man schon während der Bauphase einen Index für jede Clique und jeden Separator anlegt, in dem zu jedem Eintrag in der CPT der Clique die Position des korrespondierenden Eintrags in der Nachricht abgelegt wird, die über die zum Separator gehörende Verbindung verschickt werden wird. Weil die CPTs der Nachrichten stets für eine Teilmenge der beteiligten Cliques gelten, können auf diese Weise Summation, Multiplikation und Division in linearer Zeit (gemessen an der Anzahl der CPT-Einträge) vorgenommen werden.

Wie in Abschnitt 2.5 erwähnt wurde, ist das Inferenzproblem in allgemeinen Bayesischen Netzwerken NP-vollständig; daher ist auch der Aufwand des Cliquesbaum-Algorithmus im schlechtesten Fall exponentiell. Dies liegt daran, dass in diesem Fall die CPTs einzelner Cliques enorm groß werden. Im Normalfall ist die Inferenz jedoch ausreichend schnell. Auf einem Rechner mit Intel Xeon Prozessor mit 3 GHz Taktfrequenz, Linux-Betriebssystem (Kernel 2.4.20) und gcc-Compiler (Version 2.95) benötigte der Algorithmus für ein Netz aus 3 Knoten und 10 Variablen durchschnittlich 10.1 Mikrosekunden; für ein größeres Netz aus 17 Knoten und 996 Variablen benötigte er durchschnittlich 1820 Mikrosekunden.

3.4 Lernen

3.4.1 Einleitung

Wie schon in Abschnitt 2.6 besprochen, ist für die Anwendung im RoboCup das Lernen mittels Gradientenabstieg besonders geeignet. Bei diesem Lernverfahren betrachtet man die vereinigte Wahrscheinlichkeitsfunktion des Bayesischen Netzes. Man erweitert diese Funktion derart, dass sie als Parameter alle Einträge in den CPTs der Knoten und die Evidenz aller für das Training vorhandener Beispiele hat. Dann besteht die Optimierungsaufgabe darin, die CPT-Einträge so anzupassen, dass die erweiterte Wahrscheinlichkeitsfunktion für die gegebene Evidenz maximal wird. Dies erreicht man nach dem Prinzip des Gradientenabstiegs dadurch, dass man die Ableitung der Funk-

tion nach jedem CPT-Eintrag berechnet und ihn dann so verändert, dass der Funktionswert in Richtung des Gradienten verschoben wird. Wie Binder et al. in [BKRRK97] gezeigt haben, lässt sich der Gradient über eine einfache Formel mit wenigen Informationen berechnen, die üblicherweise durch die Berechnungen eines Inferenzalgorithmus an einem Knoten lokal bereits zur Verfügung stehen.

Für die Realisierung des Verfahrens stellten sich insbesondere folgende Anforderungen, die in den nächsten Abschnitten genauer betrachtet werden:

- Erstellung eines komfortablen Frameworks, um die Lernalgorithmen möglichst einfach benutzen zu können
- Verknüpfung mit den Inferenzalgorithmen, um die benötigten Werte möglichst rasch zu berechnen
- Anpassung der theoretischen Berechnungen, um innerhalb der durch die Wahrscheinlichkeitseigenschaften vorgegebenen Grenzen für die Variablen zu bleiben (siehe Abschnitt 3.4.4).

3.4.2 Herleitung der Gradientenabstiegsformel

Zum besseren Verständnis der folgenden Abschnitte wird hier die Herleitung der Anpassungsformel für die CPT-Einträge beim Gradientenabstieg wiedergegeben, wie sie in [BKRRK97] beschrieben ist. Sei $w_{ijk} = P(X_i = x_{ij} | \mathbf{U}_i = \mathbf{u}_{ik})$ ein Eintrag in der CPT von X_i und $P_{\mathbf{w}}(\mathbf{D})$ die durch das Netzwerk repräsentierte Wahrscheinlichkeitsfunktion in Abhängigkeit von den CPT-Einträgen \mathbf{w} und der Evidenz \mathbf{D} . Zuerst wird gezeigt, dass man den Beitrag jedes Beispiels zur Veränderung der Werte einzeln berechnen und diese Beiträge am Ende summieren kann:

$$\begin{aligned}
 \frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial w_{ijk}} &= \frac{\partial \ln \prod_{l=1}^m P_{\mathbf{w}}(D_l)}{\partial w_{ijk}} && \text{(da die Beispiele voneinander unabhängig sind)} \\
 &= \sum_{l=1}^m \frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial w_{ijk}} \\
 &= \sum_{l=1}^m \frac{\partial P_{\mathbf{w}}(D_l) / \partial w_{ijk}}{P_{\mathbf{w}}(D_l)}
 \end{aligned} \tag{3.4}$$

Auf einfache Weise berechnet werden soll also jeder einzelne Summand in dieser Formel. Dazu wird zunächst über die Variablen X_i und \mathbf{U}_i gemittelt:

$$\begin{aligned} \frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)} &= \frac{\frac{\partial}{\partial w_{ijk}}(\sum_{j',k'} P_{\mathbf{w}}(D_l|x_{ij'}, \mathbf{u}_{ik'})P_{\mathbf{w}}(x_{ij'}, \mathbf{u}_{ik'}))}{P_{\mathbf{w}}(D_l)} \\ &= \frac{\frac{\partial}{\partial w_{ijk}}(\sum_{j',k'} P_{\mathbf{w}}(D_l|x_{ij'}, \mathbf{u}_{ik'})P_{\mathbf{w}}(x_{ij'}|\mathbf{u}_{ik'})P_{\mathbf{w}}(\mathbf{u}_{ik'}))}{P_{\mathbf{w}}(D_l)} \end{aligned}$$

In der gesamten im Zähler dieses Ausdrucks stehenden Summe erscheint w_{ijk} nur in einem einzigen Summanden, nämlich für $j' = j$ und $k' = k$. Die Ableitung aller anderen Summanden ist 0; in diesem Summanden aber ist $P_{\mathbf{w}}(x_{ij'}|\mathbf{u}_{ik'})$ gerade der CPT-Eintrag, also w_{ijk} . Es ist also

$$\begin{aligned} \frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)} &= \frac{P_{\mathbf{w}}(D_l|x_{ij}, \mathbf{u}_{ik})P_{\mathbf{w}}(\mathbf{u}_{ik})}{P_{\mathbf{w}}(D_l)} \\ &= \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}|D_l)P_{\mathbf{w}}(D_l)P_{\mathbf{w}}(\mathbf{u}_{ik})}{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik})P_{\mathbf{w}}(D_l)} \quad (\text{Bayessche Regel}) \\ &= \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}|D_l)}{P_{\mathbf{w}}(x_{ij}|\mathbf{u}_{ik})} \\ &= \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}|D_l)}{w_{ijk}} \end{aligned} \tag{3.5}$$

Dieser letzte Ausdruck muss also im Zuge der Inferenz berechnet werden, wie im folgenden Abschnitt erläutert wird.

3.4.3 Berechnung des Gradienten

Wie oben gezeigt, ist es für die Gradientenberechnung nur notwendig, die Wahrscheinlichkeiten $P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}|D_l)$ zu bestimmen. Dies geschieht je nach Inferenzalgorithmus auf verschiedene Weise; im Normalfall werden die Werte während eines Inferenzdurchgangs nebenher mitberechnet.

3.4.3.1 Berechnung mit smile

Bei der Benutzung der smile-Bibliothek ist es nicht möglich, den Inferenzalgorithmus direkt zu benutzen, da der Sourcecode nicht verfügbar ist. Daher muss ein umständlicher Weg gegangen werden, der viel Berechnungszeit kostet. Dazu berechnet man

$$\begin{aligned} P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}|D_l) &= P_{\mathbf{w}}(x_{ij}|\mathbf{u}_{ik}, D_l)P_{\mathbf{w}}(\mathbf{u}_{ik}|D_l) \\ &= P_{\mathbf{w}}(x_{ij}|\mathbf{u}_{ik}, D_l)P_{\mathbf{w}}(u_{ik_1}|u_{ik_2}, \dots, u_{ik_m}, D_l)P_{\mathbf{w}}(u_{ik_2}, \dots, u_{ik_m}|D_l) \\ &= \dots \\ &= P_{\mathbf{w}}(x_{ij}|\mathbf{u}_{ik}, D_l) \prod_{l=1}^{m-1} P_{\mathbf{w}}(u_{ik_l}|u_{ik_{l+1}}, \dots, u_{ik_m}, D_l)P_{\mathbf{w}}(u_{ik_m}|D_l) \end{aligned} \tag{3.6}$$

Die gesuchten Werte lassen sich demnach durch folgenden Algorithmus ermitteln:

```
function parentProb(i, j, k: Integer) -- berechnet  $P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}|D_l)$ 
  probability: double = 1.0;
  U := parents( $X_i$ );
  for U :=  $U_m$  to  $U_1$  do
    inference();
    probability *=  $P(U = u_{kr})$ ;
    setEvidence(U,  $u_{kr}$ );
  od
  inference();
  probability *=  $P(X_i = j)$ ;
  Entferne gesetzte Evidenz der Eltern wieder
  return probability;
end function
```

Falls eine bei den Elternknoten zu setzende Evidenz im Konflikt mit D steht, so kann der Algorithmus natürlich mit dem Rückgabewert 0 abgebrochen werden, was durchaus eine deutliche Beschleunigung ergeben kann. Trotzdem ist der Algorithmus nur für Netze mit geringem Verzweigungsgrad ($|U|$ klein) brauchbar, da die vielen Inferenzdurchgänge sonst zu viel Zeit kosten.

3.4.3.2 Berechnung während Polybauminferenz

Zur Berechnung der Werte im Zuge des Polybauminferenzalgorithmus substituieren wir zunächst x für x_{ij} , \mathbf{u} für \mathbf{u}_{ik} , P für $P_{\mathbf{w}}$ und E für D_l und erhalten so die Notation von Abschnitt 3.3.1.1. Falls x keine Evidenz trägt und die Zustände der Knoten in U nicht im Konflikt mit E stehen, so gilt:

$$\begin{aligned} P(x, \mathbf{u}|E) &= P(\mathbf{u}|E)P(x|\mathbf{u}, E) = P(\mathbf{u}|E)P(x|\mathbf{u}, E_X^+, E_X^-) \\ &= P(\mathbf{u}|E)P(E_X^-|x, \mathbf{u}, E_X^+)P(x|\mathbf{u}, E_X^+) * 1/P(E_X^-|\mathbf{u}, E_X^+) \end{aligned} \quad (3.7)$$

Des weiteren ist

$$P(E_X^-|\mathbf{u}, E_X^+) = \frac{P(\mathbf{u}|E_X^-, E_X^+)P(E_X^-|E_X^+)}{P(\mathbf{u}|E_X^+)} = \alpha \frac{P(\mathbf{u}|E)}{P(\mathbf{u}|E_X^+)} \quad (3.8)$$

Setzt man dies in Gleichung 3.7 ein und beachtet außerdem, dass $P(E_X^-)$ bei gegebenem x unabhängig von \mathbf{u} ist und E_X^+ sowie $P(x)$ bei gegebenem \mathbf{u} unabhängig von $E_X^+ \setminus U$ sind, so erhält man

$$P(x, \mathbf{u}|E) = \beta P(E_X^-|x)P(x|\mathbf{u})P(\mathbf{u}|E_X^+) \quad (3.9)$$

Dabei ist $P(E_X^-|x)$ durch einen Aufruf von `evidenceExcept` berechenbar (falls nicht schon gespeichert), $P(x|\mathbf{u})$ ist ein Eintrag in der CPT von X , und $P(\mathbf{u}|E_X^+)$ lässt sich berechnen als

$$P(\mathbf{u}|E_X^+) = \prod_U P(u_l|E_X^+) = \prod_U P(u_l|E_{U_l \setminus X})$$

da im Polybaum die Eltern unabhängig voneinander sind, wenn die Evidenz von X und dessen Nachkommen nicht betrachtet wird. $P(u_l|E_{U_l \setminus X})$ lässt sich mit einem Aufruf von `supportExcept` berechnen, falls es noch nicht gespeichert ist.

Ist hingegen Evidenz beim Knoten X (und x ist der Evidenzzustand), so rechnet man:

$$\begin{aligned} P(x, \mathbf{u}|E) &= P(x|E)P(\mathbf{u}|x, E) = P(\mathbf{u}|E) \\ &= P(\mathbf{u}|x, E_x^+) = \frac{P(x|\mathbf{u}, E_x^+)P(\mathbf{u}|E_X^+)}{P(x|E_X^+)} \\ &= P(x|\mathbf{u})P(\mathbf{u}|E_x^+) * 1/P(x|E_X^+) \end{aligned} \quad (3.10)$$

Diese Werte kann man analog zu denen im ersten Fall berechnen.

3.4.3.3 Berechnung während Cliquesbauminferenz

Falls die Inferenz mit Hilfe des in Abschnitt 3.3.2 beschriebenen Algorithmus in einem Cliquesbaum durchgeführt wird, so lassen sich die benötigten Werte auf einfache Weise aus den CPTs der Cliques berechnen. Falls ein Knoten X einer bestimmten Clique C zugeordnet wurde, so sind auch seine Elternknoten in dieser Clique enthalten, und es gilt:

$$P_{\mathbf{w}}(X, \mathbf{U}_i|D_l) = \alpha \sum_{C \setminus (\mathbf{U}_i \cup \{X\})} P_{\mathbf{w}}(C|D_l) \quad (3.11)$$

mit einem Normalisierungsfaktor α .

3.4.4 Adaption des Gradienten

Nachdem der Gradient berechnet wurde, muss er noch angepasst werden, damit die im Bayesschen Netzwerk inhärenten Einschränkungen der CPT-Werte eingehalten werden. Diese Einschränkungen sind primär:

1. Alle Werte müssen im Intervall $[0.0, 1.0]$ liegen, da sie Wahrscheinlichkeiten repräsentieren
2. Für jede Kombination der Elternzustände eines Knotens muss die Summe der Werte über seine Zustände 1 ergeben, also $\sum_j w_{ijk} = 1.0 \quad \forall i, k$

Außerdem sollte für den Benutzer des Algorithmus die Möglichkeit bestehen, einzelne Werte auf bestimmte Intervalle zu begrenzen oder einzelne 'Spalten' der CPT (also Kombinationen von Elternzuständen) als völlig unveränderlich festzulegen. Dies ist beispielsweise nützlich, wenn manche Evidenzen sehr selten auftreten, da der Lernvorgang sonst die entsprechenden Einträge der CPT zu rasch auf 0.0 reduzieren würde.

Die zweite Einschränkung ist relativ einfach zu erfüllen: Es genügt, den Gradienten auf die Ebene zu projizieren, die dadurch bestimmt ist, dass die Summe ihrer Koordinaten 1 ergibt. Normalenvektor dieser Ebene ist $(1, 1, \dots, 1)^T$ und die Projektion liefert das Ergebnis

$$\Delta \mathbf{w}'_{ik} = \Delta \mathbf{w}_{ik} - \frac{1}{J} \sum_j \Delta w_{ijk} (1, 1, \dots, 1)^T$$

Das bedeutet, dass von jeder Koordinate der Durchschnitt der Koordinaten des Gradienten abgezogen wird. Da dann $\sum_j \Delta w_{ijk} = 0$, bleibt man innerhalb der vorgegebenen Grenze.

Die Beachtung der Grenzen für einzelne Koordinaten erfordert etwas mehr Überlegung. Man könnte beim Überschreiten einer dieser Grenzen einfach den Gradientenvektor soweit verkürzen, dass man innerhalb des vorgeschriebenen Gebietes bleibt. Dies würde jedoch bedeuten, dass der Algorithmus häufig nur äußerst kleine Schritte macht, falls er sich in auch nur einer Koordinate nahe an der Grenze befindet, was eine langsame Konvergenz zur Folge hätte. Deshalb wurde hier diese Methode nur angewandt, wenn der Algorithmus in fast jeder Richtung auf eine Grenze stößt. Ansonsten wurden die betroffenen Koordinaten im Gradientenvektor auf 0 gesetzt und die Projektion auf die Ebene neu berechnet. Dies entspricht einem „Entlanglaufen“ an der Grenze in Richtung der verbleibenden Werte des Gradienten.

Alternativ zum „naiven“ Hillclimbing wurde auch ein Gradientenabstieg nach der Methode von Polak-Ribière implementiert, der eine schnellere Konvergenz bei schmalen Tälern in der Oberfläche herbeiführt (siehe [Pri92]). Es konnte aber bei den hier verwendeten Netzwerken kein signifikanter Unterschied festgestellt werden.

Für die in Kapitel 4 besprochenen Lernvorgänge wurde eine Schrittweite zwischen 0.001 und 0.1 verwendet. Größere Schrittweiten führen zu einer rascheren Annäherung an das Optimum, kleinere Schrittweiten jedoch zu geringeren Abweichungen vom Ideal; man kann auch mit größeren Schrittweiten anfangen und diese dann nach und nach verkleinern, wie es beim „Simulated Annealing“ der Fall ist. Eine hier ebenfalls angewendete Methode ist, die Schrittweite bei korrekter Berechnung eines Beispiels deutlich kleiner zu setzen als bei fehlerhafter Berechnung. Die Idee hierbei ist, dass nahe am Optimum die meisten Trainingsbeispiele schon korrekt berechnet werden, sodass die durchschnittliche Schrittweite mit zunehmender Optimierung ebenfalls geringer wird.

3.4.5 Architektur des Frameworks

Das Framework basiert auf dem Zusammenspiel dreier Komponenten: dem Bayesschen Netz selbst, dem Lernalgorithmus und den Trainingsbeispielen. Die statische Struktur

zeigt Abbildung 3.6. Ein Trainingsbeispiel besteht einerseits aus vorgegebenen Evidenzen für die Knoten, an denen üblicherweise während der späteren Benutzung des Netzes Evidenz eingegeben wird („Quellevidenzen“); im RoboCup sind dies normalerweise die Sensorinformationen. Andererseits hält es außerdem vorgegebene Evidenzen für einen oder mehrere Knoten, an denen neu berechnete Wahrscheinlichkeiten aus dem Netz ausgelesen werden sollen („Zielevidenzen“). Trainingsbeispiele können auf einfache Weise in Klartextdateien geschrieben oder aus diesen wieder ausgelesen werden und haben Hilfsmethoden um beispielsweise herauszufinden, ob ein bestimmtes Netz das Beispiel korrekt berechnet – dies ist dann der Fall, wenn bei Festlegung der Quellevidenzen im Netz die Wahrscheinlichkeit der Zustände der Zielevidenzen gegenüber den anderen Zuständen ihrer Knoten am größten ist.

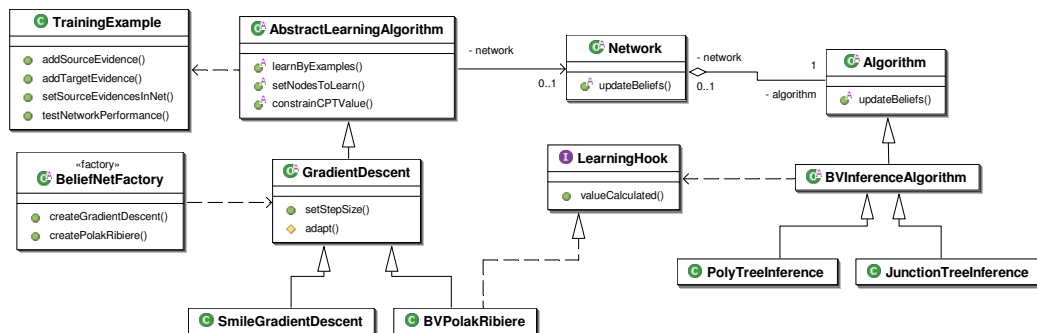


Abbildung 3.6: Framework zum Trainieren von Bayesschen Netzen

Dem Lernalgorithmus werden die Beispiele einzeln (v.a. beim Verstärkungslernen) oder besser als Liste übergeben, und er adaptiert das Netz entsprechend. Es wurden abstrakte Oberklassen (*AbstractLearningAlgorithm* und *GradientDescent*) eingefügt, um die Erweiterung durch verschiedene Algorithmen einfach zu machen. Für einzelne CPT-Einträge können wie im vorherigen Abschnitt angesprochen Grenzen festgelegt werden. Die selbst implementierten Inferenzalgorithmen (*BVInferenceAlgorithm* und dessen Unterklassen) bieten die Möglichkeit, die Gradientenabstiegsverfahren (welche die Schnittstelle *LearningHook* implementieren) innerhalb eines Inferenzdurchgangs von den für sie interessanten Werten $P(x_{ij}, u_{ik} | D_l)$ zu unterrichten, wie es in Abschnitt 3.4.3 erläutert ist.

Die Anpassung der CPT-Werte geschieht nach jedem einzelnen Beispiel statt gesammelt nach allen Beispielen. Das bedeutet, dass der jeweilige Gradientenvektor stets schon anhand der veränderten Werte berechnet wird und nicht wie in der theoretischen Herleitung anhand der ursprünglichen Werte. Es ist jedoch im Mittel davon auszugehen, dass das nach einem Beispiel adaptierte Netz besser ist als das ursprüngliche Netz, sodass diese Vorgehensweise eher noch zu schnellerer Konvergenz führt. Man sollte allerdings darauf achten, dass die Beispiele nicht allzu „geordnet“ vorliegen, da sonst der Lerneffekt der ersten Beispiele durch die Masse anders gearteter späterer Beispiele zunichte gemacht werden kann.

Detaillierte Hinweise zur Benutzung und Erweiterung des Frameworks sind im Anhang A ab Seite 73 zu finden.

Kapitel 4

Anwendung Bayesscher Netze im RoboCup

4.1 Besonderheiten des Szenarios

Bei der Verwendung Bayesscher Netze im RoboCup-Szenario gilt es einige Besonderheiten zu berücksichtigen, die durch die spezielle Anwendung bei autonomen mobilen Robotern hervortreten. Bayessche Netze werden in anderen Bereichen – beispielsweise bei der medizinischen Diagnose – häufig so eingesetzt, dass nach Festlegung der Evidenzen durch einen Experten einmalig Inferenz durchgeführt wird und der Experte die Ergebnisse ausliest. Im Unterschied dazu wird ein Netz hier sehr häufig (mit einer Frequenz von 5-20 Auswertungen pro Sekunde) neu ausgewertet. Hierbei muss stets automatisch die Evidenz aus den Sensordaten berechnet und ins Netz eingespeist werden, und dann muss Inferenz durchgeführt werden. Dieser Umstand schließt sowohl sehr große Netze als auch langsame Inferenzalgorithmen von Anfang an aus. Er stellt außerdem harte Anforderungen an die verwendete Software-Bibliothek: Bei sämtlichen mit der Inferenz befassten Teilen ist ein schonender Umgang mit den Systemressourcen absolut notwendig, und hohe Stabilität ist hier ebenso ein Muss wie ausreichende Geschwindigkeit.

Des Weiteren muss jedes Netz so entworfen werden, dass es völlig automatisch ausgewertet werden kann. Aus den Daten im Weltmodell des Roboters muss die Evidenz durch einen klaren (und einfachen) Algorithmus berechenbar sein; ebenso muss aus den im Netz berechneten Wahrscheinlichkeiten nutzbare Information gezogen werden können, ohne dass eine menschliche Interpretation notwendig wäre. Diese Anforderungen sind zwar normalerweise nicht allzu schwierig zu erfüllen. Dennoch müssen sie bei der Modellierung im Hinterkopf behalten werden.

Ein dritter Unterschied zu vielen typischen Verwendungen der Bayesschen Netze ist, dass relativ wenig Trainingsdaten vorliegen. Man hat hier keine Datenbank, in der hunderte oder tausende von Datensätzen für eine spezielle Anwendung gespeichert sind. Stattdessen müssen die Beispiele meist explizit generiert werden – aus diesem Grund bietet sich freilich auch Verstärkungslernen an (dazu mehr bei den einzelnen

Netzen). Außerdem sind die Netze üblicherweise nicht vollständig beobachtbar – die Gründe dafür sind im Abschnitt 4.2 beschrieben – wodurch viele Lernalgorithmen nicht verwendbar sind (siehe Abschnitt 2.6).

Schließlich sollte erwähnt werden, dass die Sensordaten im Weltmodell eines Roboters zum allergrößten Teil kontinuierliche Variablen sind, beispielsweise Geschwindigkeiten oder Abstände. Diese müssen vor der Verwendung diskretisiert werden.

4.2 Modellierungstechniken

4.2.1 Diskretisierung

Wie schon erwähnt, besteht ein erster wichtiger Schritt bei der Modellierung eines Bayesschen Netzes für den RoboCup darin, die kontinuierlichen Variablen im Weltmodell zu diskretisieren. Eine sehr typische Art der Diskretisierung ist beispielsweise die von Entfernungen: der genaue Entfernungswert wird umgesetzt in einen von vier oder fünf Zuständen, die dann “Extrem nah”, “Sehr nah”, “Nah”, “Mittelweit” und “Weit” genannt werden. Eine Schwierigkeit dabei liegt darin, die richtige Zustandsanzahl zu finden: Zu wenige Zustände machen das Netz inflexibel und ‘sprunghaft’, zu viele Zustände bedeuten viel mehr Aufwand beim Vorbelegen der CPTs und bei der Inferenzberechnung, und außerdem werden viel mehr Trainingsdaten benötigt. Im oben genannten Beispiel ist es erfahrungsgemäß so, dass bei kurzen Entfernungen eine feinere Diskretisierung notwendig ist als bei weiten Entfernungen (“zu weit” z.B. reicht als ein Zustand ab einer gewissen Entfernung völlig aus), was auch durch die Zustandsnamen schon angedeutet wird.

Eine zweite Schwierigkeit bei der Diskretisierung besteht darin, die Grenzen zwischen den einzelnen Zuständen richtig festzulegen. Hierbei helfen vor allem Erfahrung und Ausprobieren. Alternativ könnten die Grenzen auch durch ein Lernverfahren bestimmt werden. Darauf wurde hier allerdings verzichtet, denn das Modellierungsverfahren der “Sensorknoten” (Abschnitt 4.2.2) relativiert das Problem.

4.2.2 “Sensorknoten”

Eine äußerst wichtige und nützliche Technik ist die Modellierung von Sensordaten (nach der Diskretisierung) mit Hilfe zweier Knoten – sie findet sich in allen hier vorgestellten Netzen. Dabei erhält ein Knoten (der „eigentliche Sensorknoten“) die Evidenz aus den Weltmodelldaten, sie wird also normalerweise anhand der Berechnungen anderer Komponenten der Robotersoftware bestimmt (siehe auch Abschnitt 1.2, Seite 2). Der andere Knoten steht für den „realen“ Sachverhalt in der wirklichen Welt („Weltknoten“). Da eine Änderung in der realen Welt eine Änderung der Sensorinformationen verursacht und nicht umgekehrt, ist im Netz der Sensorknoten vom Weltknoten abhängig, denn die Modellierung Bayesscher Netze sollte so weit als möglich kausale Beziehungen übernehmen.

Die Art der Modellierung ist (u.a.) beschrieben in [RN95]; ein Beispiel (aus dem Netz, das in Abschnitt 4.3 beschrieben wird) zeigt Abbildung 4.1. Modelliert wird hier der Abstand des Roboters zum Ball, der von der Kamera gemessen wird.

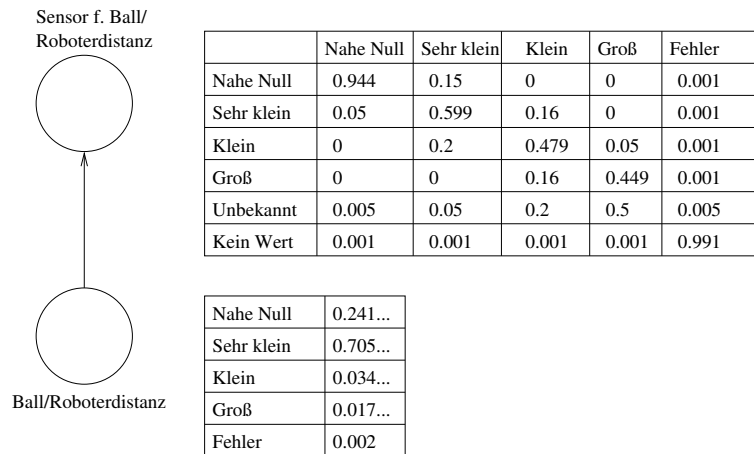


Abbildung 4.1: *Beispiel eines Sensorknotens*

Der erste Vorteil dieser Aufteilung in zwei Knoten ist offensichtlich: sie ermöglicht die Modellierung der Fähigkeiten des (physischen oder virtuellen) Sensors. So ist im Beispiel das “Vertrauen” in die Korrektheit der Sensordaten umso größer, je näher der Ball an der Kamera liegt (erkennbar an den sinkenden Werten in der Diagonale der CPT).

Besonders wichtig wird dieser Aspekt angesichts der Tatsache, dass der Ball gar nicht immer von der Kamera gesehen wird. Im Beispiel wurde die Wahrscheinlichkeit dafür, dass er außerhalb des Sichtfeldes liegt oder verdeckt ist, mit der tatsächlichen Entfernung des Balles deutlich erhöht. Ist der Ball “weit” entfernt, so ist die Wahrscheinlichkeit dafür, dass ihn die Kamera nicht sieht und der Sensor daher “unbekannt” liefert, ganze 50 Prozent. Umgekehrt (und für die Inferenz wichtiger) bedeutet das dann auch, dass die Wahrscheinlichkeit für einen “weit” entfernten Ball viel höher ist als für einen “sehr nahen” Ball, wenn der Sensor “unbekannt” meldet.

Von großem Vorteil ist weiterhin bei dieser Methode, dass auch die Möglichkeit eines komplett ausgefallenen Sensors direkt mitmodelliert werden kann. Eine gesonderte Behandlung ist nicht notwendig; im Beispiel wurde einfach ein Zustand “kein Wert” zum Sensorknoten und ein Zustand “Fehler” zum Weltknoten hinzugefügt. In diesem Fall wird primär davon ausgegangen, dass der Sensordefekt bemerkt wird. Ein gutes Beispiel dafür sind per Netzwerk übermittelte Daten von Teammitgliedern, also anderen Robotern. Falls der Mitspieler oder das Netzwerk ausfallen, so liefert der Sensor den Zustand “kein Wert”. Dies ist am wahrscheinlichsten, wenn der Weltknoten den Zustand “Fehler” hat; daher wird (trotz der geringen a-priori-Wahrscheinlichkeit dieses Zustands) nach der Inferenz der Zustand “Fehler” die höchste Wahrscheinlichkeit zugeordnet bekommen. Falls der Sensor ausfällt, ohne dass dies bemerkt wird, ist im Beispiel wahrscheinlich die Kamera defekt und liefert stets “Unbekannt” als Wert.

Falls das Netz mit entsprechenden Daten trainiert wird, so kann es sich auch daran anpassen, indem beispielsweise der CPT-Eintrag für (Fehler/Unbekannt) erhöht wird.

4.2.3 Innere Knoten

Wie in Abschnitt 2.7 dargelegt, ist es einer der großen Vorteile Bayesscher Netze, dass sie eine Wahrscheinlichkeitsfunktion sehr kompakt repräsentieren können. Dies gilt allerdings nur, wenn bei der Modellierung das Netz “sinnvoll” strukturiert wurde. Falls alle Sensorknoten direkt einen Zielknoten beeinflussen, so wird dessen CPT enorm groß – mit den schon an anderer Stelle genannten Nachteilen. Daher sollte man bei der Modellierung sog. “Innere Knoten” einfügen, die aus manchen Sensoren schon Teilwissen extrahieren. Im Idealfall erleichtern sie außerdem noch dem Menschen das Verständnis des Netzwerks, weil sie für Sachverhalte in der realen Welt stehen. In [RBKK95] wurden die Vorteile innerer Knoten auch experimentell belegt.

Im RoboCup sind innere Knoten normalerweise nicht beobachtbar, da nur die in den Sensorknoten modellierte Information zugänglich ist (könnte man einen inneren Knoten direkt beobachten, so könnte man seine Vorgängerknoten streichen und ihn als Sensorknoten benutzen). Deshalb sind viele Standardlernverfahren nicht anwendbar (siehe Abschnitt 2.6).

4.2.4 Baumstrukturen

Wie in Abschnitt 2.5 beschrieben, sind die meisten Inferenz-Algorithmen deutlich schneller, wenn das Netzwerk eine Polybaumstruktur hat; manche Algorithmen funktionieren sogar nur in diesem Fall. Deshalb wurde stets versucht, den Netzwerken eine solche Struktur zu geben. Es kommt jedoch häufiger vor, dass eine bestimmte Sensorinformation für mehrere Berechnungen notwendig ist. Falls aber tatsächlich nur der Sensor mehrfach benutzt werden soll und kein daraus berechneter innerer Knoten, so läßt sich dieses Problem einfach lösen, wie Abbildung 4.2 zeigt.

Die Struktur ist aus einem Netz entnommen, das berechnen soll, ob ein Torschuss sinnvoll ist (mehr dazu in Abschnitt 4.4). Im linken Teil der Abbildung sieht man die ursprüngliche Situation: die Sensorinformation darüber, wie weit das nächste Hindernis in Y-Richtung entfernt ist, wird für zwei innere Knoten verwendet. Beide Knoten wiederum beeinflussen einen anderen Knoten, sodass ein ungerichteter Zyklus entsteht. Um diesen aufzubrechen, wurden die Sensorknoten einfach verdoppelt, wie der rechte Teil der Abbildung zeigt. Das vergrößert das Netz zwar etwas, aber dieser Nachteil wird durch den Erhalt der Baumstruktur mehr als wettgemacht. Für die Berechnung werden beide Sensorknoten stets mit der gleichen Evidenz belegt.

Dass die beiden Netze äquivalent sind, ist aus der Struktur sofort ersichtlich: haben die Sensorknoten in beiden Fällen die gleiche Evidenz, so bleiben auch die Wahrscheinlichkeiten der ihnen nachgeordneten Knoten gleich, und daher auch die Wahrscheinlichkeiten an dem Knoten, an dem sich die Pfade wieder vereinigen.

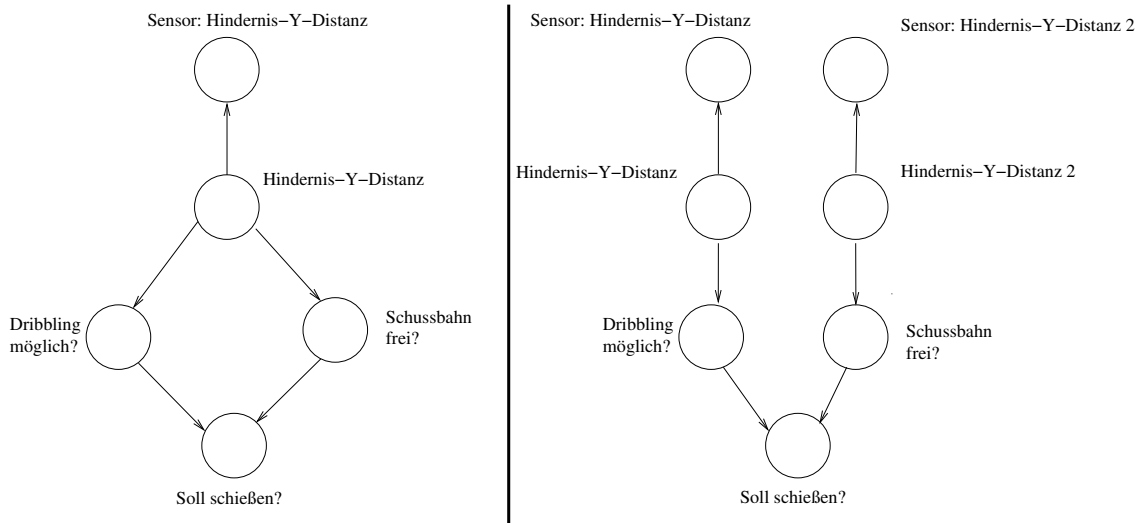


Abbildung 4.2: Erzeugung von Baumstrukturen

Das formale Argument ist folgendes: Im abgebildeten Teil des ursprünglichen Netzwerkes hat nur der Sensorknoten Evidenz. Unterhalb des Knotens, an dem die Pfade wieder zusammenkommen (N , hier: „soll schießen?“) ist keine Evidenz im Netzwerk. Das bedeutet, dass die Wahrscheinlichkeiten in den restlichen abgebildeten Knoten nur von der Evidenz beim Sensorknoten abhängen, denn von allen anderen Knoten mit Evidenz sind sie durch N d-separiert. Im veränderten Netzwerk sind die unteren der verdoppelten Knoten (also die Weltknoten) wiederum nur von den ihn zugeordneten Sensorknoten abhängig, weil N sie vom jeweils anderen Sensorknoten d-separiert. Legt man in den beiden neuen Sensorknoten also die gleiche Evidenz fest, so haben auch die beiden Weltknoten die gleichen Wahrscheinlichkeiten wie der Weltknoten im ursprünglichen Netzwerk, und daher sind auch die Wahrscheinlichkeiten in den anderen Knoten des Netzwerkes unverändert.

Die Äquivalenz der beiden Netze gilt freilich nur für die Inferenz während der Benutzung des Netzes. Während des Trainings eines Bayesschen Netzes mit der Gradientenabstiegsmethode (siehe Abschnitt 3.4) wird ja bei den Zielknoten (oder den Bewertungsknoten, siehe unten) Evidenz eingefügt, und diese Evidenz hebt die d-Separation der verdoppelten Knoten auf. Dieser Umstand ist aber nicht weiter von Belang: durch das Training wird schließlich das Netz so verändert, dass es während der üblichen Inferenz bessere Ergebnisse liefert. Während dieser üblichen Inferenz sind die beiden Varianten des Netzes jedoch wieder äquivalent.

4.2.5 Entscheidungen und Nützlichkeit

Wie in Abschnitt 2.2 erläutert, sind es die Entscheidungsnetzwerke, die besonders geeignet sind, um einen Roboter oder allgemein einen autonomen Agenten Entscheidungen treffen zu lassen. Die Entscheidung hängt dabei generell von zwei Komponenten ab: erstens der Wahrscheinlichkeit für das Eintreten einer bestimmten Situation, falls eine

bestimmte Entscheidung getroffen wird und zweitens der Nützlichkeit dieser Situation für den Agenten.

Soll der Roboter nun selbstständig lernen, welche Entscheidung in welcher Situation die beste ist, so stellt sich bei unveränderter Verwendung der Entscheidungsnetzwerke das Problem, dass sowohl die Bewertungsfunktion für Situationen als auch die Eintrittswahrscheinlichkeiten zunächst unbekannt sind. Dadurch erhöht sich die Zahl der freien Parameter und somit auch der Lernaufwand.

Man kann jedoch mit einer in [RN95] beschriebenen Methode Bewertungsfunktion und Eintrittswahrscheinlichkeiten in einem Knoten zusammenfassen und so das Lernen stark vereinfachen. Dies wird am besten an einem Beispiel klar. Nehmen wir an, der Roboter soll entscheiden, ob er versucht, zwischen einen angreifenden Gegner und das Tor zu fahren, um ihn am Angriff zu hindern. Im Idealfall gelingt dies, im schlechtesten Fall jedoch gelingt es nicht, und stattdessen verdeckt der Roboter seinem Torwart die Sicht. Den relevanten Teil eines (hypothetischen) Netzwerkes für diese Entscheidung zeigt Abbildung 4.3.

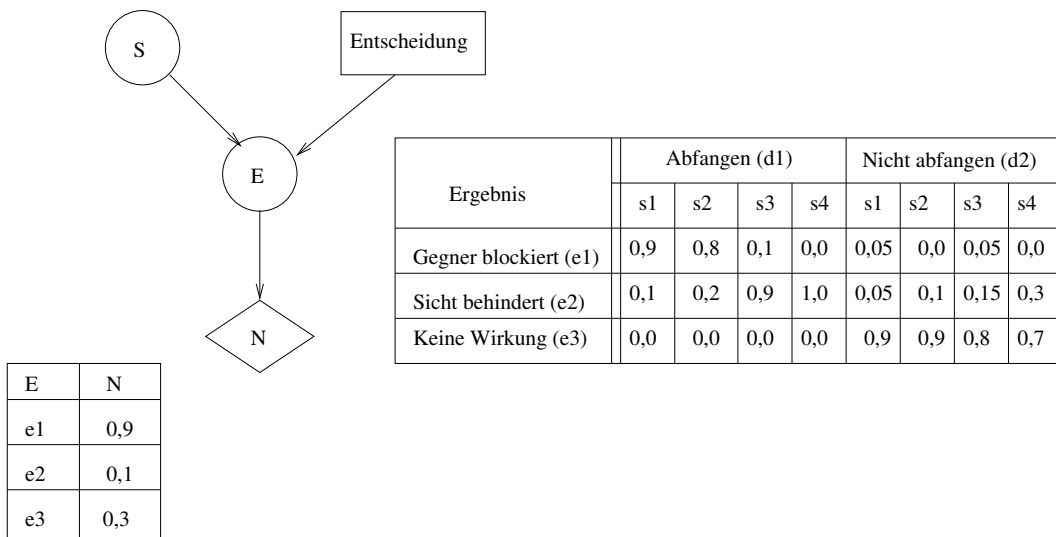


Abbildung 4.3: Beispiel zu Entscheidungen: Ursprüngliches Netz

Der Wahrscheinlichkeitsknoten S steht hierbei stellvertretend für die Situation, in der sich der Roboter befindet (sie könnte z.B. vom Abstand zum Gegner, vom Abstand zum Tor usw. abhängen) und hat beispielhaft vier Zustände. Der Knoten E modelliert die möglichen Folgesituationen (Ergebnisse). Die *erwartete Nützlichkeit* N einer Entscheidung d berechnet sich dann als $\sum_E N(e) \sum_S P(e|s, d) = \sum_S \sum_E P(e|s, d) * N(e)$, mit $N(e)$ als der Nützlichkeit einer Situation e . Dabei ist eine Nützlichkeit oberhalb von 0,5 als gut anzusehen, eine Nützlichkeit unterhalb von 0,5 als schlecht.

Nun kann man den Knoten E entfernen, und einen veränderten Knoten N' stattdessen einfügen, wie es Abbildung 4.4 zeigt.

Die CPT von N' wird so berechnet, dass der Eintrag für eine Situation s und eine Entscheidung d genau $\sum_E P(e|s, d) * N(e)$ ist. Der Wert für Situation $s3$ und Ent-

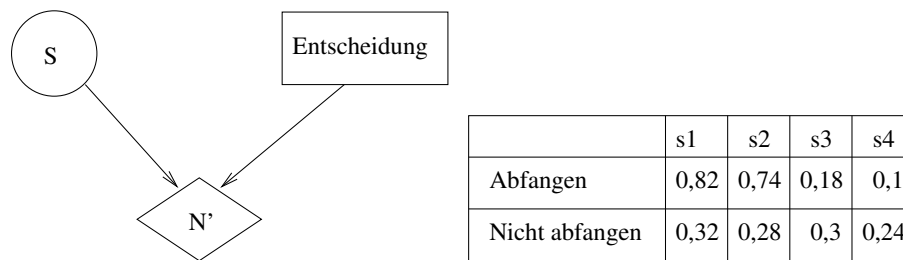


Abbildung 4.4: Beispiel zu Entscheidungen: Reduziertes Netz

scheidung „Nicht abfangen“ beispielsweise ist also $N'(s3, d2) = 0,05 * 0,9 + 0,15 * 0,1 + 0,8 * 0,3 = 0,3$. Die erwartete Nützlichkeit einer Entscheidung d ist dann wiederum $\sum_S N'(s, d) = \sum_S \sum_E P(e|s, d) * N(e)$, sie bleibt also gleich. Die Anzahl der freien Variablen hat sich in diesem Beispiel von 27 auf 8 verringert, bei mehr möglichen Folgesituationen ist die Einsparung noch größer.

Ein weiterer wichtiger Vorteil dieser Reduzierung liegt darin, dass man im neuen Netzwerk den Entscheidungsknoten verlustfrei entfernen und den Nützlichkeitsknoten (nach Normalisierung) als Wahrscheinlichkeitsknoten modellieren kann. So können die üblichen Inferenz- und Lernalgorithmen jeder Bibliothek für das Training herangezogen werden.

Nachteil der Reduzierung ist theoretisch der Informationsverlust – da Eintrittswahrscheinlichkeit und Nützlichkeit nicht mehr explizit getrennt modelliert werden, ist eine Änderung in einer dieser Funktionen für den Menschen schwerer in der Modellierung nachzuvollziehen. Da die kombinierte Funktion jedoch trainiert wird, ist dieser Nachteil nur noch sehr geringfügig und besteht hauptsächlich darin, dass im trainierten Netz die Einzelfunktionen nicht abgelesen werden können.

4.2.6 Bewertungsknoten

Insbesondere beim Verstärkungslernen ist es wichtig, dem Agenten (hier: dem Roboter) nach jeder Entscheidung ein Feedback geben zu können, das auch mehrere über „gut“ und „schlecht“ hinausgehende Stufen umfassen mag. Benutzt man zur Entscheidungsfindung ein Bayessches Netzwerk mit einer Struktur, wie sie im vorhergehenden Abschnitt besprochen wurde, so ist dies nicht direkt möglich. Der Grund dafür ist, dass bei den üblichen Lernalgorithmen (siehe Abschnitt 2.6) die Bewertung als Evidenz einfließen muss, man also die richtige Entscheidung vorgeben muss.

Um dieses Problem zu lösen, kann man den (reduzierten) Entscheidungsknoten in mehrere Knoten aufspalten, wobei für jede mögliche Entscheidung ein neuer Knoten entsteht. Die neuen Knoten haben stets zwei Zustände („Entscheidung getroffen“ und „Entscheidung nicht getroffen“), wobei die CPT-Werte für den ersten Zustand genau gleich den entsprechenden CPT-Werten im ursprünglichen Knoten sind. In jeder Situation gibt nun der Knoten die Entscheidung an, dessen Wahrscheinlichkeit für den ersten Zustand am höchsten ist. Es ist klar, dass das alte und das neue Netz daher

äquivalent sind. Im Laufe des Trainings verändern sich die Werte in den neuen Knoten zwar, eine Rückwärtstransformation ist jedoch durch Normalisierung stets problemlos möglich.

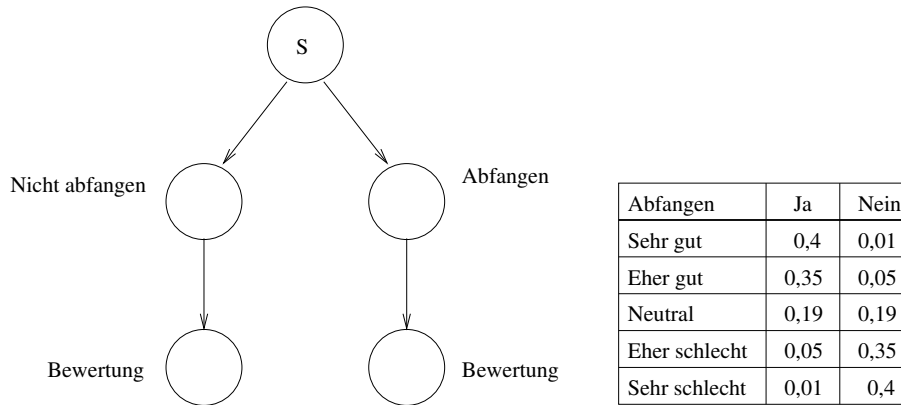


Abbildung 4.5: Beispiel für Bewertungsknoten

Damit nun Bewertungen für die Entscheidungen vorgenommen werden können, fügt man zu jedem neuen Entscheidungsknoten einen weiteren, von ihm abhängenden Knoten hinzu. Solange diese Knoten keine Evidenz tragen, ändert sich an den Wahrscheinlichkeiten der Entscheidungsknoten und damit auch an der getroffenen Entscheidung nichts. Die Knoten erhalten beliebig fein gestufte Bewertungszustände, wobei die Wahrscheinlichkeiten im Fall der positiven Entscheidung von gut nach schlecht abfallen und umgekehrt im Fall der negativen Entscheidung von gut nach schlecht ansteigen. Dies garantiert, dass eine im Lernschritt in den Bewertungsknoten eingefügte Evidenz den gewünschten Effekt hat: gute Bewertungen erhöhen die Wahrscheinlichkeit der Entscheidung in der jeweiligen Situation, schlechte Bewertungen erniedrigen sie. Die Bewertungsknoten selbst dürfen beim Training natürlich nicht verändert werden.

Abbildung 4.5 zeigt die entsprechende Veränderung des Beispielnetzwerkes aus dem vorhergehenden Abschnitt.

4.3 Ballbesitz

Die erste von einem Bayesschen Netz zu lösende Aufgabe war, herauszufinden, welche Mannschaft sich zu einem bestimmten Zeitpunkt im Ballbesitz befindet. Diese Aufgabe erschien aus mehreren Gründen besonders geeignet:

- Sie ist relativ komplex; d.h., eine Problemlösung auf andere Art und Weise – beispielsweise mittels if-then-Verzweigungen im Stile eines Entscheidungsbaums – ist nicht sofort ersichtlich. Insbesondere müssen auch Daten mehrerer Sensoren miteinander verknüpft werden, um das richtige Ergebnis zu erhalten.

- Die Aufgabe gehört in den Bereich der reinen Weltmodellierung. Es ist also noch kein Entscheidungsprozess an sie angeknüpft, was die Entwurfskomplexität verringert.
- Das entstehende Netz selbst (siehe Abbildung 4.6) ist einerseits groß genug, um die entwickelten Datenstrukturen und Algorithmen einem intensiven “realen” Test zu unterziehen; andererseits ist es durch seine einfache Struktur und die aussagekräftigen inneren Knoten gut durch den Menschen zu kontrollieren.

4.3.1 Aufbau des Netzes

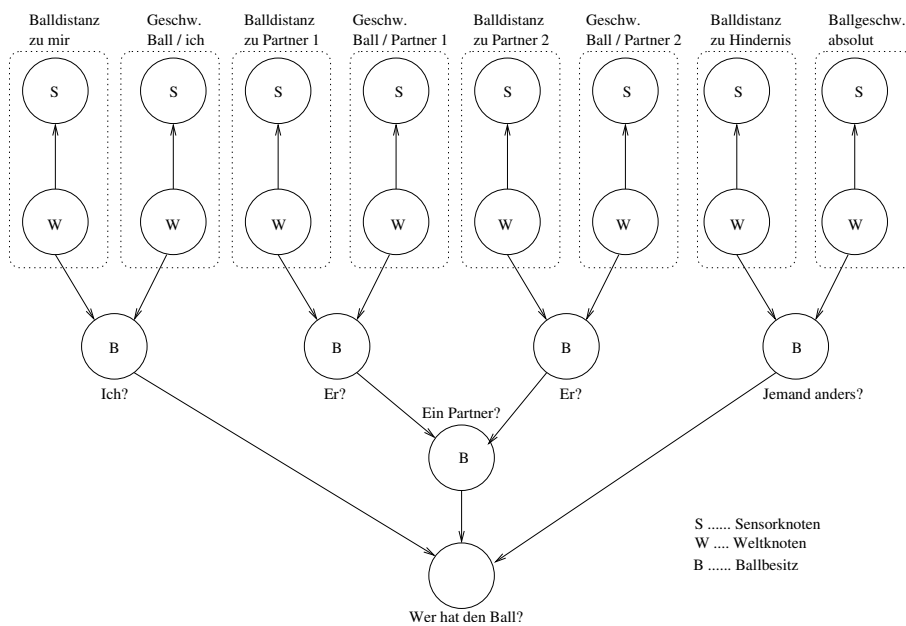


Abbildung 4.6: Netz zur Ermittlung des Ballbesitzes

Abbildung 4.6 zeigt die Struktur des Netzes, das entworfen und trainiert wurde. Es gliedert sich grob in drei Teile:

Der erste Teil (links oben) bestimmt, ob der berechnende Roboter selbst den Ball kontrolliert. Dies geht aus zwei Sensorknoten (siehe Abschnitt 4.2.2, Seite 44) hervor. Der eine liefert den relativen Abstand des Balles zum Roboter (genauer gesagt, zum Kickermechanismus), der andere die Relativgeschwindigkeit zwischen Ball und Roboter¹. Die beiden Werte werden jeweils diskretisiert in Zustände zwischen “nahezu Null” und “groß”. Zusätzlich haben beide Sensorknoten einen Zustand “unbekannt”, den sie annehmen, falls der Ball nicht im Blickfeld des Roboters liegt, und einen Fehlerzustand (wie in Abschnitt 4.2.2 beschrieben). Der Zustand “unbekannt” wird z.B. auch eingenommen, wenn der Ball direkt neben oder hinter dem Roboter liegt. Der jeweilige innere Knoten hat diesen Zustand nicht mehr (schließlich ist der reale Abstand zwischen Ball und Roboter immer in einer der Kategorien “nahezu Null” bis

¹Ein anderer Teil der Software berechnet diese mittels eines Kalman-Filters.

“groß”); die CPT des Sensorknotens ist jedoch so ausgelegt, dass der innere Knoten eine deutlich höhere Wahrscheinlichkeit für große Ballabstände annimmt, wenn der Sensor “unbekannt” liefert, was zu einer geringen Wahrscheinlichkeit für den eigenen Ballbesitz führt.

Geschw.	Abstand	Ja	Nein	Fehler
≈ Null	≈ Null	0.9998	0.001	0.001
	Sehr klein	0.3325	0.6675	0.0
	Klein	0.1693	0.8307	0.0
	Groß	0.0947	0.8606	0.0447
	Fehler	0.1	0.5	0.4
Klein	≈ Null	0.906	0.094	0.0
	Sehr klein	0.885	0.115	0.0
	Klein	0.5971	0.4029	0.0
	Groß	0.099	0.8518	0.491
	Fehler	0.1	0.5	0.4
Mittel	≈ Null	0.5084	0.4916	0.0
	Sehr klein	0.5115	0.4885	0.0
	Klein	0.3939	0.6061	0.0
	Groß	0.0485	0.853	0.0985
	Fehler	0.1	0.5	0.4
Groß	≈ Null	0.1294	0.8706	0.0
	Sehr klein	0.1625	0.8375	0.0
	Klein	0.0991	0.8518	0.0491
	Groß	0.0	0.9006	0.0994
	Fehler	0.5	0.5	0.0
Fehler	≈ Null	0.8	0.15	0.05
	Sehr klein	0.7	0.2	0.1
	Klein	0.6	0.2	0.2
	Groß	0.1	0.7	0.2
	Fehler	0.05	0.3	0.65

Tabelle 4.1: CPT des Knotens “Habe ich den Ball?”

Tabelle 4.1 zeigt die CPT des Knotens, der die Wahrscheinlichkeit für den eigenen Ballbesitz bestimmt (nach dem Training). Man kann u.a. erkennen, dass sehr wenige Beispiele einen Knoten im Fehlerzustand hatten (die Kamera fällt sehr selten aus), sodass sich an den entsprechenden Stellen die vorgegebenen Werte der CPT kaum geändert haben. Sie sind hier aber auch gerundet dargestellt.

Der zweite Teil des Netzwerks – in der Mitte von Abbildung 4.6 – bestimmt, ob einer der Mitspieler den Ball hat. Hierfür wird in genauer Entsprechung des ersten Teils für jeden Mitspieler bestimmt, ob er im Ballbesitz ist. Die dazu notwendigen Sensordaten übertragen die Mitspieler über das Kommunikationsnetzwerk. Einziger Unterschied ist, dass der Fehlerzustand hier häufiger eingenommen wird, da es schon einmal vorkommen kann, dass das Netzwerk defekt oder zu langsam ist oder auch dass ein Mitspieler

wegen Software- oder Hardwarefehlern ganz ausfällt. Die Wahrscheinlichkeiten für die Mitspieler werden dann in einem weiteren Zwischenknoten zusammengefasst, der modelliert, ob einer der beiden Mitspieler im Ballbesitz ist. Die CPT dieses Knotens war ursprünglich als reine Oder-Verknüpfung vorgegeben, hat sich durch das Training jedoch verändert – die Wahrscheinlichkeit für “Ja” dieses Knotens ist im trainierten Netz ca. 52 %, wenn für einen Mitspieler “Ja” gilt, und ca. 72 %, wenn dies für beide gilt. Diese Tatsache wird verständlich, wenn man sich vor Augen hält, dass zur Berechnung der Gesamtwahrscheinlichkeit im Endknoten (“Wer hat den Ball?”) eine Summation über alle Möglichkeiten durchgeführt wird.

Der dritte Teil schließlich bestimmt, ob *irgendein anderer* Spieler (also abgesehen vom berechnenden Roboter selbst) im Ballbesitz ist. Dies wird aus dem Abstand des Balles zum ihm nächsten Hindernis und der absoluten Ballgeschwindigkeit berechnet. So ist die Wahrscheinlichkeit für einen freien Ball größer, wenn er ruht oder sich – nach einem Schuss – sehr schnell bewegt, aber kleiner, wenn er sich mit mäßiger Geschwindigkeit voranbewegt; dann wird er möglicherweise von einem anderen Spieler geführt. Besser geeignet zur Bestimmung wäre auch hier die Relativgeschwindigkeit zwischen Hindernis und Ball. Im derzeitigen Weltmodell der Roboter wird aber für die Hindernisse keine Geschwindigkeitsberechnung durchgeführt.

Von allen drei Teilen hängt schließlich ein “Endknoten” ab, der die Zustände “Ich”, “Partner”, “Gegner”, “Niemand” und “Fehler” besitzt. Auch hier findet nicht nur eine triviale logische Verknüpfung statt, da die einzelnen Teile sich bestärkende oder widersprüchliche Information geben können. Eine hohe Wahrscheinlichkeit dafür, dass “irgendjemand” den Ball hat, verstärkt beispielsweise die Wahrscheinlichkeit, dass ein Partner im Ballbesitz ist, falls der zweite Teil eine solche Information liefert. Hat der Roboter selbst den Ball, so wird dies dagegen diese Wahrscheinlichkeit verringern (den eigenen Sensoren wird mehr Vertrauen entgegengebracht als den kommunizierten Daten der Mitspieler).

4.3.2 Training

Das Training des Netzes geschah ‘offline’. Da das Netzwerk keine Entscheidungen trifft, sondern der reinen Weltmodellierung dient, ist aktives Verstärkungslernen so wieso nicht anwendbar; zudem kann das korrekte Ergebnis nur durch den Menschen bestimmt werden (sonst hätte man das Netz nicht gebraucht). Es wurden also mit einem Hilfsprogramm, in dem ein menschlicher Benutzer Situationen vorgibt und dabei festlegt, welche Mannschaft sich im Ballbesitz befindet, Trainingsdatensätze generiert und das Netz mit ihnen trainiert.

Ein Vorteil der Bayesschen Netze ist es ja, dass jeder Knoten eine eigene Bedeutung hat und man seine Funktion an der CPT ablesen kann. Dies wurde beim Training ausgenutzt, indem das Netz in Abschnitten trainiert wurde. Dabei wurde zunächst der Teil trainiert, der bestimmt, ob der Roboter selbst den Ball hat (in Abbildung 4.6 ganz links oben). Die geänderten CPTs der Knoten können dann für die Mitspieler übernommen werden. Dann wurde der Teil trainiert, der bestimmt, ob ein Gegner den

Ball hat (ganz rechts oben in Abbildung 4.6). Im letzten Schritt schließlich wurden die zwei verbleibenden Knoten trainiert (in der Abbildung die beiden untersten), die dem Netz seine Gesamtfunktion geben.

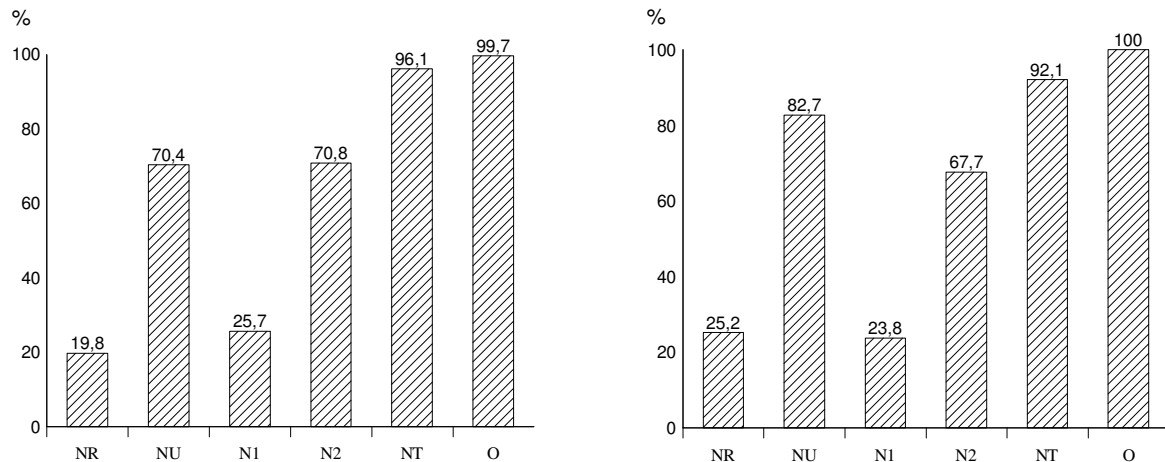


Abbildung 4.7: Anteil der von verschiedenen Netzen korrekt berechneten Beispiele. Links bezogen auf alle vorhandenen Beispiele, rechts bezogen auf die unterschiedlichen Situationen. NR = Netzwerk mit zufälligen CPT-Einträgen, NU = untrainiertes Netzwerk, N1 = Netzwerk nach dem ersten Trainingsschritt, N2 = Netzwerk nach dem zweiten Trainingsschritt, NT = fertig trainiertes Netzwerk, O = theoretisch erreichbares Optimum.

Der linke Teil der Abbildung 4.7 zeigt den Prozentsatz der vom Netz korrekt berechneten Trainingsbeispiele für verschiedene Stadien des Trainings (insgesamt 929 Beispiele). Der rechte Teil zeigt den Prozentsatz der korrekt berechneten *Situationen*, also unterschiedlichen Kombinationen der Quellevidenzen (202 verschiedene Situationen). Keiner der beiden Abbildungsteile ist für sich selbst genommen besonders aussagekräftig, da das Netz einerseits in jeder möglichen Situation korrekte Daten liefern sollte und andererseits aber häufig vorkommende Situationen wichtiger sind als seltene Situationen. Auch der rechte Teil der Abbildung zeigt nicht den Prozentsatz bezüglich *aller* Quellevidenzkombinationen, da einige extrem unwahrscheinlich und andere sogar unmöglich sind. Ein gewisser Anteil der Beispiele kann nicht korrekt berechnet werden, weil er widersprüchlich zu anderen Beispielen ist, daher ist auch der optimale Prozentsatz im linken Teil der Abbildung nicht 100 %. Dies liegt daran, dass auch der Mensch in manchen Situationen nicht immer gleich entscheidet: ist der Ball beispielsweise zwischen zwei Robotern eingeklemmt, so ist sowohl der eine als auch der andere Roboter im Ballbesitz. Der Abfall zwischen dem untrainierten Netz und dem Netz nach dem ersten Trainingsschritt erklärt sich dadurch, dass die veränderten Wahrscheinlichkeiten in den neu trainierten Knoten das Ergebnis im Endknoten negativ beeinflussen, bevor auch die anderen Trainingsschritte ausgeführt sind.

Abbildung 4.8 zeigt die Ergebnisse des Netzwerkes während des Trainings im letzten Schritt. Hierbei wurde das Netz mehrfach mit der 929 Beispiele großen Datei trainiert. Zum Vergleich wurde auch der Fortschritt eines Netzes angegeben, das nicht auf sinnvoll vorgegebenen Anfangswerten für die CPTs der zu trainierenden Knoten basiert.

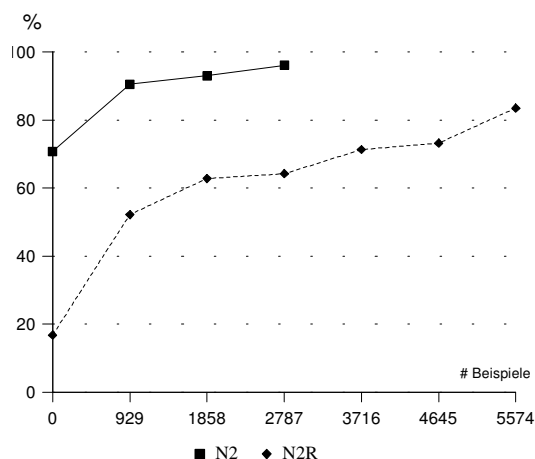


Abbildung 4.8: Ergebnisse während des letzten Trainingsschritts. N2 zeigt den Fortschritt beim Training des Netzes, das nach den ersten beiden Trainingsschritten entstand. N2R zeigt den Fortschritt eines Netzes, das im Unterschied zu N2 Zufallswerte in den zu trainierenden Knoten hat. Das Verfahren war Hillclimbing mit einer Schrittweite von 0,1.

In diesem Fall wird deutlich, dass die Bayesschen Netze in manchen Fällen stark von der Vorgabe guter Anfangswerte profitieren können.

Die Verbindungslinien im Diagramm dienen nur der besseren Orientierung des Lesers und sollten nicht als konstanter Fortschritt des Netzes interpretiert werden. Zwar passen sich die CPT-Werte durchaus graduell weiter an, sodass das Netz nach und nach dem Optimum näher kommt; jedoch geschieht die Auswertung hier ja bezüglich der korrekt berechneten Beispiele. Dadurch treten in Wahrheit an bestimmten Stellen Sprünge auf, immer dann, wenn eine oder mehrere Situationen zu den korrekt berechneten hinzukommen.

4.4 Schussentscheidung

4.4.1 Aufbau des Netzes

Bei der nächsten Problemstellung, die angegangen wurde, sollte durch das Bayessche Netz eine Entscheidung getroffen werden, die direkt das Verhalten des Roboters bestimmt. Dazu wurde folgendes Szenario betrachtet: der Roboter hat den Ball und befindet sich schon in der Nähe des gegnerischen Tores. Nur ein Torwart der gegnerischen Mannschaft steht einem Torerfolg möglicherweise noch im Wege. Jetzt stellt sich die Frage, zu welchem Zeitpunkt der Roboter schießen soll. Diese Entscheidung hängt von einer Reihe von Sensorinformationen ab, vor allem natürlich der Position des Roboters zum Tor und der Position des gegnerischen Torwarts.

Abbildung 4.9 auf der nächsten Seite zeigt das ursprüngliche Netzwerk zur Schussentscheidung vor Anpassungen, wie sie in den vorherigen Abschnitten beschrieben wurden. Rechts wird aus dem Abstand des Roboters zum Tor und der Schussrichtung

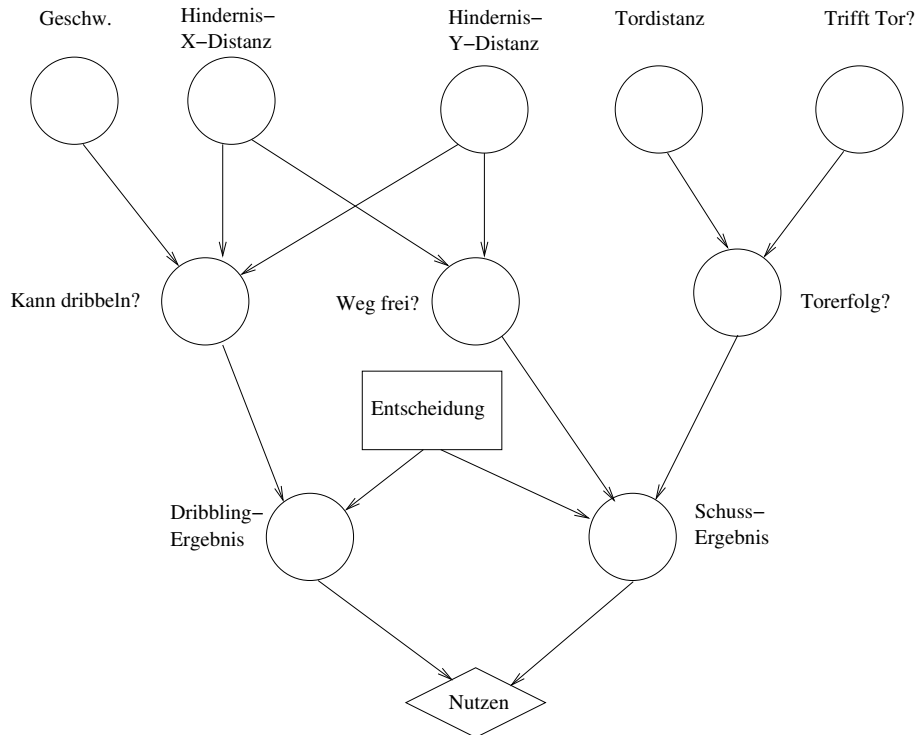


Abbildung 4.9: Ursprüngliches Netzwerk zur Schussentscheidung

bestimmt, mit welcher Wahrscheinlichkeit der Ball bei einem Schuss ohne Hindernisse ins Tor ginge. Zusätzliche Sicherheit vor unsinnigen Schussversuchen erhält man, indem man automatisch die Tordistanz auf „weit“ und „trifft Tor“ auf „nein“ setzt, falls das Tor nicht von der Kamera erkannt wird. Abgesehen davon, dass so Schüsse, die überhaupt nicht in Richtung des Tores gehen, verhindert werden können, modelliert dieser innere Knoten auch in gewisser Weise die Schussfähigkeiten des Roboters. In der Mitte oben sind zwei Knoten, deren Evidenz die Lage des Hindernisses angibt, das einen Torschuss am meisten stört (dies ist nicht einfach das Hindernis, das dem Roboter am nächsten ist; Hindernisse hinter ihm können z.B. ignoriert werden). Aus ihnen kann einerseits berechnet werden, mit welcher Wahrscheinlichkeit der Ball auf seinem Weg zum Tor von diesem Hindernis aufgehalten wird; andererseits auch, mit welcher Wahrscheinlichkeit der Roboter den Ball verliert, wenn er versucht, das Hindernis zu umspielen. Beim letztgenannten Wert spielen auch die Eigengeschwindigkeit des Roboters (bei hoher Geschwindigkeit sind Kurven schwieriger mit Ball zu fahren) und seine Dribblingfähigkeiten eine große Rolle. Zur feineren Abstufung hat der Knoten „Weg frei?“ nicht nur zwei Zustände „Ja“ und „Nein“, sondern zwei zusätzliche Zwischenzustände, die ungefähr „der Weg ist noch frei, könnte aber bald vom Gegner blockiert werden“ und „ein Schuss käme noch am Gegner vorbei, aber eine rasche Fahrt des Roboters selbst führt wohl zur Kollision“ bedeuten.

Aus den drei inneren Knoten können dann Wahrscheinlichkeiten für mögliche Resultate einer Entscheidung berechnet werden. Solche Resultate wären z.B. „Der Roboter hat ein Tor geschossen“, „Der Roboter hat den Ball verloren“ oder „Der Roboter hat den

Ball behalten". Die Trennung in Schuss- und Dribblingresultate sieht komplizierter aus, als sie im Endeffekt ist, da die Wahrscheinlichkeit für jedes Dribblingresultat bei Schussentscheidung 0 ist und umgekehrt. Jedem Resultat kann eine Nützlichkeit zugeordnet werden, und zusammen mit den errechneten Wahrscheinlichkeiten lässt sich ein durchschnittlich erwarteter Nutzen für jede Entscheidung bestimmen – die Entscheidung mit dem höchsten erwarteten Nutzen wird dann getroffen.

Dieses ursprüngliche Netzwerk wurde dann mit den in den vorherigen Abschnitten Methoden verändert, um es einfacher benutzbar zu machen. Das Ergebnis zeigt Abbildung 4.10. Hier wird schlussendlich in einem Knoten die Entscheidung über drei mögliche Aktionen getroffen: Dribbling, Schuss mit vorhergehender Beschleunigung des Roboters (um die Ballgeschwindigkeit zu erhöhen) oder Schuss ohne Beschleunigung (wegen Kollisionsvermeidung). Man sollte auch daran denken, dass diese Entscheidung mehrere Male pro Sekunde getroffen wird; eine Entscheidung für das Dribbling bedeutet also keineswegs automatisch, dass der Roboter zehn Sekunden lang oder zwei Meter weit dribbeln wird, sondern nur, dass er zumindest bis zum nächsten Entscheidungszyklus dribbeln wird.

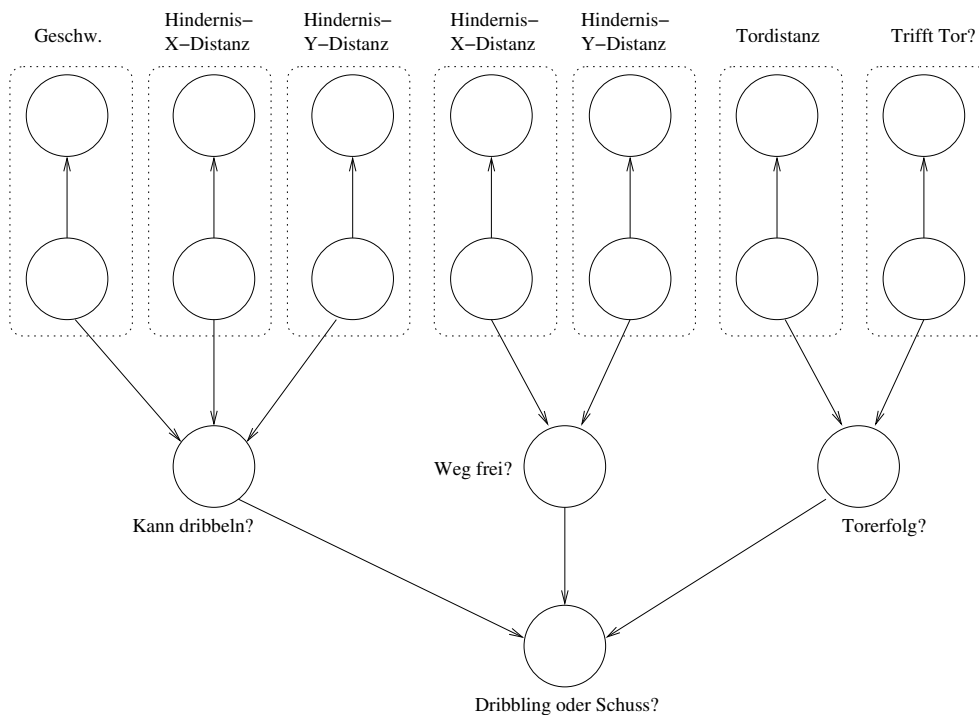


Abbildung 4.10: Netzwerk zur Schussentscheidung

4.4.2 Training

Um das Netzwerk zu optimieren, wurde Verstärkungslernen angewendet. Dazu wurde ein Programm geschrieben, das den Roboter mit dem Ball auf eine zufällige Position im Spielfeld setzt, wobei er genau in Richtung der gegnerischen Grundlinie blickt.

Mitspieler gibt es in dieser Situation keine, einziger Gegenspieler ist ein Torwart. Der Roboter wird also auf das Tor zudribbeln und irgendwann schießen. Sobald er die Entscheidung für den Schuss getroffen hat (nicht sobald er geschossen hat: wegen einer eventuellen Beschleunigung vor dem Schuss liegen diese beiden Zeitpunkte möglicherweise etwas auseinander) wartet das Programm noch einige Zeit auf ein Ereignis und generiert dann ein Trainingsbeispiel, mit dem das Netzwerk auch gleich trainiert wird. Dieses Trainingsbeispiel muss, damit der Gradientenabstiegsalgorithmus korrekt funktioniert, sowohl die Quellevidenzen (also die Evidenz bei allen Sensorknoten) als auch die Zielevidenz (hier die richtige Entscheidung) enthalten. Dann kann der Algorithmus die CPT-Werte so anpassen, dass bei Festlegung der Quellevidenzen die Wahrscheinlichkeit für die richtige Entscheidung erhöht wird. Dabei wird je nach Ergebnis des Schusses eine andere Entscheidung als richtig angesehen, wobei drei Folgesituationen unterschieden werden:

- Der Roboter kollidiert mit dem Torwart. Dann ist erst einmal davon auszugehen, dass der Schuss das Tor getroffen hätte (vielleicht hätte der Torwart ihn aber auch gehalten). Da durch die Kollision der Roboter jedoch ein Foul begangen hat, war die Aktion nicht erfolgreich. Es wird ein Schuss „aus dem Stand“ als korrekt angesehen. (Falls der Roboter in einer späteren Situation ohne Beschleunigung schießt und der Torwart hält, so wird sich die korrekte Entscheidung in der Situation automatisch zum Dribbling hin verschieben.)
- Der Roboter hat ein Tor erzielt. In dem Fall wird die Entscheidung des Roboters – egal ob Schuss mit oder ohne Beschleunigung – als korrekt angesehen.
- Keines der obigen Ereignisse tritt ein. Dann hat der Roboter das Tor verfehlt oder der Torwart hat den Schuss gehalten. In diesem Fall wird Dribbling als die richtige Alternative angesehen. Es mag sein, dass auch Dribbling in dieser Situation zu keinem Torerfolg führt (etwa, weil der Roboter direkt vor dem Torwart steht); in diesem Fall hätte der Roboter schon früher schießen müssen.

Dadurch, dass ein Beispiel erst nach erfolgter Schussentscheidung generiert wird, könnte es passieren, dass der Roboter immer später (und seltener) schießt, weil in einer Situation nie ein Schuss als richtig bewertet wird, falls der Roboter sich entscheidet, zu dribbeln. Daher wurde eine Explorationskomponente eingeführt, derart, dass der Roboter sich mit einer geringen Wahrscheinlichkeit auch dann für einen Schuss entscheidet, wenn das Netzwerk als Ergebnis eigentlich ein Dribbling empfehlen würde.

Bei den derzeitigen physikalischen und „motorischen“ Fähigkeiten des Roboters und der Navigatorkomponente kann vom Roboter nicht erwartet werden, dass er gegen einen gut agierenden Torwart (der sich beispielsweise stets zwischen Ball und Tor platziert) viele Tore schießt. Dafür ist der Schuss zu unpräzise und vor allem zu langsam. Deshalb wurden zwei andere Implementierungen eines Torwartverhaltens alternativ für das Training des Netzwerks benutzt. Im ersten Fall bewegt sich der Torwart überhaupt nicht; im zweiten Fall bewegt er sich regelmäßig und mit konstanter Geschwindigkeit vor dem Tor hin und her. Dennoch ist diese Situation keinesfalls als sehr unrealistisch

zu bewerten: erfahrungsgemäß haben viele gegnerische Teams in einem Turnier häufiger Probleme mit der Stabilität ihrer Hard- oder Software, und es ist gar nicht so selten, dass der Torwart der Gegner defekt ist oder den Ball nicht mehr erkennt.

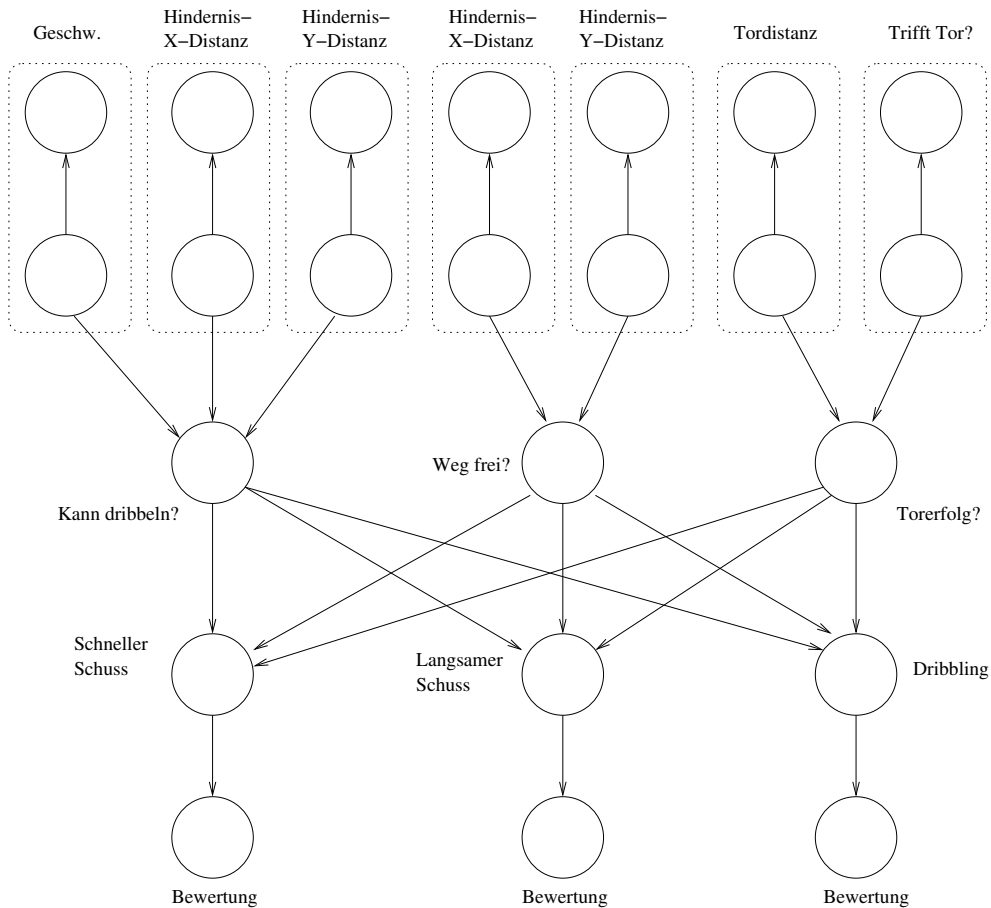


Abbildung 4.11: Variante des Netzwerks zur Schussentscheidung, bei der Bewertungsknoten für jede Entscheidung verwendet werden

Trotzdem stellte sich bei den ersten Versuchen rasch heraus, dass eine „reine“ Anwendung des Verstärkungslernens nicht zum Erfolg führt. Das Problem lag vor allem darin, dass der Roboter in vielen Situationen, in denen die Entscheidung für einen Schuss durchaus korrekt war, trotzdem kein Tor erzielte. Das lag z.B. daran, dass die Sensorinformationen leicht ungenau waren, daran, dass der Ball leicht vom Roboter wegrollte und deshalb nicht genug Impuls erhielt, oder daran, dass der Roboter während der Beschleunigungsphase die Kontrolle über den Ball verlor. Durch den hohen Nutzen, den der Roboter (bzw. seine Mannschaft) von einem Tor hat, ist es aber angebracht, einen Schuss auch dann zu riskieren, wenn er nicht mit absoluter Sicherheit zum Tor führt, sondern auch dann, wenn nur eine gewisse Wahrscheinlichkeit besteht, ein Tor zu erzielen. Der Misserfolg bei vielen Torschussversuchen führte jedoch dazu, dass sich der Roboter bald in fast jeder Situation gegen einen Torschuss entschied. Diesem Effekt wurde mit mehreren Maßnahmen begegnet:

- Die Explorationsquote wurde erhöht, sodass der Roboter öfter auch dann einen Schuss riskiert, wenn er eigentlich das Dribbling für erfolgreicher hält. Für das Training wurden jedoch nur diejenigen aus der Exploration erhaltenen Beispiele benutzt, die ein Tor zur Folge hatten.
- Für Beispiele mit Torerfolg wurde beim Training eine etwas höhere Schrittweite verwendet als für Beispiele ohne Torerfolg. Dies spiegelt in gewisser Hinsicht ebenfalls den hohen Nutzen eines Tores wieder.
- Die Trainingsdaten wurden derart bearbeitet, dass für Situationen, in denen die Misserfolgsquote zwischen 10 und 40 Prozent lag, Beispiele ohne Torerfolg entfernt wurden. So wurde für diese Situationen eine deutliche Präferenz für den Schuss (entweder mit oder ohne Beschleunigung) erzeugt. Das Netz wurde dann 'offline' mit den bearbeiteten Beispielen trainiert.

Die „Sensorknoten“ im Netzwerk wurden nicht trainiert, da nach einigen Tests klar war, dass die vorgegebenen CPT-Einträge ausreichend gut waren (bei der Modellierung konnte schon auf die Erfahrungen des Netzwerks zum Ballbesitz zurückgegriffen werden). Dadurch wird der Suchraum eingeschränkt und das Training beschleunigt.

Mit der Methode der „Bewertungsknoten“ (siehe Abschnitt 4.2.6) wurde noch eine Variante des Netzwerks erstellt, bei der für jede Entscheidung ein eigener Knoten existiert (siehe Abbildung 4.11 auf der vorherigen Seite). Die CPT der Bewertungsknoten ist gleich der im Beispiel in Abbildung 4.5. In einem Trainingsbeispiel ist dann also statt der korrekten Entscheidung eine Bewertung für jede der möglichen Entscheidungen enthalten. Die einzelnen Ergebnisse eines Schusses wurden dann folgendermaßen bewertet:

- Kollision oder Tor durch langsamen Schuss: „eher schlecht“ für schnellen Schuss, „sehr gut“ für langsamen Schuss und „neutral“ für Dribbling
- Tor durch schnellen Schuss: „sehr gut“ für schnellen Schuss, „neutral“ für langsamen Schuss und „eher schlecht“ für Dribbling
- Kein Tor und keine Kollision: „eher schlecht“ für schnellen und langsamen Schuss, „sehr gut“ für Dribbling

Wie erwartet waren keine signifikanten Leistungsunterschiede zwischen den beiden Varianten des Netzwerks festzustellen.

Statischer Torwart

Abbildung 4.12 auf der nächsten Seite zeigt die Ergebnisse des Trainings für einen statischen Torwart. Im linken Bild wird für einzelne Netzwerke ihre Leistung aufgeschlüsselt nach den Anteilen an Beispielen, die korrekt oder falsch berechnet werden und nach den Anteilen der Beispiele, die eine Schuss- bzw. Dribblingentscheidung als

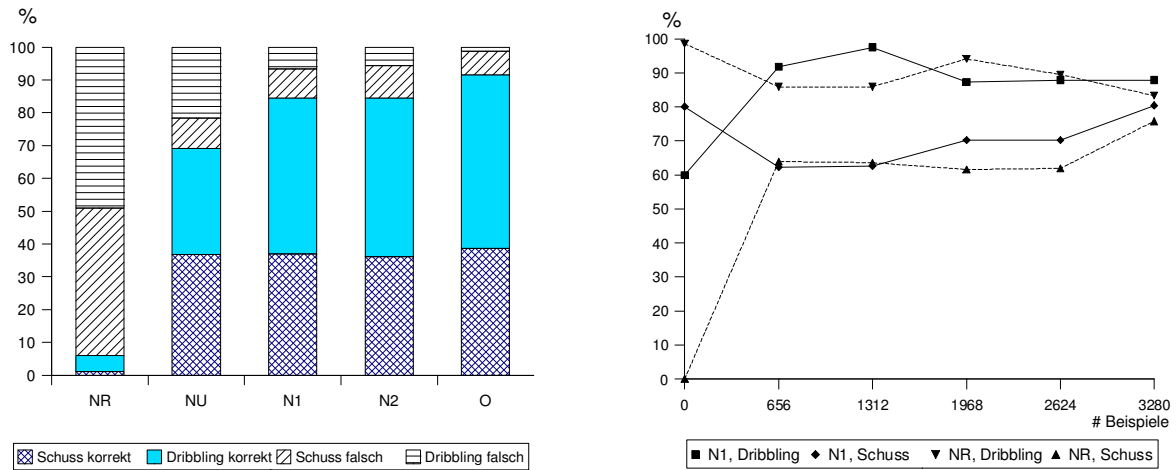


Abbildung 4.12: Ergebnisse und Trainingsfortschritt mit statischem Torwart. Links: Anteile der Entscheidungen, dabei ist NR = Netz mit zufälligen CPT-Werten in den zu trainierenden Knoten, NU = untrainiertes Netz, N1 = trainiertes Netz mit einem Zielknoten, N2 = trainiertes Netz mit Bewertungsknoten, O = theoretisch erreichbares Optimum. Rechts: Trainingsfortschritt

korrekt vorgeben. Insgesamt wurden 656 Beispiele verwendet, davon waren 302 mit Schussempfehlung. Wie auch schon beim Netzwerk zur Ermittlung des Ballbesitzes ist auch hier das theoretische Optimum kleiner als 100 %, weil für einige Situationen Beispiele mit unterschiedlichen Entscheidungsvorgaben vorhanden waren. Es ist klar zu sehen, dass das untrainierte Netzwerk mit den durch den Menschen vorgegebenen CPT-Einträgen schon recht gut ist, aber noch deutlich zu häufig eine Schussempfehlung abgibt.

Der rechte Teil der Abbildung zeigt den Trainingsfortschritt des Netzes mit einem Zielknoten (N1) und des gleichen Netzes, bei dem die CPTs der zu trainierenden Knoten mit Zufallswerten gefüllt waren (NR). Gezeigt wird jeweils der Anteil korrekt berechneter Beispiele für Schuss- und Dribblingsentscheidung in Prozent. Zu beachten ist, dass ein Netz erst dann gut ist, wenn es sowohl für die Fälle mit Schuss- als auch für diejenigen mit Dribblingsentscheidung gute Ergebnisse erzielt. Das zufällige Netz gibt z.B. fast immer eine Dribbling-Empfehlung ab, sodass es zwar praktisch alle Situationen korrekt berechnet, in denen gedribbelt werden soll, aber keine, in der geschossen werden soll. Das Training wurde mit Hillclimbing und einer Schrittweite von 0,04 durchgeführt. Mit dem trainierten Netz gelang es dem Roboter in ca. 64% seiner Versuche, ein Tor zu erzielen.

Tabelle 4.2 auf der nächsten Seite zeigt exemplarisch an der CPT des Knotens „Torserfolg“ (siehe Abbildung 4.10 auf Seite 57) die Veränderungen durch das Training in einem inneren Knoten. Es ist fast überall ein Anstieg der Wahrscheinlichkeiten für einen Torerfolg zu beobachten, was bedeutet, dass die zuerst festgelegten Werte zu pessimistisch waren. Wie oben erwähnt, hat das untrainierte Netzwerk jedoch zu häufig eine Empfehlung für einen Schuss abgegeben. Der scheinbare Widerspruch erklärt sich dadurch, dass im trainierten Netzwerk das relative Gewicht des Knotens „Torserfolg“ in Bezug auf die endgültige Entscheidung geringer geworden ist.

Tordistanz	Trifft Tor?	NU	N1
Sehr nah	Ja	0.98	0.512
	Nein	0.2	0.150
Nah	Ja	0.92	0.884
	Nein	0.1	0.215
Recht nah	Ja	0.6	0.616
	Nein	0.05	0.118
Mittel	Ja	0.25	0.260
	Nein	0.001	0.178
Weit	Ja	0.6	0.603
	Nein	0.1	0.304

Tabelle 4.2: CPT des Knotens „Torerfolg?“ vor und nach dem Training mit statischem Torwart. Dargestellt ist jeweils nur der Wert für den Zustand „Ja“; Wahrscheinlichkeiten für Fehlerzustände der Elternknoten wurden der Übersichtlichkeit halber weggelassen. NU = CPT im untrainierten Netzwerk, N1 = CPT im trainierten Netzwerk mit einem Zielknoten.

Torwart mit regelförmiger Bewegung

Für einen sich regelförmig bewegenden Torwart zeigt Abbildung 4.13 die Ergebnisse des Trainings (analog zu Abbildung 4.12). Der linke Teil zeigt genau die gleichen Netzwerke wie in Abbildung 4.12, nur dass das Training natürlich mit anderen Beispielen erfolgte. Im rechten Teil ist der Trainingsfortschritt für die beiden Varianten des Netzes gezeigt. Es wurde ebenfalls Hillclimbing mit einer Schrittweite von 0,04 benutzt; in jedem Trainingsdurchgang wurden 814 Beispiele verwendet, von denen 349 eine Entscheidung für einen Schuss beinhalteten. Hier hat das untrainierte Netzwerk deutlich zu oft das Dribbling als korrekte Aktion empfohlen.

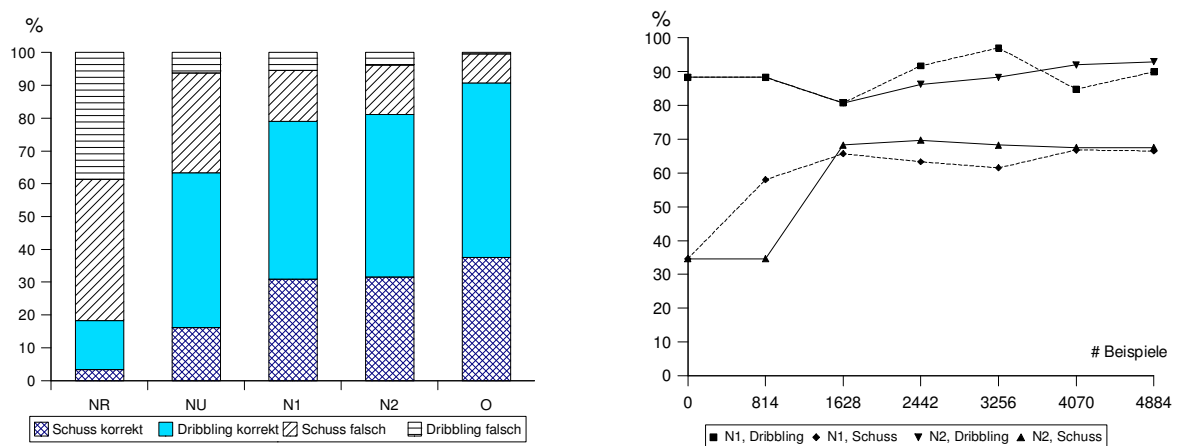


Abbildung 4.13: Ergebnisse und Trainingsfortschritt mit sich regelförmig bewegendem Torwart. Links: Anteile der Entscheidungen, dabei ist NR = Netz mit zufälligen CPT-Einträgen, NU = untrainiertes Netz, N1 = trainiertes Netz mit einem Zielknoten, N2 = trainiertes Netz mit Bewertungsknoten, O = theoretisch erreichbares Optimum. Rechts: Trainingsfortschritt, dabei ist N1 = Netz mit einem Zielknoten, N2 = Netz mit Bewertungsknoten

Tabelle 4.3 zeigt die Veränderungen in der CPT des Zielknotens beim Training mit sich regelförmig bewegendem Torwart. Hierbei sollte man beachten, dass die Elternknoten keine Evidenz tragen, sondern deren Zustände stets auch nur eine gewisse Wahrscheinlichkeit kleiner Eins besitzen. Auch hier geht die Bedeutung des Knotens „Torerfolg?“ während des Trainings zurück (es bleibt aber insgesamt dabei, dass nicht geschossen wird, falls das Tor nicht gesehen wird). Der Knoten „kann dribbeln?“ erhält vor allem Gewicht, wenn der Weg entweder völlig frei oder nur noch knapp frei ist: falls nicht gedribbelt werden kann, wird im ersten Fall der beschleunigte Schuss, im zweiten Fall der langsame Schuss bevorzugt.

Weg frei?	KD?	Torerfolg?	NU, BS	N1, BS	NU, LS	N1, LS
Ganz frei	Ja	Ja	0.7	0.324	0.05	0.050
		Nein	0.05	0.510	0.05	0.076
	Nein	Ja	0.9	0.814	0.05	0.021
		Nein	0.61	0.588	0.05	0.036
Noch frei	Ja	Ja	0.9	0.736	0.05	0.030
		Nein	0.05	0.347	0.05	0.045
	Nein	Ja	0.9	0.795	0.05	0.186
		Nein	0.6	0.604	0.2	0.050
Knapp frei	Ja	Ja	0.05	0.011	0.25	0.399
		Nein	0.05	0.134	0.05	0.147
	Nein	Ja	0.1	0.060	0.85	0.927
		Nein	0.05	0.041	0.6	0.832
Blockiert	Ja	Ja	0.5	0.259	0.2	0.059
		Nein	0.01	0.384	0.01	0.039
	Nein	Ja	0.05	0.374	0.65	0.537
		Nein	0.05	0.212	0.5	0.406

Tabelle 4.3: CPT des Knotens „Dribbling oder Schuss?“ vor und nach dem Training mit sich regelförmig bewegendem Torwart. KD = „Kann dribbeln?“, NU = untrainiertes Netzwerk, N1 = trainiertes Netzwerk mit einem Zielknoten, BS = Schuss mit vorhergehender Beschleunigung des Roboters, LS = Langsamer Schuss ohne Beschleunigung.

Beim Training für diese Situation mussten die Beispiele stärker angepasst werden (wie oben beschrieben), weil der sich bewegende Torwart die durch die oben erwähnten Probleme entstehenden zu langsamen Schüsse häufiger abfangen kann. Aber auch mit dieser Maßnahme verbleibt immer noch eine gewisse Anzahl Beispiele, in denen der Roboter zwar das Tor traf, deren Situation aber nach dem Training nicht als günstig bewertet wird, weil zu viele „Gegenbeispiele“ ohne Torerfolg vorhanden sind. Mit dem trainierten Netzwerk konnte der Roboter gegen den sich bewegenden Torwart in ca. 35% der Versuche ein Tor erzielen.

4.5 Passspiel

4.5.1 Aufbau des Netzes

Die Entscheidung, ob der Roboter einen Pass zu einem besser postierten Mitspieler versuchen soll, oder ob er selbst mit dem Ball Richtung Tor dribbeln soll, ist ebenso wichtig wie kompliziert. Wegen der eher geringen Fähigkeiten der Roboter, einen Schuss in eine präzise Richtung abzugeben oder einen sich bewegenden Ball abzufangen, und wegen der mechanischen Unfähigkeit der Roboter, die Stärke eines Schusses zu dosieren, ist ein Pass stets mit einem hohen Risiko verbunden, den Ball zu verlieren. Da die Roboter jedoch auch nur eine begrenzte Fähigkeit dazu haben, mit dem Ball Kurven zu fahren, und eine Unstetigkeit im Fahrverhalten auch häufig zu Ballverlust führt, ist in vielen Situationen auch ein Dribbling nicht von Erfolg gekrönt. Häufig kann man bei einem Misserfolg – unabhängig davon, welche Entscheidung getroffen wurde – daher nicht sagen, ob die Wahl von Dribbling oder Pass die richtige war.

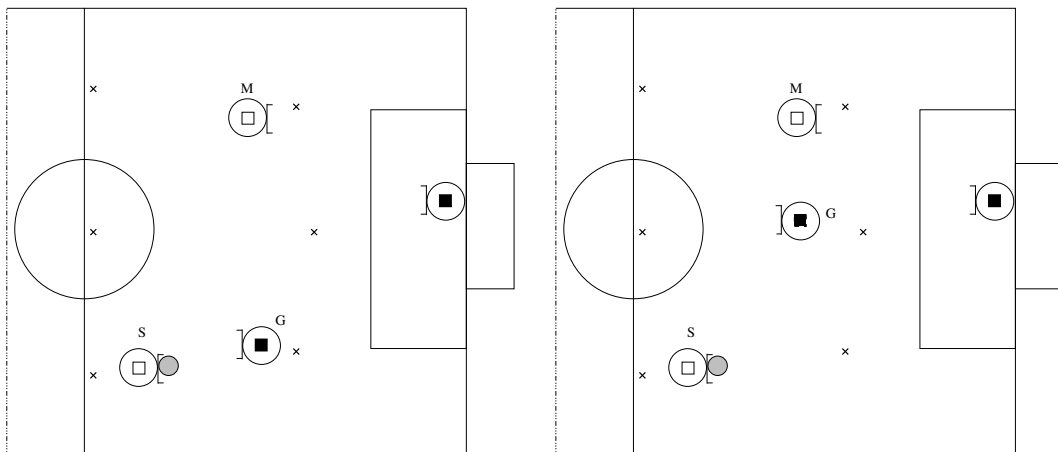


Abbildung 4.14: Beispielsituationen zur Entscheidung „Pass oder Dribbling?“. Links ist ein Pass die bessere Wahl, rechts das Dribbling.

Dennoch kann man zumindest in einigen Situationen eine klare Präferenz für die eine oder die andere Aktion festlegen; automatisiert natürlich insbesondere dann, wenn die Aktion erfolgreich war. Abbildung 4.14 zeigt beispielhaft zwei Situationen, in der sich der ballführende Roboter (S) entscheiden muss. Eine Aktion wird als erfolgreich angesehen, wenn durch sie der Ball am Gegner (G) vorbeigebracht wird. Steht der Gegner also bereit, den Roboter abzufangen wie in der linken Situation, so sollte dieser einen Pass zu seinem Mitspieler (M) versuchen. Steht der Gegner jedoch weiter weg, wie in der Situation rechts, so sollte auf jeden Fall gedribbelt werden.

Die entworfene Lösung der Aufgabe besteht in der Auswertung mehrerer Bayesscher Netze. Es wurden zunächst sechs Zielpositionen festgelegt, die für einen Pass in Frage kommen und ungefähr gleichmäßig über die gegnerische Hälfte verteilt sind (in Abbildung 4.14 durch die Kreuze markiert). Der Pass wird also *nicht* direkt zu einer Position gespielt, auf der gerade ein Mitspieler steht, sondern zu einer dieser vorher festgelegten

Positionen. Ein Mitspieler ohne Ball bewegt sich dabei zu einer dieser Positionen (genau die, die auch der Passspieler als am besten geeignet erachtet). Weil ein Pass relativ unpräzise ist und die sechs Positionen alle nützlichen Gebiete des Spielfelds abdecken (die hinteren für Pässe aus der eigenen Hälfte), entsteht durch diese Festlegung also kein Nachteil.

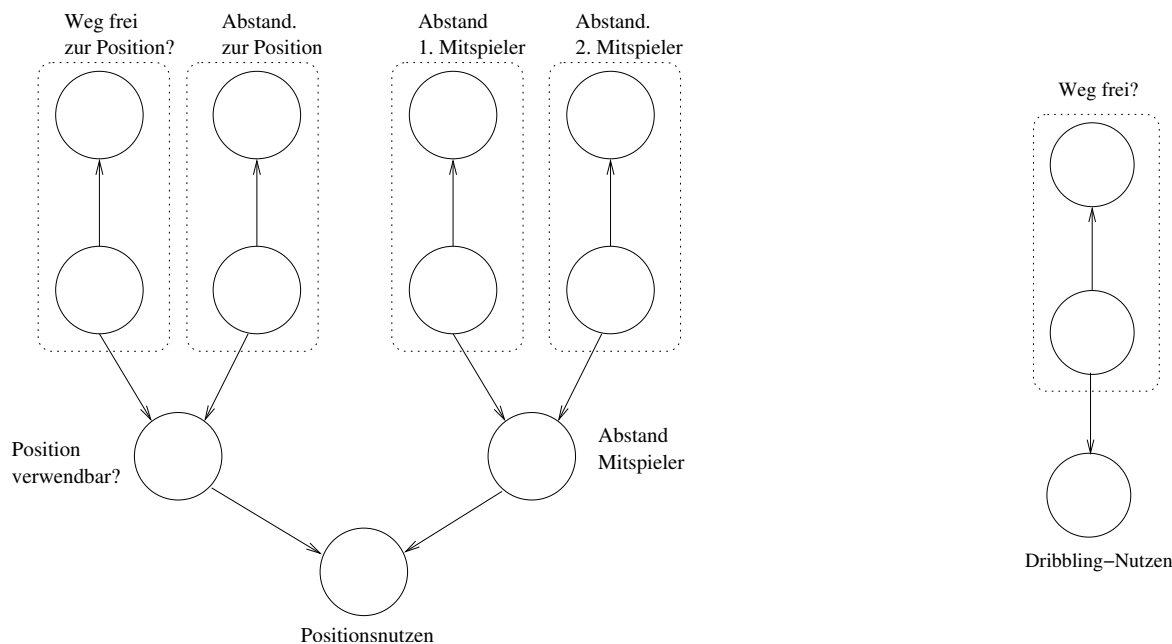


Abbildung 4.15: Netze zur Entscheidung über Pass oder Dribbling. Links das Netz zur Bestimmung des Nutzens eines Passes zu einer bestimmten Position, rechts das Netz zur Bestimmung des Nutzens eines Dribblings.

Für jede Position wird bewertet, ob der Schussweg zu ihr frei ist und wie weit die Mitspieler von ihr entfernt sind. Dies geschieht jeweils in einem Bayesschen Netz gleicher Art, wie es in Abbildung 4.15 abgebildet ist. Der rechte Teil bildet das Minimum aus den Abständen der (maximal) zwei Mitspieler. Im linken Teil wird die Information, ob der Schussweg zur Position frei ist, mit dem Abstand zu dieser Position verknüpft, um zu ermitteln, ob die Position überhaupt nutzbar ist. Sowohl zu weit entfernte (und somit nicht vernünftig erreichbare) als auch zu nahe Positionen bringen keinen echten Nutzen im Verhältnis zum Risiko eines Passes. Aus den beiden Teilen des Netzes wird schließlich eine Wahrscheinlichkeit dafür ermittelt, dass ein Pass zur Position nützlich wäre. Dieser Endknoten kann wiederum als eine Komprimierung eines Entscheidungsnetzwerks angesehen werden, wie es in Abschnitt 4.2.5 erläutert wird.

Ein weiteres kleines Netzwerk (rechts in Abbildung 4.15) bestimmt in ähnlicher Weise den wahrscheinlichen Nutzen des Dribblings. Eine Entscheidung wird dann getroffen, indem die wahrscheinlichen Nutzen aller möglichen Pässe und des Dribblings verglichen werden und die Aktion mit dem maximalen Nutzen gewählt wird. Außer Acht wird dabei gelassen, dass in manchen Situation vielleicht ein Rückpass hilfreich wäre, obwohl der Weg für das Dribbling frei ist (etwa falls der Roboter nahe einer Ecke steht und vom Tor weg schaut). Rückpässe sind jedoch prinzipiell besonders riskant, da sie dem

Gegner gewaltigen Vorteil bringen können, falls sie fehlschlagen; daher werden sie durch die gewählte Struktur so gut wie ausgeschlossen.

4.5.2 Evaluation

Der Test der Netzwerke wurde durchgeführt, indem in bestimmten Situationen der Mensch die richtige Entscheidung vorgab. Diese Entscheidung wurde dann nicht durch Ausspielen der Situation verifiziert. Ein solches Ausspielen, also der tatsächliche Versuch des Roboters, den Ball zu einem Mitspieler zu passen oder einen Gegenspieler zu umdribbeln, wäre wegen der geringen Ballkontrolle der Roboter sinnlos gewesen, denn wie schon erläutert wurde, führen in vielen Situationen weder Dribbling noch Pass zu einem Erfolg.

Schon die untrainierten Netzwerke (also mit vom Menschen vorgegebenen CPT-Einträgen) erreichten im Test eine Quote von 85 % richtiger Entscheidungen (theoretisches Optimum 97,9 %), wobei die Fehler hauptsächlich in der Entscheidung für das Dribbling statt einem Pass (8,2 %) und in der Wahl eines falschen Mitspielers (4,8 %) lagen. Der eigentlich kritischere Fehler, einen Pass zu spielen, wenn eigentlich noch ein Dribbling erfolgversprechender wäre, wurde nur in 2,0 % der Fälle gemacht. Dabei gehen diese Zahlen auch zu einem größeren Teil auf Grenzfälle zurück, als in Wirklichkeit zu erwarten ist: Beispielsweise ist im Spiel die Strategie normalerweise nicht so ausgerichtet, dass beide Mitspieler in unmittelbarer Nähe einer Passposition sind, weil normalerweise ein Spieler als Sicherung in der eigenen Hälfte bleibt.

Wie oben angesprochen, sind Pässe sowieso ein Risiko wegen der ungenügenden Fähigkeiten zu präzisen und dosierten Schüssen. Dieses Risiko überdeckt ein geringeres Risiko falscher Entscheidungen, insbesondere der Entscheidungen für Dribbling statt des Passes oder falsch gewählte Passpositionen. Eine extreme Präzision der Entscheidungsfindung ist hier also gar nicht erforderlich. Daher wurde darauf verzichtet, unter größerem Aufwand noch ein Training der Netze durchzuführen, das die Erfolgsquote vielleicht auf 91 % - 94 % angehoben hätte.

Kapitel 5

Zusammenfassung und Ausblick

5.1 Beitrag und Abgrenzung der Arbeit

In dieser Arbeit wird demonstriert, wie Bayessche Netze im RoboCup zur Entscheidungsfindung eingesetzt werden können. Es werden Techniken vorgestellt, die bei der Modellierung von in einem solchen Szenario gut verwendbaren Netzen helfen, und anhand einiger Beispielnetzwerke gezeigt, wie sie für verschiedene Aufgabenstellungen benutzt werden können. Die Arbeit demonstriert auch, wie mit Hilfe eines auf Gradientenabstieg basierenden Verfahrens die Netze mit Beispielen trainiert werden können, sodass ihre Erfolgsquote deutlich ansteigt.

Die Arbeit zeigt so, dass Bayessche Netze im RoboCup und ähnlichen Aufgabenstellungen ein vortreffliches Mittel für autonome Roboter sind, um Entscheidungen zu treffen, bei denen Unsicherheit über die aktuelle Situation oder die Folgen einer Entscheidung besteht. Durch das Training können sie eine Güte erhalten, die durch reine Modellierung oder logische Verknüpfungen nicht erreicht werden kann. Bei guter Vorbelegung durch den Benutzer und sorgfältiger Erstellung der Beispiele erreichen sie auch schon mit relativ wenigen Beispielen hohe Erfolgsquoten.

Durch die während der Arbeit entstandene Software – insbesondere die Bibliothek für die Verwendung und das Training Bayesscher Netze – wurde die Grundlage geschaffen, um auf einfache Weise weitere Netze einsetzen zu können. Bei ihrer Entwicklung wurde außerdem stets auf ihre Erweiterungsfähigkeit geachtet und sie ist ausführlich dokumentiert, sodass sie leicht an neue Aufgaben angepasst werden kann.

5.2 Ausblick

Die Verwendung Bayesscher Netze im RoboCup ist vielversprechend, steht aber noch weit am Anfang, weil sich die Forschungsaktivitäten bislang größtenteils auf Sensorik und Motorik richteten. Die Erstellung eines Netzes ist für nahezu jede Entscheidung, die eine Spielerkomponente treffen muss, denkbar. Sehr vielversprechend erscheint auch

die Möglichkeit, mittels speziell vorgeschalteter Knoten in ein Netz Wahrscheinlichkeiten einzubringen, die von anderen Softwareteilen (beispielsweise anderen Bayesschen Netzen, Markovketten o. ä.) berechnet wurden.

Für stetig wiederkehrende Entscheidungen, bei denen nicht nur die aktuelle Situation, sondern in gewissem Maße auch die Vergangenheit eine Rolle spielt, ist die Verwendung Dynamischer Bayesscher Netze geradezu prädestiniert. Hinsichtlich des Trainings der Netzwerke wäre ein Vergleich des Gradientenabstiegs mit einem anderen Verfahren wie etwa dem EM-Algorithmus genauso interessant wie auf der anderen Seite ein Vergleich mit alternativen Lerntechniken wie den Neuronalen Netzen.

Die entstandene Bibliothek kann auf einfache Weise erweitert werden, um andere Knotenarten, andere Inferenzalgorithmen oder andere Lernverfahren zu verwenden. Ebenso sind spezielle Erweiterungen für Dynamische Bayessche Netze denkbar.

Derzeit noch Zukunftsmusik, aber durchaus möglich, ist die Verwendung von Netzen mit kontinuierlichen Variablen und das selbstständige Erlernen geeigneter Netzwerkstrukturen durch den Roboter.

Anhang A

Benutzung der Software

A.1 Bibliothek für Bayessche Netze

A.1.1 Grundlegende Benutzung

Die Architektur der Bibliothek wurde bereits in Abschnitt 3.2 vorgestellt, und ihre wichtigsten Klassen sind in Abbildung 3.1 auf Seite 24 gezeigt. Damit die Bibliothek generell benutzt werden kann, muss im Hauptprogramm eine Fabrik instanziiert werden, die die konkreten Instanzen einer bestimmten Implementierung erzeugt. Dies geschieht mittels eines Aufrufs von

```
BeliefNetFactory::setFactory(new BVFactory());
```

(oder einer anderen Fabrik anstelle der BVFactory).

Dann kann man diese Fabrik ein Netzwerk erzeugen lassen:

```
Network* network =  
    BeliefNetFactory::getFactory()->createNetwork("MeinNetzwerk");
```

Erstellen eines Netzwerks

Hat man das Netzwerk schon auf andere Weise, etwa mit einem grafischen Editor wie Genie, erstellt, so kann man es danach einfach aus einer Datei laden:

```
network->loadFromFile("meinNetzwerk.dsl");
```

Ansonsten kann man das Netzwerk auch „von Hand“ durch Code erstellen, was hier ganz kurz anhand einiger Beispielanweisungen demonstriert werden soll:

```
int handle1 = network->addChanceNode("Knoten1");  
int handle2 = network->addChanceNode("Knoten2");  
network->addConnection(handle1, handle2);  
ChanceNode* node1 = network->getChanceNode(handle1);
```

```

vector<string> states;
states.push_back("Zustand1");
states.push_back("Zustand2");
node1->setStates(states);
vector<int> coords;
coords.push_back(0);
node1->setCPTValue(coords, 0.2);
coords[0] = 1;
node1->setCPTValue(coords, 0.8);

```

Es ist wichtig, dass ein Knoten stets zu einem bestimmten Netzwerk gehört und ohne das Netzwerk nicht existiert. Die auf dem Heap angelegten Knoten werden automatisch vom Destruktor des Netzwerks gelöscht und dürfen nicht von Hand dealloziert werden. Der Knoten wird normalerweise über ein Handle identifiziert, über das man ihn rasch vom Netzwerk bekommen kann, falls dies notwendig ist. Referenzen auf den Knoten selbst sollte man nicht länger als unbedingt nötig halten; in Datenstrukturen sollten also nicht die Knoten selbst, sondern deren Handles abgelegt werden. Das hat den zusätzlichen Bonus, dass diese Datenstrukturen kleiner und schneller werden.

Ein Wort zur Adressierung der Einträge in den CPTs: dies geschieht normalerweise über einen `vector` aus `ints`, wobei für jeden Elternknoten eine Koordinate vorhanden ist, die den Zustand des Elternknotens identifiziert. Die Reihenfolge der Eltern muss also bekannt sein (sie ändert sich auch beim Laden und Speichern nicht). Notfalls kann sie mittels der Methode `getParentNodes` des Netzwerks bestimmt werden, die einen `vector` mit den Handles der Eltern in der richtigen Reihenfolge zurückgibt. Die letzte Koordinate im Adressierungs-`vector` steht für den Zustand des Knotens selbst.

Durchführen von Inferenz

Um Inferenz durchzuführen, muss man zunächst einmal den Algorithmus festlegen. Die BV-Bibliothek kennt wie in Kapitel 3 besprochen Implementierungen des Polybaum- und des Cliquesbaum-Algorithmus.

```

BeliefNetFactory* factory = BeliefNetFactory::getInstance();
network->setInferenceAlgorithm(factory->
    createInferenceAlgorithm(LAURITZEN));

```

Default-Algorithmus der BV-Bibliothek ist der Cliquesbaum-Algorithmus. Algorithmen dürfen genauso wenig wie Knoten vom Benutzer selbst dealloziert werden.

Danach legt man einfach die Evidenzen fest, führt Inferenz durch und kann die Wahrscheinlichkeiten auslesen:

```

node1->setEvidence(0);
network->updateBeliefs();
double probability = node2->getProbability(0); // 0: erster Zustand

```

Thread-Sicherheit

Greifen mehrere Threads gleichzeitig auf ein Netzwerk zu, so sollte man es kapseln, um *race conditions* zu vermeiden:

```
SynchronizedNetwork* synchronizedNetwork =
    new SynchronizedNetwork(network);
```

Die Klasse `SynchronizedNetwork` bietet selbst schon die wichtigsten Methoden, um auf das gekapselte Netzwerk zuzugreifen, die jeweils die Sperren richtig setzen. Dies sind vor allem die Methoden zum Setzen von Evidenz, zur Durchführung von Inferenz und zum Auslesen der Wahrscheinlichkeiten. Man kann jedoch auch das Netzwerk direkt gesperrt erhalten, sollte die Sperre dann aber sobald wie möglich auch wieder freigeben:

```
Network* lockedNetwork = synchronizedNetwork->getWriteLockedNetwork();
...
lockedNetwork->unlock();
```

Hierbei ist auch auf korrekte Behandlung eventuell auftretender Ausnahmen zu achten.

Falls man bei der Übersetzung die Konstante `MARS_DEBUG` definiert, so achten die Klassen `Network` und `ChanceNode` auf korrekte Sperren. Das bedeutet, dass beispielsweise ein in einem `SynchronizedNetwork` gekapseltes `Network` beim Aufruf von `findChanceNode` prüft, ob der Aufrufer sich vorher um eine Lesesperre gekümmert hat, und im Fehlerfall (also wenn keine Sperre benutzt wurde) das Programm durch eine fehlgeschlagene Assertion beendet wird.

Die Header-Dateien, besonders die „öffentlichen“ im Verzeichnis `include`, enthalten ausführliche Kopfkomentare zu jeder Klasse und Methode, in denen u.a. Zweck der Methode, Parameter und Ausnahmen beschrieben werden.

A.1.2 Adapter zum Weltmodell

Die Adapterklassen zwischen Bayesschen Netzen und Weltmodell sind in Abbildung 3.3 auf Seite 26 gezeigt. Die Benutzung ist zunächst nicht völlig intuitiv, da Berechnungsaufwand vermieden werden soll, wird anhand eines Beispiels jedoch schnell klar. Soll etwa ein Sensorknoten seinen Zustand in Abhängigkeit der Entfernung des eigenen Roboters zum gegnerischen Tor gesetzt bekommen, so programmiert man:

```
GoalData* goalData = GoalData::getInstance(0); // 0: eigener Roboter
goalData->bindToNode(mySynchronizedNetwork,
                    myNodeHandle, 0); // 0: Aspekt Torentfernung
```

Nicht vergessen sollte man, die Knoten auch wieder abzumelden, bevor das Netz dealloziert wird (mittels `removeBinding`). Interessiert man sich für Daten von Teammitgliedern, so sollte man darauf achten, ob sich die Anzahl der Mitspieler ändert (es gibt ein Objekt im Weltmodell, das diese Anzahl speichert) und gegebenenfalls die Methode `tryRebind` eines Adapters aufrufen, sobald das entsprechende Objekt im Weltmodell existiert.

Schon vorhanden sind Adapter für folgende Daten:

- Abstand zum Tor und ob ein Schuss das Tor treffen würde (ohne Hindernisberücksichtigung): `GoalData`
- Abstand des Roboters zu bestimmten Positionen, die in der Konfigurationsdatei angegeben sind: `RobotPositionDistance`
- Abstand zwischen dem Ball und dem ihm nächsten Hindernis: `BallObstacleDistance`
- Abstand des Balls zu einem Roboter, wie ihn die Kamera misst: `BallRobotDistance`
- Relativgeschwindigkeit zwischen dem Ball und einem Roboter: `BallRobotVelocity`
- Absolute Geschwindigkeit des Balles: `BallVelocity`
- Abstand des 'störendsten' Hindernisses zum Roboter: `ObstaclePath`
- Ob der Schussweg zu bestimmten, in der Konfigurationsdatei angegebenen Positionen frei ist: `PositionPath`
- Absolute Geschwindigkeit eines Roboters: `RobotVelocity`

Andere Adapter können entweder (mit der Vorlage der bereits existierenden) rasch erstellt werden oder über eine Instanz von `GeneralSensorNode` realisiert werden, beispielsweise so:

```
GeneralSensorNode* myAdapter =
    new GeneralSensorNode(0, mySourceName, myFeatureName);
// Intervalle: ]-inf,0[, [0,1500[, [1500,3000[, [3000,inf[
int lowerBound = 0; int upperBound = 3000; int nrOfStates = 4;
myAdapter->setObjectIsInteger(lowerBound, upperBound, nrOfStates);
myAdapter->setErrorState(4);
```

Eine umgekehrte Transformation von Wahrscheinlichkeiten eines Knotens in ein Objekt im Weltmodell ist mittels der Klasse `WoMoObjectNode` gleichfalls einfach möglich.

Alle Adapter setzen die Benutzung eines `SynchronizedNetworks` voraus, da durch die Verwendung des Weltmodells die Benutzung durch verschiedene Threads sehr wahrscheinlich wird. Detaillierte Information zu den Klassen und Methoden ist wiederum in den Kopfkomentaren in den Header-Dateien zu finden.

A.1.3 Trainieren eines Bayesschen Netzes

Die Architektur des Frameworks zum Trainieren Bayesscher Netze wurde in Abschnitt 3.4.5 vorgestellt und die wichtigsten Klassen sind in Abbildung 3.6 auf Seite 42 gezeigt. Nachdem man sich überlegt hat, welche Knoten des Netzes trainiert werden sollen und welche die Eingabe- und die Ausgabeevidenzen tragen, ist der erste programmiertechnische Schritt normalerweise die Erzeugung eines oder mehrerer Trainingsbeispiele:

```
TrainingExample* example = new TrainingExample();
example->setSourceEvidence(node1, state1);
...
example->setTargetEvidence(node2, state2);
```

Die Beispiele kann man auch einfach in eine Datei schreiben und später wieder auslesen. Um mit den vorhandenen Werkzeugen kompatibel zu sein, bietet sich dafür an, zwei Beispiele stets durch ein Zeichen (z.B. ein Komma) zu trennen. Dann kann man sie auch mit wenigen Zeilen Code wieder in eine Liste bringen:

```
list<TrainingExample*> examples;
ifstream in("dateiname"); char ch;
while(in) {
    TrainingExample* example = new TrainingExample();
    in >> *example;
    examples.push_back(example);
    if (!in.eof()) in >> ch;
}
```

Die restlichen Schritte sind einfach und nahezu selbsterklärend. Sie bestehen hauptsächlich in der Parametrisierung des Lernalgorithmus:

```
GradientDescent* trainer = factory->createPolakRibiere(network);
list<int> nodesToLearn;
nodesToLearn.push_back(handle1);
...
trainer->setNodesToLearn(nodesToLearn);
trainer->setStepSize(0.1);
// evtl. Beschränkungen einzelner CPT-Werte
int returnCode = trainer->learnByExamples(examples);
network->writeToFile("neues_netzwerk.dsl");
```

Um Statistiken zu erhalten, ist es nützlich zu prüfen, ob ein Beispiel vom Netzwerk korrekt berechnet wurde. Dies bewerkstelligt man einfach mittels eines Aufrufs von `example->testNetworkPerformance(network)`.

Mehr Information, insbesondere zur Beschränkung von CPT-Werten oder zum Umgang mit dem Lernalgorithmus ist wiederum in den Kopfkomentaren der Headerdateien zu finden.

A.1.4 Dateiformat

Die BV-Bibliothek speichert und lädt die Netzwerke in Form von Klartextdateien des smile-Formats, wie sie vom Werkzeug „Genie“ ([smi]) erzeugt werden. Der Zeilenumbruch muss allerdings im UNIX-Format sein, sodass eventuell eine Konvertierung der Dateien mit den bekannten Werkzeugen (z.B. „dos2unix“) erforderlich ist. Der Inhalt der Datei ist im Großen und Ganzen selbsterklärend; die Syntax wird hier in EBNF-Notation angegeben:

```

Netzwerk := 'net ' NAME lf '{' lf NETZWERKDATEN lf KNOTEN* '};'
NAME := String
NETZWERKDATEN := HEADER ERSTELLUNG SAMPLES ANZEIGE USERPROPS

HEADER := 'HEADER = ' lf '{' lf ID NAME COMMENT '};' lf
ID := 'ID = "' String '";' lf
NAME := 'NAME = "' String '";' lf
COMMENT := 'COMMENT = "' String '";' lf

ERSTELLUNG := 'fEATION = ' lf '{' lf CREATOR CREATED MODIFIED '};' lf
CREATOR := 'CREATOR = "' String '";' lf
CREATED := 'CREATED = "' Date '";' lf
MODIFIED := 'MODIFIED = "' Date '";' lf

SAMPLES := 'NUMSAMPLES = ' Integer ';' lf
ANZEIGE := SCREEN WINDOW BKCOLOR lf
SCREEN := 'SCREEN = ' lf '{' lf
    SCREENPOS COLOR SELCOLOR FONT BORDER '};' lf
SCREENPOS := 'POSITION = ' lf POSITION

POSITION := '{' lf 'CENTER_X = ' Integer ';' lf
    'CENTER_Y = ' Integer ';' lf
    'WIDTH = ' Integer ';' lf
    'HEIGHT = ' Integer ';' lf '};' lf

COLOR := 'COLOR = ' Long_Integer ';' lf
SELCOLOR := 'SELCOLOR = ' Long_Integer ';' lf
FONT := 'FONT = ' Integer ';' lf 'FONTCOLOR = ' Integer ';' lf
BORDER := 'BORDERTHICKNESS = ' Integer ';' lf
    'BORDERCOLOR = ' Long_Integer ';' lf
WINDOW := 'WINDOWPOSITION = ' POSITION
BKCOLOR := 'BKCOLOR = ' Long_Integer ';' lf

USERPROPS := 'USER_PROPERTIES = ' lf '{' lf '};' lf

KNOTEN := 'node ' NAME lf '{' lf TYP HEADER

```

```

NODEINFO ELTERN DEFINITION '};' lf
TYP := 'TYPE = CPT;' lf
NODEINFO := SCREEN USERPROPS DOKUMENTATION
DOKUMENTATION := 'DOCUMENTATION = ' lf '{' lf '};' lf

ELTERN := 'PARENTS = ' LIST_OF_String ';' lf
DEFINITION := 'DEFINITION = ' lf '{' STATES CPT '};' lf
STATES := 'NAMESTATES = ' LIST_OF_String ';' lf
CPT := 'PROBABILITIES = ' LIST_OF_Double ';' lf

LIST_OF_X := '(' X? (',' X)* ')'
```

'lf' steht dabei für einen Zeilenumbruch. Zeitliche Daten werden in der Form 13:22:39 Monday, June 30, 2003 angegeben. Gegenüber zusätzlichen Leerzeilen und White-space am Anfang von Zeilen ist die Bibliothek normalerweise tolerant, eine Netzwerkdatei sollte aber sowieso nur in Ausnahmefällen von Hand bearbeitet werden. Kommentare in der Datei sind nicht vorgesehen.

Von diesen Daten benutzt die BV-Bibliothek Beschreibungen, Kommentare und alle zur Anzeige gehörenden Daten nicht.

Die BV-Bibliothek ist außerdem in der Lage, Dateien im Format von Hugin 6.x ([hug]) zu schreiben und Dateien dieses Formats einzulesen. In letzterem Fall muss die Datei wiederum im UNIX-Format vorliegen. Außerdem können Hugin-spezifische Informationen (wie z.B. automatische Inferenz, Art der Cliquenerstellung etc.) verloren gehen.

A.1.5 Hilfsprogramme

NetworkRandomizer

Mit Hilfe des Programms *NetworkRandomizer* kann man die CPTs eines Netzwerks mit Zufallszahlen belegen. Dabei wird nur darauf geachtet, dass die üblichen Bedingungen eingehalten werden (die Zahlen sind also zwischen 0 und 1 und die Summe der Wahrscheinlichkeiten für eine bestimmte Kombination von Elternzuständen ergibt immer 1). Man kann dem Programm auch die Handles von Knoten übergeben, deren CPTs nicht geändert werden sollen. Die Netzwerkdateien müssen im Smile-Format vorliegen. Die Aufrufsyntax des Programms ist:

```
NetworkRandomizer eingabedatei ausgabedatei [fester Knoten]*
```

NetworkEqualizer

Sehr ähnlich wie *NetworkRandomizer* funktioniert *NetworkEqualizer*. Einziger Unterschied ist, dass die CPTs nicht mit Zufallswerten gefüllt werden, sondern für jeden Zustand stets die gleiche Wahrscheinlichkeit gilt (bei vier Zuständen beispielsweise 25% für jeden Zustand). Die Aufrufsyntax lautet

```
NetworkEqualizer eingabedatei ausgabedatei [fester Knoten]*
```

ExampleAnalyzer

Das Programm *ExampleAnalyzer* kann verwendet werden, um eine Datei mit Trainingsbeispielen zu analysieren. Es liefert u.a. die Zahl der Beispiele, die Zahl der Beispiele pro Zielevidenzkombination, die Zahl der Beispiele pro Quellevidenzkombination und die Zahl der Beispiele pro Gesamtevidenzkombination. So kann man beispielsweise herausfinden, in welchem Anteil die möglichen Ergebnisse in der Datei vorhanden sind, wieviele falsche Beispiele vorhanden sind oder ob manche Quellevidenzkombinationen nutzlos sind, weil sie annähernd gleichverteilt zu mehreren Ergebnissen führen. Ebenfalls nützlich ist die Analyse, um zu überprüfen, wie gut die Abdeckung über die möglichen Quellevidenzkombinationen ist. Die Ausgabe kann auf der Shell natürlich auch durch weitere Programme wie **grep**, **sort** oder **awk** weiterverarbeitet werden. Der *ExampleAnalyzer* erwartet die Beispiele in der Datei als kommagetrennte Liste, der eventuell eine ebenfalls kommagetrennte und eingeklammerte Liste der zu trainierenden Knoten vorausgehen kann. In diesem Fall muss man dem Programm einen (beliebigen) zweiten Parameter mitgeben. Die Aufrufsyntax lautet also

```
ExampleAnalyzer beispieldatei [mit_header]
```

ExamplesMixer

Der *ExamplesMixer* dient dazu, Listen von Trainingsbeispielen zu vermischen, sodass nicht alle Beispiele für ein bestimmtes Ergebnis beisammen stehen (dies hat negative Effekte aufs Training, wie in Abschnitt 3.4.5 erwähnt wurde). Eine solche zu hohe Ordnung in einer Beispielliste kann etwa durch die automatisierte Erstellung der Beispiele geschehen. Der *ExamplesMixer* nimmt an, dass es für jedes Ergebnis ungefähr gleich viele Beispiele gibt. Er gibt die Beispiele dann so aus, dass die Ergebnisse immer abwechselnd von je einem Beispiel vorgegeben werden (bei drei verschiedenen Ergebnissen also in der Reihenfolge e_1, e_2, e_3, e_1 , usw.). Die Beispiele müssen in der Datei durch Kommas getrennt vorliegen, wobei der *ExamplesMixer* am Anfang noch eine kommagetrennte und eingeklammerte Liste mit den Handles der zu trainierenden Knoten erwartet. Die vom Programm erstellte Datei hat das gleiche Format. Die Aufrufsyntax des Programms lautet:

```
ExamplesMixer eingabedatei ausgabedatei
```

ExampleMinimizer

Das Programm *ExampleMinimizer* liest eine Datei mit Trainingsbeispielen ein und gibt in einer anderen Datei eine „minimierte“ Version aus. In dieser Version existiert für jede Situation (d.h., für jede Quellevidenzkombination) aus der ursprünglichen Datei nur noch *ein* Beispiel, wobei als Zielevidenzkombination diejenige ausgewählt wird, die in der Datei am häufigsten vorkam. Die neue Datei kann beispielsweise benutzt wer-

den, um die Performanz eines Netzwerks in Bezug auf die verschiedenen Situationen zu untersuchen, ohne auf die Wahrscheinlichkeit der einzelnen Situationen Rücksicht zu nehmen. Das Format der Ein- und Ausgabedatei ist das gleiche wie beim Example-Analyzer (s.o.). Die Aufrufsyntax ist:

```
ExampleMinimizer eingabedatei ausgabedatei [mit_header]
```

Konvertierungsprogramme

Es wurden auch zwei Programme erstellt, die Netzwerkdateien zwischen den Formaten von smile und Hugin konvertieren können. Sie lauten *smile2hugin* und *hugin2smile*, der Aufruf ist:

```
konvertierungsprogramm eingabedatei ausgabedatei
```

```
also beispielsweise hugin2smile mein_netzwerk.net mein_netzwerk.dsl.
```

A.1.6 Verzeichnis- und Projektstruktur

Die Bibliothek für Bayessche Netze befindet sich im Teilprojekt *BeliefNets*. Im *src*-Verzeichnis dort sind auch die erwähnten Hilfsprogramme. Die Adapter zum Weltmodell sind im eigenen Teilprojekt *BNWoMo* abgelegt. Innerhalb des Teilprojekts *WoMoDataProcessing* ist die Klasse `BallPossessionNet`, die als „Prozessor“ berechnet, wer den Ball hat. Das dafür notwendige Netzwerk ist im Verzeichnis *data* dieses Teilprojekts. Die neue Spielerarchitektur befindet sich im Teilprojekt *Player*. Dort sind auch die notwendigen Klassen für die auf Bayesschen Netzen beruhenden Entscheidungen. Die Bayesschen Netze selbst befinden sich wiederum im Verzeichnis *data* dieses Projekts.

A.2 Neue Spielerarchitektur

Während der Diplomarbeit wurde auch eine neue Architektur für den Spieler des CoPS-Teams entwickelt (für eine Einordnung des Spielers in der Gesamtarchitektur siehe Abbildung 1.2 auf Seite 3). Die bereits bestehende Architektur erwies sich als zu starr und inflexibel; insbesondere bestand keine Trennung zwischen den konzeptionell unterschiedlichen Vorgängen der Entscheidung über eine Aktion und der Ausführung durch den Navigator.

Die neu entwickelte Architektur zeigt Abbildung A.1. Die Spielerklasse selbst beinhaltet nur noch den Thread und Methoden zur Kontrolle von außen. Die Schnittstelle zum Navigator wird durch die Einführung einer Adapterklasse stabilisiert. Ausgelagert ist die Entscheidung, welcher Navigator-Befehl als nächstes ausgeführt werden sollte. Im Hauptprogramm legt man eine konkrete Unterklasse von `Decider` fest, die diese Entscheidungen trifft:

```
player->setDecider(new RuleBasedDecider());
```

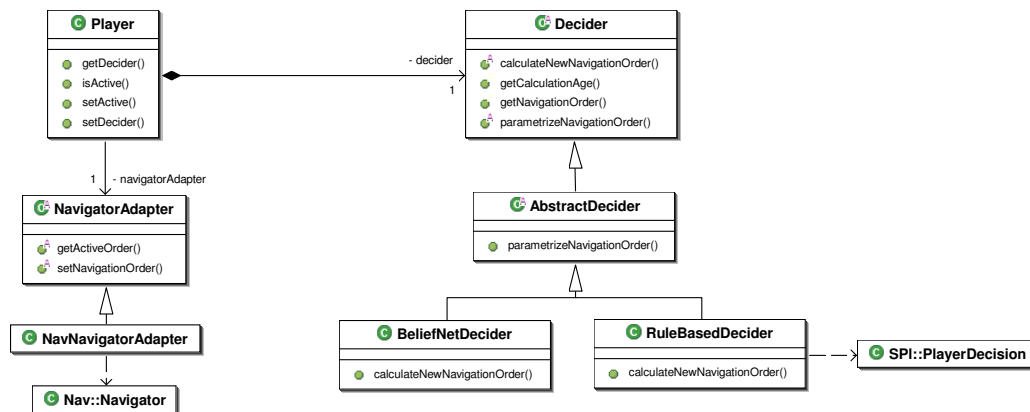


Abbildung A.1: Neue Architektur des Spielers

Diese Unterklassen können natürlich nur Fassaden für dahinterliegende komplexere Strukturen sein (gerade `RuleBasedDecider` ist ein einfacher Adapter für den bisherigen Spieler im Teilprojekt SP1).

In `Decider` ist ein Aufzählungstyp definiert, der als Rückgabewert für die kritische Methode `calculateNewNavigationOrder` dient. Seine möglichen Werte sind:

- `NO_CHANGE`: der neue Befehl ist gleich wie der aktuell ausgeführte Befehl (dieser wird der Methode als Parameter übergeben). In dem Fall muss der Navigator in der Ausführung nicht gestört werden.
- `PARAM_CHANGE`: der neue Befehl ist vom gleichen Typ wie der aktuell ausgeführte Befehl (z.B. „Suche den Ball“), aber ein oder mehrere Parameter des Befehls haben sich geändert. In diesem Fall kann man es vermeiden, dem Navigator einen neuen Befehl zu übergeben, und den Entscheider einfach den bisherigen Befehl mittels eines Aufrufs von `parametrizeNavigationOrder` anpassen lassen.
- `TYPE_CHANGE`: der neue Befehl ist von einem anderen Typ als der aktuell ausgeführte Befehl. Dem Navigator muss dieser neue Befehl (den man mit `getNavigationOrder` erhalten kann) übergeben werden.

Mit der neuen Architektur können verschiedene Arten der Entscheidungsfindung einfach hinzugefügt und miteinander verglichen werden; durch die Methoden in `Player` ist es theoretisch sogar möglich, dynamisch zwischen ihnen umzuschalten.

A.3 Spezielle Software für die hier vorgestellten Netze

A.3.1 Ballbesitz

Drei Programme wurden geschrieben, die den Umgang mit dem Netz zur Ermittlung des Ballbesitzes und sein Training zu erleichtern. Außerdem existiert natürlich Software zur Benutzung des Netzes.

Benutzung des Netzes

Im Teilprojekt *WoMoDataProcessing* existiert die Klasse *BallPossessionNet*. Diese ist ein *WoMoDataProcessor*. Sie lädt das Netz und führt regelmäßig Inferenz durch (die Berechnung gehört zum *CommonProcessorThread*). Das Ergebnis wird in Form eines int-Wertes, der den wahrscheinlichsten Zustand angibt, in ein *WoMoDataObj* geschrieben. *Source-* und *Feature-Type* lauten jeweils „BallPossession“. Die Zustände haben folgende Bedeutung: 0 – ich habe den Ball; 1 – ein Mitspieler von mir hat den Ball; 2 – ein Gegner hat den Ball; 3 – niemand hat den Ball; 4 – es ist nicht bekannt, wer den Ball hat (keine Sensorinformationen oder Fehler). Das verwendete Netz heißt „WhoHasTheBall.dsl“ und liegt im Unterverzeichnis *data*. Wird allerdings beim Aufruf eines Programmes nicht der Kommandozeilenparameter `--local` verwendet, so wird es im globalen Verzeichnis „\$PROJ_TOPDIR/interfaces/WoMoDataProcessing0-X/data/“ gesucht.

Testprogramm

Im Teilprojekt *SExec* liegt ein kleines Testprogramm namens *TestBallPossession*. Es ist dafür gedacht, im Simulator gestartet zu werden, wobei der Benutzer Testsituationen erzeugt. Sobald eine Änderung des Wertes im oben erwähnten *WoMoDataObj* auftritt, gibt das Programm auf der Konsole die Evidenzen der Sensorknoten und den neuen Wert des Objektes im Klartext aus.

Generierung von Trainingsbeispielen

Damit der Benutzer Trainingsbeispiele generieren kann, wurde das Programm *BPDataGenerator* erstellt, das ebenfalls in *SExec* liegt. Der Benutzer kann per Tastendruck festlegen, wer gerade im Ballbesitz ist und das Programm schreibt alle 400 Millisekunden ein Beispiel in die Datei „bpexamples.dat“ (falls sie schon existiert, wird sie erweitert).

Weil keine Kommandozeilenparameter übergeben werden können (Beschränkung durch die *CommandLineParameters*, die für die korrekte Initialisierung der Simulatordaten etc. gebraucht wird), muss durch Auskommentieren und Neuübersetzen entschieden

werden, welche Art von Beispielen generiert werden soll. Es gibt drei Arten von Beispielen mit jeweils einer Methode: Beispiele dafür, ob der Roboter selbst den Ball hat, Beispiele dafür, ob ein anderer Roboter den Ball hat, und Beispiele dafür, wer insgesamt im Ballbesitz ist. Siehe auch Abschnitt 4.3.2.

Training des Netzes

Das Training des Netzes wird durch das Programm *TrainBallPossession* durchgeführt, das sich im Teilprojekt *BeliefNets* befindet. Dieses ist durch Kommandozeilenparameter hochgradig konfigurierbar. Die Aufrufsyntax lautet:

```
TrainBallPossession (--learnHaveTheBall | --learnSomeoneBall
  | --learnWhoHasTheBall) Eingabernetzwerkdatei
  Beispieldatei Ausgabernetzwerkdatei
  [--polak-ribiere] [--showFalseExamples] [Schrittweite]
```

Die Bedeutung der Parameter ist folgende:

- `--learnHaveTheBall`: Trainiert, ob der Roboter selbst den Ball hat
- `--learnSomeoneBall`: Trainiert, ob ein anderer Roboter den Ball hat
- `--learnWhoHasTheBall`: Trainiert, wer sich im Ballbesitz befindet
- *Eingabernetzwerkdatei*: Datei mit dem ursprünglichen Netzwerk
- *Beispieldatei*: Datei mit den Trainingsbeispielen
- *Ausgabernetzwerkdatei*: Datei, in die das das Programm das trainierte Netzwerk schreibt
- `--polak-ribiere`: Gibt an, dass die Methode von Polak und Ribiere verwendet werden soll
- `--showFalseExamples`: Falls dieser Parameter angegeben wird, gibt das Programm *vor* dem Training alle falsch berechneten Beispiele auf der Konsole aus
- *Schrittweite*: Gibt die Schrittweite des Gradientenabstiegs an

Das Programm gibt auf jeden Fall vor und nach dem Training eine Statistik über die Gesamtzahl der Beispiele und über die Anzahl der korrekt berechneten Beispiele aus.

A.3.2 Schussentscheidung

Um den Umgang mit dem Netz zur Entscheidung, ob ein Torschuss oder ein Dribbling angebracht sind (siehe Abschnitt 4.4 ab Seite 55), zu erleichtern, wurden neben der normalen Software zur Benutzung drei weitere Programme geschrieben, deren Verwendung hier kurz erklärt wird. Die komplette Software für die Schussentscheidung befindet sich im Teilprojekt *Player*.

Benutzung des Netzes

Zur Benutzung des Netzes gibt es die Klassen *DribbleOrShoot*, *DribbleOrShootSingle* und *DribbleOrShootMultiple*. Die letzten beiden sind Spezialisierungen der Oberklasse *DribbleOrShoot*, jeweils für die Varianten des Netzes mit einem einzigen oder mit mehreren getrennten Knoten für die möglichen Entscheidungen. Welches Netz konkret benutzt wird, wird im Code der Klasse *DribbleOrShoot* festgelegt. Diese liefert gemäß dem Singletonmuster dann die richtige Instanz zurück.

Eine Berechnung der Entscheidung (durch Inferenz) muss explizit angefordert werden. Danach können sowohl die Entscheidung selbst (ein Aufzählungstyp) als auch die Wahrscheinlichkeiten für die möglichen Entscheidungen mit einfachen Zugriffsmethoden abgefragt werden. Die Benutzung der Bibliothek für die Bayesschen Netze und das korrekte Setzen von Sperrern geschieht intern und ist transparent.

Das verwendete Netzwerk heißt „DribbleOrShoot.dsl“ und liegt im Unterverzeichnis *data*. Falls jedoch beim Aufruf eines Programms nicht der Kommandozeilenparameter `--local` übergeben wird, so wird es im Verzeichnis „\$PROJ_TOPDIR/interfaces/Player0-X/data“ gesucht.

Test des Netzes

Die Funktion des Netzes kann mit dem Programm *TestDribbleOrShoot* getestet werden. Es ist dafür gedacht, im Simulator gestartet zu werden. Der Benutzer kann dort dann Situationen erzeugen, und das Programm gibt alle 2 Sekunden die Evidenz bei den Sensorknoten und die Entscheidung des Netzes aus. Das Programm funktioniert nur mit der Version des Netzes, die einen einzelnen Entscheidungsknoten hat.

Test von Beispielen

Hat man eine Datei mit Beispielen für ein Netz (entweder das mit einem oder das mit mehreren Endknoten), so kann man mittels des Programms *CheckDoSNetwork* ermitteln, wie gut ein Netz diese Beispiele berechnet. Der Aufruf des Programms lautet `CheckDoSNetwork <Netzwerkdatei> <Beispieldatei>`

Es gibt alle falsch berechneten Beispiele und eine kurze Statistik aus, aus der die Anzahl der Beispiele, die Anzahl der korrekt berechneten Beispiele und der jeweilige Anteil an Schussentscheidungen hervorgeht.

Training des Netzwerks

Das Training des Netzwerks wird mit dem Programm *TrainDribbleOrShoot* durchgeführt. Durch Ein- und Auskommentieren zweier Methoden kann es in zwei verschiedenen Modi laufen: entweder im 'online'-Modus, wobei „klassisches“ Verstärkungslernen

angewandt wird und gleichzeitig eine Beispieldatei generiert wird, oder im 'offline'-Modus, wobei das Netzwerk mit einer vorhandenen Beispieldatei trainiert wird.

Im 'online'-Modus wird zunächst das normale Netzwerk geladen und trainiert. Die ursprüngliche Datei bleibt erhalten, das ursprüngliche Netzwerk wird zudem in die Datei „dribbleorshoot_untrained.dsl“ geschrieben. Das Training erfolgt durch die Generierung von Situationen wie in Abschnitt 4.4 beschrieben. Es kann durch Eingabe von 'q' beendet werden. Dann werden das trainierte Netzwerk in die Datei „dribbleorshoot_trained.dsl“ geschrieben und die generierten Beispiele an die Datei „dosexamples.dat“ angehängt.

Im 'offline'-Modus wird das Netzwerk mit den Beispielen aus der Datei „dosexamples.dat“ trainiert und die trainierte Version des Netzwerks wird in die Datei „dribbleorshoot_offlinetrained.dsl“ geschrieben. Da das Netzwerk über den Standardweg geladen wird, ist auch in diesem Modus die Angabe des Kommandozeilenparameters `--local` notwendig.

A.3.3 Passspiel

Benutzung der Netze

Die Software zur Entscheidung über einen Pass befindet sich ebenfalls im Teilprojekt *Player*. Die beiden verwendeten Netze „UsePassPosition.dsl“ und „CanDribble.dsl“ (siehe Abschnitt 4.5) liegen wie üblich im Unterverzeichnis *data*. In der Datei `global.config` im Teilprojekt *DAT* sind zudem einige Konfigurationswerte abgelegt, welche die möglichen Zielpositionen festlegen: `pass_position_count` gibt die Anzahl der Positionen an, und zu jeder Position p gibt es die Werte `pass_position_x_p`, `pass_position_y_p` und `pass_position_t_p`, durch die die Position definiert wird (Angabe in Millimetern vom Spielfeldmittelpunkt und Dezigrad von der Richtung zum gegnerischen Tor).

Die Klasse *PassOrDribble* im Teilprojekt *Player* lädt die Netze und verbindet sie mit den entsprechenden Weltmodelldaten. Auf Anforderung führt sie Inferenz in allen Netzen aus (wohlgemerkt ein Netz pro Passposition). Danach liefert sie zurück, ob gepasst werden sollte, und außerdem entweder die zu verwendende Passposition oder auch die errechnete Wahrscheinlichkeit (konzeptionell: Nützlichkeit) für eine bestimmte Position. Weiterhin stellt sie auch Methoden zum Auslesen der vorhandenen Positionen bereit.

Test der Entscheidungen

Zum Test, ob der Entscheidungsprozess korrekt funktioniert, kann das Programm *Test-PassOrDribble* im Teilprojekt *Player* benutzt werden. Es kann im Simulator gestartet werden (sinnvollerweise mindestens zweifach, um zwei Roboter zu simulieren) und gibt alle drei Sekunden die Nützlichkeiten für die verschiedenen Positionen sowie die getroffene Entscheidung auf der Konsole aus.

Zur Generierung von Statistiken über den Anteil korrekt getroffener Entscheidungen ist das Programm *TrainPassOrDribble* vorgesehen (ursprünglich auch zum Training gedacht, daher der etwas irreführende Name). Wird es im Simulator gestartet, so kann der Benutzer – am besten unter gleichzeitiger Verwendung einer oder zwei Instanzen von *TestPassOrDribble* – Situationen generieren und an der Konsole vorgeben, ob Dribbling oder ein Pass zu einer bestimmten Situation die richtige Wahl sind. Beim Programmende gibt es dann die ermittelten Zahlen auf der Konsole aus.

Anhang B

Verwendete Netzwerke

Hier werden Auszüge aus den verwendeten Netzwerken im oben beschriebenen Dateiformat aufgelistet. Um den Umfang der Arbeit nicht unverhältnismäßig zu erhöhen, werden nur die CPTs der Knoten gezeigt. Die Netzwerke sind in der Reihenfolge, in der sie in der Arbeit besprochen wurden. Es werden jeweils nur die trainierten Versionen der Netzwerke abgedruckt.

B.1 Ballbesitz

```
net WhoHasTheBall
{
  node D_R1
  {
    PARENTS = ();
    DEFINITION =
    {
      NAMESTATES = (IrrelevantlySmall, VerySmall, Small, Big, Unknown);
      PROBABILITIES = (0.240631, 0.705112, 0.0349849, 0.0172717, 0.002);
    };
  };

  node S_D_R1
  {
    PARENTS = (D_R1);
    DEFINITION =
    {
      NAMESTATES = (IrrelevantlySmall, VerySmall, Small, Big, Unknown, NoValue);
      PROBABILITIES = (0.944, 0.05, 0, 0, 0.005,
0.001, 0.15, 0.599, 0.2, 0,
0.05, 0.001, 0, 0.16, 0.479,
0.16, 0.2, 0.001, 0, 0,
```

```

0.05, 0.449, 0.5, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
};
};

```

```

node V_R_1
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, Unknown);
    PROBABILITIES = (0.964248, 8.49e-06, 0.0141936, 0.0195499, 0.002);
  };
};

```

```

node S_VR1
{
  PARENTS = (V_R_1);
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, Unknown, NoValue);
    PROBABILITIES = (0.889, 0.05, 0.01, 0, 0.05,
0.001, 0.07, 0.739, 0.1, 0.01,
0.08, 0.001, 0.005, 0.12, 0.514,
0.16, 0.2, 0.001, 0.005, 0.01,
0.15, 0.335, 0.499, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
  };
};

```

```

node IBall
{
  PARENTS = (V_R_1, D_R1);
  DEFINITION =
  {
    NAMESTATES = (Yes, No, Unknown);
    PROBABILITIES = (0.9998, 0.0001, 0.0001, 0.332504, 0.667496,
0, 0.169347, 0.830653, 0, 0.0947021,
0.860596, 0.0447021, 0.0999808, 0.500046, 0.399973,
0.905773, 0.094227, 0, 0.884736, 0.115264,
0, 0.597132, 0.402868, 0, 0.0990953,
0.851809, 0.0490953, 0.0999996, 0.500001, 0.4,
0.508483, 0.491517, 0, 0.511505, 0.488495,
0, 0.393909, 0.606091, 0, 0.0484932,
0.853014, 0.0984932, 0.0999959, 0.500008, 0.399996,

```

```

0.129413, 0.870587, 0, 0.162495, 0.837505,
0, 0.0991105, 0.85178, 0.0491099, 0,
0.900598, 0.099402, 0.499998, 0.500002, 0,
0.79997, 0.150061, 0.0499694, 0.699995, 0.200013,
0.0999927, 0.599999, 0.200002, 0.199999, 0.0999999,
0.7, 0.2, 0.05, 0.3, 0.65);
};
};

```

```

node V_R_2
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, Unknown);
    PROBABILITIES = (0.964248, 8.49e-06, 0.0141936, 0.0195499, 0.002);
  };
};

```

```

node S_VR2
{
  PARENTS = (V_R_2);
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, Unknown, NoValue);
    PROBABILITIES = (0.889, 0.05, 0.01, 0, 0.05,
0.001, 0.07, 0.739, 0.1, 0.01,
0.08, 0.001, 0.005, 0.12, 0.514,
0.16, 0.2, 0.001, 0.005, 0.01,
0.15, 0.335, 0.499, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
  };
};

```

```

node V_R_3
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, Unknown);
    PROBABILITIES = (0.964248, 8.49e-06, 0.0141936, 0.0195499, 0.002);
  };
};

```

```

node S_VR3

```

```

{
  PARENTS = (V_R_3);
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, Unknown, NoValue);
    PROBABILITIES = (0.889, 0.05, 0.01, 0, 0.05,
0.001, 0.07, 0.739, 0.1, 0.01,
0.08, 0.001, 0.005, 0.12, 0.514,
0.16, 0.2, 0.001, 0.005, 0.01,
0.15, 0.335, 0.499, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
  };
};

node D_R2
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, VerySmall, Small, Big, Unknown);
    PROBABILITIES = (0.240631, 0.705112, 0.0349849, 0.0172717, 0.002);
  };
};

node S_D_R2
{
  PARENTS = (D_R2);
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, VerySmall, Small, Big, Unknown, NoValue);
    PROBABILITIES = (0.944, 0.05, 0, 0, 0.005,
0.001, 0.15, 0.599, 0.2, 0,
0.05, 0.001, 0, 0.16, 0.479,
0.16, 0.2, 0.001, 0, 0,
0.05, 0.449, 0.5, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
  };
};

node D_R3
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, VerySmall, Small, Big, Unknown);

```

```

        PROBABILITIES = (0.240631, 0.705112, 0.0349849, 0.0172717, 0.002);
    };
};

node S_D_R3
{
    PARENTS = (D_R3);
    DEFINITION =
    {
        NAMESTATES = (IrrelevantlySmall, VerySmall, Small, Big, Unknown, NoValue);
        PROBABILITIES = (0.944, 0.05, 0, 0, 0.005,
0.001, 0.15, 0.599, 0.2, 0,
0.05, 0.001, 0, 0.16, 0.479,
0.16, 0.2, 0.001, 0, 0,
0.05, 0.449, 0.5, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
    };
};

node IBall2
{
    PARENTS = (V_R_2, D_R2);
    DEFINITION =
    {
        NAMESTATES = (Yes, No, Unknown);
        PROBABILITIES = (0.9998, 0.0001, 0.0001, 0.332504, 0.667496,
0, 0.169347, 0.830653, 0, 0.0947021,
0.860596, 0.0447021, 0.0999808, 0.500046, 0.399973,
0.905773, 0.094227, 0, 0.884736, 0.115264,
0, 0.597132, 0.402868, 0, 0.0990953,
0.851809, 0.0490953, 0.0999996, 0.500001, 0.4,
0.508483, 0.491517, 0, 0.511505, 0.488495,
0, 0.393909, 0.606091, 0, 0.0484932,
0.853014, 0.0984932, 0.0999959, 0.500008, 0.399996,
0.129413, 0.870587, 0, 0.162495, 0.837505,
0, 0.0991105, 0.85178, 0.0491099, 0,
0.900598, 0.099402, 0.499998, 0.500002, 0,
0.79997, 0.150061, 0.0499694, 0.699995, 0.200013,
0.0999927, 0.599999, 0.200002, 0.199999, 0.0999999,
0.7, 0.2, 0.05, 0.3, 0.65);
    };
};

node HeBall2
{

```

```

PARENTS = (V_R_3, D_R3);
DEFINITION =
{
  NAMESTATES = (Yes, No, Unknown);
  PROBABILITIES = (0.9998, 0.0001, 0.0001, 0.332504, 0.667496,
0, 0.169347, 0.830653, 0, 0.0947021,
0.860596, 0.0447021, 0.0999808, 0.500046, 0.399973,
0.905773, 0.094227, 0, 0.884736, 0.115264,
0, 0.597132, 0.402868, 0, 0.0990953,
0.851809, 0.0490953, 0.0999996, 0.500001, 0.4,
0.508483, 0.491517, 0, 0.511505, 0.488495,
0, 0.393909, 0.606091, 0, 0.0484932,
0.853014, 0.0984932, 0.0999959, 0.500008, 0.399996,
0.129413, 0.870587, 0, 0.162495, 0.837505,
0, 0.0991105, 0.85178, 0.0491099, 0,
0.900598, 0.099402, 0.499998, 0.500002, 0,
0.79997, 0.150061, 0.0499694, 0.699995, 0.200013,
0.0999927, 0.599999, 0.200002, 0.199999, 0.0999999,
0.7, 0.2, 0.05, 0.3, 0.65);
};
};

node TeamBall
{
  PARENTS = (HeBall12, IBall12);
  DEFINITION =
  {
    NAMESTATES = (Yes, No, Unknown);
    PROBABILITIES = (0.848922, 0.127573, 0.0235055, 0.874535, 0.117707,
0.00775682, 0.972945, 0.0217025, 0.00535231, 0.817798,
0.140823, 0.04138, 0.0269045, 0.972934, 0.000162121,
0.000262928, 0.00799347, 0.991744, 0.91849, 0.0617201,
0.0197902, 0.000200452, 0.0362327, 0.963567, 1.87581e-06,
0.00227002, 0.997728);
  };
};

node VB
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Medium, High, Unknown);
    PROBABILITIES = (0.4, 0.4, 0.155, 0.04, 0.005);
  };
};

```

```

};

node S_VB
{
  PARENTS = (VB);
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Medium, High, Unknown, NoValue);
    PROBABILITIES = (0.4989, 0.35, 0.1, 0.0001, 0.05,
0.001, 0.2, 0.399, 0.3, 0.05,
0.05, 0.001, 0.05, 0.299, 0.3,
0.15, 0.2, 0.001, 0, 0.05,
0.15, 0.499, 0.3, 0.001, 0.001,
0.001, 0.001, 0.001, 0.005, 0.991);
  };
};

node D_RO
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (VerySmall, Small, Medium, Big, Unknown);
    PROBABILITIES = (0.101, 0.398, 0.298, 0.197, 0.006);
  };
};

node S_D_RO
{
  PARENTS = (D_RO);
  DEFINITION =
  {
    NAMESTATES = (VerySmall, Small, Medium, Big, Unknown, NoValue);
    PROBABILITIES = (0.399, 0.2, 0.05, 0, 0.35,
0.001, 0.2, 0.299, 0.2, 0.05,
0.25, 0.001, 0.02, 0.08, 0.399,
0.2, 0.3, 0.001, 0, 0.05,
0.15, 0.4, 0.399, 0.001, 0.001,
0.001, 0.001, 0.001, 0.002, 0.994);
  };
};

node SomeoneBall
{
  PARENTS = (VB, D_RO);

```

```

DEFINITION =
{
  NAMESTATES = (Yes, No, Unknown);
  PROBABILITIES = (0.9499, 0.05, 0.0001, 0.7499, 0.25,
0.0001, 0.2, 0.7999, 0.0001, 0.05,
0.9499, 0.0001, 0.15, 0.3, 0.55,
0.9998, 0.0001, 0.0001, 0.7999, 0.2,
0.0001, 0.3, 0.6999, 0.0001, 0.05,
0.9499, 0.0001, 0.2, 0.3, 0.5,
0.7999, 0.2, 0.0001, 0.5999, 0.4,
0.0001, 0.3, 0.6999, 0.0001, 0.05,
0.9499, 0.0001, 0.15, 0.3, 0.55,
0.6999, 0.3, 0.0001, 0.5, 0.4999,
0.0001, 0.15, 0.8499, 0.0001, 0.0001,
0.9998, 0.0001, 0.1, 0.3, 0.6,
0.75, 0.15, 0.1, 0.3, 0.6,
0.1, 0.1, 0.8, 0.1, 0.05,
0.85, 0.1, 0.1, 0.2, 0.7);
};
};

node WhoBall
{
  PARENTS = (TeamBall, IBall, SomeoneBall);
  DEFINITION =
  {
    NAMESTATES = (I, Partners, Opponents, Nobody, Unknown);
    PROBABILITIES = (0.712324, 0.167282, 0.0982302, 0.0220649, 0.0001,
0.85191, 0.0393077, 0.0208079, 0.087874, 0.0001,
0.899673, 0.100007, 0.000109014, 0.000111086, 0.0001,
0.00012607, 0.638941, 0.324383, 0.0364493, 0.0001,
0.000343488, 0.457244, 0.186872, 0.305541, 0.05,
1.43178e-05, 0.849753, 0.0500894, 0.0500443, 0.05,
3.86263e-10, 0.947099, 0.0527011, 9.98283e-05, 0.0001,
0.00110133, 0.84351, 0.00774232, 0.147546, 0.0001,
0.00999775, 0.929999, 0.0200024, 0.0200007, 0.02,
0.773382, 0.168743, 0.054833, 0.00294095, 0.0001,
0.920747, 0.0662365, 0.00396013, 0.00895676, 0.0001,
0.959781, 0.000123521, 0.0299973, 9.89731e-05, 0.01,
0.000422797, 0.0482656, 0.930945, 0.0202659, 0.0001,
2.48643e-09, 0.0123946, 0.0653873, 0.922118, 0.0001,
5.03513e-05, 8.68913e-05, 0.150044, 0.849719, 0.0001,
0.00193007, 0.00367425, 0.953906, 0.000489162, 0.04,
0.0438505, 0.000282665, 0.10336, 0.352607, 0.4999,
0.0499986, 9.99572e-05, 0.100001, 0.200001, 0.6499,

```

```

0.908649, 0.00277338, 0.0543445, 0.0341328, 0.0001,
0.94732, 0.00133524, 0.023269, 0.027976, 0.0001,
0.969892, 0.00999305, 0.0100069, 0.000109223, 0.01,
2.71784e-05, 0.191511, 0.697428, 0.0610345, 0.05,
0.000175362, 0.000988205, 0.0649979, 0.883838, 0.05,
6.91134e-05, 0.119972, 0.230036, 0.549923, 0.1,
0.0273, 0.198014, 0.303899, 0.000886421, 0.4699,
0.0272799, 0.0975491, 0.153975, 0.301197, 0.42,
0.0299993, 0.0599995, 0.15, 0.2, 0.56);
    };
};
};

```

B.2 Schussentscheidung

B.2.1 Netzwerk mit einem Zielknoten, für statischen Torwart

```

net DribbleOrShoot
{
node GoalHit
{
PARENTS = ();
DEFINITION =
{
NAMESTATES = (Yes, No, Error);
PROBABILITIES = (0.20000000, 0.79900000, 0.00100000);
};
};

node S_GoalHit
{
PARENTS = (GoalHit);
DEFINITION =
{
NAMESTATES = (Yes, No, Possibly, Unknown, NoValue);
PROBABILITIES = (0.58900000, 0.10000000, 0.30000000, 0.01000000,
0.00100000, 0.01000000, 0.48900000, 0.10000000, 0.40000000,
0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.20000000,
0.79700000);
};
};
};

```

```

node MySpeed
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (None, Small, Mediocre, Big, Error);
    PROBABILITIES = (0.30000000, 0.39900000, 0.20000000, 0.10000000,
    0.00100000);
  };
};

node S_MySpeed
{
  PARENTS = (MySpeed);
  DEFINITION =
  {
    NAMESTATES = (IrrelevantlySmall, Small, Mediocre, Big, NoValue);
    PROBABILITIES = (0.80000000, 0.19700000, 0.00100000, 0.00100000,
    0.00100000, 0.15000000, 0.69800000, 0.15000000, 0.00100000,
    0.00100000, 0.00100000, 0.20000000, 0.59800000, 0.20000000,
    0.00100000, 0.00100000, 0.00100000, 0.25000000, 0.74700000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.00010000,
    0.99690000);
  };
};

node ObstXDist2
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (DirectlyInFront, VeryNear, Near, Mediocre, Far,
    Error);
    PROBABILITIES = (0.10000000, 0.20000000, 0.25000000, 0.25000000,
    0.19900000, 0.00100000);
  };
};

node ObstYDist2
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (Blocking, PartlyBlocking, NearTheWay, Mediocre, Far,
    Error);
  };
};

```

```

    PROBABILITIES = (0.29900000, 0.30000000, 0.15000000, 0.10000000,
    0.15000000, 0.00100000);
};
};

```

```

node GoalDist
{
    PARENTS = ();
    DEFINITION =
    {
        NAMESTATES = (VeryNear, Near, QuiteNear, Mediocre, Far, Error);
        PROBABILITIES = (0.10000000, 0.15000000, 0.15000000, 0.15000000,
        0.44900000, 0.00100000);
    };
};

```

```

node WayFree
{
    PARENTS = (ObstXDist2, ObstYDist2);
    DEFINITION =
    {
        NAMESTATES = (CompletelyFree, FreeIfAccel, FreeNotAccel, Blocked);
        PROBABILITIES = (0.00413819, 0.00359106, 0.00501830, 0.98725246,
        0.00746248, 0.08503821, 0.39343124, 0.51406807, 0.24485874,
        0.61860809, 0.08983374, 0.04669943, 0.62637204, 0.27988915,
        0.05188652, 0.04185228, 0.83042204, 0.08538690, 0.00274219,
        0.08144888, 0.04999984, 0.19999300, 0.20000138, 0.55000578,
        0.14273109, 0.13121738, 0.07689534, 0.64915619, 0.13724591,
        0.19076567, 0.23959825, 0.43239015, 0.18460792, 0.31442673,
        0.17654453, 0.32442083, 0.44202605, 0.42999161, 0.04891142,
        0.07907093, 0.65512146, 0.10472425, 0.10031372, 0.13984058,
        0.19999976, 0.39999055, 0.15000149, 0.25000821, 0.23710650,
        0.25154400, 0.03591822, 0.47543128, 0.38065926, 0.39116610,
        0.08295007, 0.14522458, 0.28178382, 0.60152364, 0.01995640,
        0.09673613, 0.45675977, 0.32285044, 0.09655213, 0.12383766,
        0.70893723, 0.09305740, 0.06209220, 0.13591318, 0.25000317,
        0.24999649, 0.24999632, 0.25000402, 0.16835716, 0.10849754,
        0.05415042, 0.66899487, 0.48470412, 0.47322393, 0.01258693,
        0.02948502, 0.30531562, 0.49140486, 0.05779167, 0.14548785,
        0.64249874, 0.31795258, 0.00482807, 0.03472061, 0.76052178,
        0.10326393, 0.01062382, 0.12559048, 0.60000713, 0.35000147,
        0.00099063, 0.04900077, 0.32197306, 0.03891488, 0.16016138,
        0.47895070, 0.55004888, 0.43895685, 0.00584984, 0.00514442,
        0.34636847, 0.57667256, 0.02384790, 0.05311108, 0.57451729,
        0.41110980, 0.00285668, 0.01151623, 0.64590552, 0.13942162,

```

```

    0.07074680, 0.14392608, 0.40000353, 0.30000108, 0.00999526,
    0.29000013, 0.05000083, 0.09999357, 0.19999806, 0.65000753,
    0.10000357, 0.29999738, 0.29998331, 0.30001574, 0.25000334,
    0.59999350, 0.04997470, 0.10002847, 0.49999962, 0.39996509,
    0.00099882, 0.09903647, 0.79999700, 0.15982824, 0.00103990,
    0.03913486, 0.30000000, 0.04999999, 0.25000000, 0.40000001);
};
};

node ObstXDist
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (DirectlyInFront, VeryNear, Near, Mediocre, Far,
    Error);
    PROBABILITIES = (0.10000000, 0.20000000, 0.25000000, 0.25000000,
    0.19900000, 0.00100000);
  };
};

node ObstYDist
{
  PARENTS = ();
  DEFINITION =
  {
    NAMESTATES = (Blocking, PartlyBlocking, NearTheWay, Mediocre, Far,
    Error);
    PROBABILITIES = (0.29900000, 0.30000000, 0.15000000, 0.10000000,
    0.15000000, 0.00100000);
  };
};

node CanDribble
{
  PARENTS = (MySpeed, ObstXDist, ObstYDist);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.05055071, 0.94944929, 0.20213747, 0.79786253,
    0.50511645, 0.49488355, 0.81246974, 0.18753026, 0.99775242,
    0.00224758, 0.05000256, 0.94999744, 0.30387369, 0.69612631,
    0.41057379, 0.58942621, 0.56702489, 0.43297511, 0.87447346,
    0.12552654, 0.99891393, 0.00108607, 0.30000369, 0.69999631,
    0.49604099, 0.50395901, 0.55366860, 0.44633140, 0.64716465,

```

0.35283535, 0.85742126, 0.14257874, 0.96497025, 0.03502975,
0.50000069, 0.49999931, 0.68936017, 0.31063983, 0.72755808,
0.27244192, 0.84361942, 0.15638058, 0.86628674, 0.13371326,
0.89478756, 0.10521244, 0.89999699, 0.10000301, 0.99252899,
0.00747101, 0.96674447, 0.03325553, 0.97206380, 0.02793620,
0.98112207, 0.01887793, 0.96158190, 0.03841810, 0.99899908,
0.00100092, 0.60000015, 0.39999985, 0.60000136, 0.39999864,
0.60000455, 0.39999545, 0.60001251, 0.39998749, 0.60006193,
0.39993807, 0.60000000, 0.40000000, 0.05149758, 0.94850242,
0.10517823, 0.89482177, 0.21226589, 0.78773411, 0.66383433,
0.33616567, 0.93659348, 0.06340652, 0.05000161, 0.94999839,
0.20556804, 0.79443196, 0.28339550, 0.71660450, 0.35589216,
0.64410784, 0.68362558, 0.31637442, 0.93869322, 0.06130678,
0.20000221, 0.79999779, 0.34887383, 0.65112617, 0.55206643,
0.44793357, 0.69653231, 0.30346769, 0.79841759, 0.20158241,
0.91161181, 0.08838819, 0.50000146, 0.49999854, 0.76920839,
0.23079161, 0.84163177, 0.15836823, 0.95030813, 0.04969187,
0.94969074, 0.05030926, 0.96651017, 0.03348983, 0.75000180,
0.24999820, 0.96179835, 0.03820165, 0.96453601, 0.03546399,
0.98961534, 0.01038466, 0.97174320, 0.02825680, 0.97205997,
0.02794003, 0.95000138, 0.04999862, 0.50000088, 0.49999912,
0.50000462, 0.49999538, 0.50001206, 0.49998794, 0.50001431,
0.49998569, 0.50003871, 0.49996129, 0.50000000, 0.50000000,
0.00889542, 0.99110458, 0.11571105, 0.88428895, 0.43118912,
0.56881088, 0.81806342, 0.18193658, 0.97194611, 0.02805389,
0.00500126, 0.99499874, 0.11390855, 0.88609145, 0.19627713,
0.80372287, 0.37559322, 0.62440678, 0.63914105, 0.36085895,
0.98094615, 0.01905385, 0.10000229, 0.89999771, 0.26937550,
0.73062450, 0.36730412, 0.63269588, 0.58635541, 0.41364459,
0.90389174, 0.09610826, 0.98640414, 0.01359586, 0.20000300,
0.79999700, 0.66656128, 0.33343872, 0.71650798, 0.28349202,
0.85596021, 0.14403979, 0.95809594, 0.04190406, 0.98552973,
0.01447027, 0.60000338, 0.39999662, 0.86781592, 0.13218408,
0.82915879, 0.17084121, 0.83631547, 0.16368453, 0.86504299,
0.13495701, 0.89362056, 0.10637944, 0.85000143, 0.14999857,
0.30000567, 0.69999433, 0.30001541, 0.69998459, 0.50002992,
0.49997008, 0.70001826, 0.29998174, 0.90002375, 0.09997625,
0.30000000, 0.70000000, 0.00580861, 0.99419139, 0.01722968,
0.98277032, 0.23482832, 0.76517168, 0.62177401, 0.37822599,
0.83292466, 0.16707534, 0.20000175, 0.79999825, 0.05197664,
0.94802336, 0.11257203, 0.88742797, 0.33939184, 0.66060816,
0.57986651, 0.42013349, 0.90130336, 0.09869664, 0.05000249,
0.94999751, 0.21052067, 0.78947933, 0.19650474, 0.80349526,
0.28607716, 0.71392284, 0.65202191, 0.34797809, 0.90937413,
0.09062587, 0.25000153, 0.74999847, 0.54232887, 0.45767113,

```

0.29518095, 0.70481905, 0.27489862, 0.72510138, 0.60465970,
0.39534030, 0.91943399, 0.08056601, 0.49999955, 0.50000045,
0.95539329, 0.04460671, 0.81894888, 0.18105112, 0.77961475,
0.22038525, 0.90814185, 0.09185815, 0.98968598, 0.01031402,
0.94999938, 0.05000062, 0.20000778, 0.79999222, 0.20001504,
0.79998496, 0.35003151, 0.64996849, 0.65002135, 0.34997865,
0.85003524, 0.14996476, 0.20000000, 0.80000000, 0.24000002,
0.75999998, 0.30000009, 0.69999991, 0.40000018, 0.59999982,
0.65000015, 0.34999985, 0.95000036, 0.04999964, 0.40000000,
0.60000000, 0.30000013, 0.69999987, 0.40000044, 0.59999956,
0.50000065, 0.49999935, 0.65000037, 0.34999963, 0.95000050,
0.04999950, 0.50000000, 0.50000000, 0.60000037, 0.39999963,
0.60000073, 0.39999927, 0.60000105, 0.39999895, 0.70000055,
0.29999945, 0.95000030, 0.04999970, 0.60000000, 0.40000000,
0.80000041, 0.19999959, 0.80000055, 0.19999945, 0.80000110,
0.19999890, 0.80000056, 0.19999944, 0.95000019, 0.04999981,
0.80000000, 0.20000000, 0.95000014, 0.04999986, 0.95000013,
0.04999987, 0.95000037, 0.04999963, 0.95000023, 0.04999977,
0.95000020, 0.04999980, 0.95000000, 0.05000000, 0.70000000,
0.30000000, 0.70000000, 0.30000000, 0.70000000, 0.30000000,
0.70000000, 0.30000000, 0.70000000, 0.30000000, 0.70000000,
0.30000000);
};
};

node ShotSuccess
{
  PARENTS = (GoalDist, GoalHit);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.51212589, 0.48787411, 0.14959854, 0.85040146,
0.60000069, 0.39999931, 0.88443249, 0.11556751, 0.21487675,
0.78512325, 0.37000962, 0.62999038, 0.61607644, 0.38392356,
0.11821575, 0.88178425, 0.10005283, 0.89994717, 0.25992677,
0.74007323, 0.17795773, 0.82204227, 0.01010418, 0.98989582,
0.60338492, 0.39661508, 0.30377634, 0.69622366, 0.15012767,
0.84987233, 0.50000103, 0.49999897, 0.49999693, 0.50000307,
0.50000000, 0.50000000);
  };
};

node Shoot
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);

```

```

DEFINITION =
{
  NAMESTATES = (ShootWithAccel, ShootSlow, Dribble);
  PROBABILITIES = (0.48460646, 0.00031899, 0.51507455, 0.53082277,
0.00000006, 0.46917716, 0.85269396, 0.00003625, 0.14726978,
0.63911991, 0.00004242, 0.36083768, 0.61061012, 0.00037247,
0.38901739, 0.21071749, 0.00016989, 0.78911262, 0.79983054,
0.00000032, 0.20016913, 0.68851062, 0.00000000, 0.31148938,
0.08490578, 0.00006070, 0.91503350, 0.11137030, 0.00000003,
0.88862967, 0.25685120, 0.61185250, 0.13129629, 0.09791991,
0.41965685, 0.48242322, 0.18783316, 0.07649620, 0.73567063,
0.23143085, 0.00030319, 0.76826594, 0.35895782, 0.08685961,
0.55418256, 0.26728835, 0.07564368, 0.65706797);
};
};

node S_ObstXDist
{
  PARENTS = (ObstXDist);
  DEFINITION =
  {
    NAMESTATES = (DirectlyInFront, VeryNear, Near, Mediocre, Far,
    NoValue);
    PROBABILITIES = (0.94600000, 0.05000000, 0.00100000, 0.00100000,
0.00100000, 0.00100000, 0.39600000, 0.40100000, 0.20000000,
0.00100000, 0.00100000, 0.00100000, 0.04000000, 0.30000000,
0.45800000, 0.20000000, 0.00100000, 0.00100000, 0.00100000,
0.05000000, 0.20000000, 0.55000000, 0.19800000, 0.00100000,
0.00100000, 0.00100000, 0.00100000, 0.29600000, 0.70000000,
0.00100000, 0.10000000, 0.00100000, 0.00100000, 0.00100000,
0.00500000, 0.89200000);
  };
};

node S_ObstXDist2
{
  PARENTS = (ObstXDist2);
  DEFINITION =
  {
    NAMESTATES = (DirectlyInFront, VeryNear, Near, Mediocre, Far,
    NoValue);
    PROBABILITIES = (0.94600000, 0.05000000, 0.00100000, 0.00100000,
0.00100000, 0.00100000, 0.39600000, 0.40100000, 0.20000000,
0.00100000, 0.00100000, 0.00100000, 0.04000000, 0.30000000,
0.45800000, 0.20000000, 0.00100000, 0.00100000, 0.00100000,

```

```

    0.05000000, 0.20000000, 0.55000000, 0.19800000, 0.00100000,
    0.00100000, 0.00100000, 0.00100000, 0.29600000, 0.70000000,
    0.00100000, 0.10000000, 0.00100000, 0.00100000, 0.00100000,
    0.00500000, 0.89200000);
};
};

node S_ObstYDist2
{
  PARENTS = (ObstYDist2);
  DEFINITION =
  {
    NAMESTATES = (Blocking, PartlyBlocking, NearTheWay, Mediocre, Far,
    NoValue);
    PROBABILITIES = (0.80100000, 0.19500000, 0.00100000, 0.00100000,
    0.00100000, 0.00100000, 0.39600000, 0.50100000, 0.10000000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.25000000,
    0.49700000, 0.25000000, 0.00100000, 0.00100000, 0.00100000,
    0.05000000, 0.25000000, 0.49800000, 0.20000000, 0.00100000,
    0.00100000, 0.00100000, 0.00100000, 0.19600000, 0.80000000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.00100000,
    0.00500000, 0.99100000);
  };
};

node S_GoalDist
{
  PARENTS = (GoalDist);
  DEFINITION =
  {
    NAMESTATES = (VeryNear, Near, QuiteNear, Mediocre, Far, Unknown,
    NoValue);
    PROBABILITIES = (0.88000000, 0.09600000, 0.02000000, 0.00100000,
    0.00100000, 0.00100000, 0.00100000, 0.15000000, 0.59800000,
    0.15000000, 0.05000000, 0.00100000, 0.05000000, 0.00100000,
    0.00100000, 0.20000000, 0.40000000, 0.24800000, 0.05000000,
    0.10000000, 0.00100000, 0.00100000, 0.01000000, 0.15000000,
    0.48800000, 0.20000000, 0.15000000, 0.00100000, 0.00100000,
    0.00100000, 0.01000000, 0.20000000, 0.40000000, 0.38700000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.00100000,
    0.00100000, 0.20000000, 0.79500000);
  };
};

node S_ObstYDist

```

```

{
  PARENTS = (ObstYDist);
  DEFINITION =
  {
    NAMESTATES = (Blocking, PartlyBlocking, NearTheWay, Mediocre, Far,
    No_Value);
    PROBABILITIES = (0.80100000, 0.19500000, 0.00100000, 0.00100000,
    0.00100000, 0.00100000, 0.39600000, 0.50100000, 0.10000000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.25000000,
    0.49700000, 0.25000000, 0.00100000, 0.00100000, 0.00100000,
    0.05000000, 0.25000000, 0.49800000, 0.20000000, 0.00100000,
    0.00100000, 0.00100000, 0.00100000, 0.19600000, 0.80000000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.00100000,
    0.00500000, 0.99100000);
  };
};
};
};

```

B.2.2 Netzwerk mit einem Zielknoten, für sich regelförmig bewegenden Torwart

Es sind nur die Unterschiede zum Netzwerk für den statischen Torwart aufgeführt.

```

net DribbleOrShoot
{
  node WayFree
  {
    PARENTS = (ObstXDist2, ObstYDist2);
    DEFINITION =
    {
      NAMESTATES = (CompletelyFree, FreeIfAccel, FreeNotAccel, Blocked);
      PROBABILITIES = (0.00717357, 0.00752309, 0.00653851, 0.97876483,
      0.00791769, 0.06448558, 0.42898236, 0.49861436, 0.15685516,
      0.55045197, 0.23131242, 0.06138044, 0.54052563, 0.23961725,
      0.17446231, 0.04539481, 0.79132417, 0.00450435, 0.12882211,
      0.07534940, 0.04999543, 0.19999490, 0.20000592, 0.55000374,
      0.09628343, 0.11716506, 0.09349079, 0.69306071, 0.09710944,
      0.22583394, 0.30892295, 0.36813365, 0.01840500, 0.17522146,
      0.54110508, 0.26526846, 0.30261728, 0.37860897, 0.25865927,
      0.06011448, 0.67783617, 0.12923874, 0.11192756, 0.08099752,
      0.19999728, 0.39999585, 0.15000346, 0.25000341, 0.19631280,
      0.11523008, 0.18117102, 0.50728610, 0.25608326, 0.25290222,
      0.14225309, 0.34876142, 0.01585158, 0.29075893, 0.44906660,
      0.24432288, 0.23796037, 0.14895381, 0.44491138, 0.16817443,

```

```

0.71600313, 0.24837199, 0.00584052, 0.02978435, 0.24999934,
0.24999948, 0.24999902, 0.25000216, 0.32899387, 0.24310553,
0.01989942, 0.40800119, 0.45854432, 0.22876944, 0.04853963,
0.26414661, 0.55715929, 0.35700391, 0.02736554, 0.05847126,
0.64108102, 0.18372832, 0.04643036, 0.12876029, 0.76523252,
0.16925233, 0.01443030, 0.05108485, 0.60000339, 0.35000384,
0.00099117, 0.04900159, 0.53743933, 0.29649863, 0.00921679,
0.15684524, 0.57122886, 0.24043576, 0.02333673, 0.16499865,
0.61845620, 0.34993858, 0.00955137, 0.02205385, 0.68080644,
0.26057315, 0.00152949, 0.05709091, 0.66004550, 0.26111933,
0.00429301, 0.07454215, 0.40000374, 0.30000165, 0.00999326,
0.29000136, 0.05000280, 0.10000565, 0.19999367, 0.64999786,
0.09997200, 0.29997238, 0.30003506, 0.30002056, 0.24990930,
0.59990011, 0.05012940, 0.10006117, 0.49990986, 0.39991294,
0.00112814, 0.09904904, 0.79988881, 0.15988214, 0.00114378,
0.03908528, 0.30000000, 0.05000000, 0.25000000, 0.40000000);
};
};

node CanDribble
{
  PARENTS = (MySpeed, ObstXDist, ObstYDist);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.04924356, 0.95075644, 0.20870757, 0.79129243,
0.52638956, 0.47361044, 0.80836668, 0.19163332, 0.95731419,
0.04268581, 0.05000023, 0.94999977, 0.29906834, 0.70093166,
0.39971120, 0.60028880, 0.54346309, 0.45653691, 0.83558817,
0.16441183, 0.95782105, 0.04217895, 0.29999918, 0.70000082,
0.49684184, 0.50315816, 0.54397686, 0.45602314, 0.54851482,
0.45148518, 0.76527965, 0.23472035, 0.90697308, 0.09302692,
0.49999645, 0.50000355, 0.69203043, 0.30796957, 0.77468149,
0.22531851, 0.80077561, 0.19922439, 0.80026555, 0.19973445,
0.85201247, 0.14798753, 0.89999569, 0.10000431, 0.99327308,
0.00672692, 0.99086382, 0.00913618, 0.97143753, 0.02856247,
0.96801408, 0.03198592, 0.97066394, 0.02933606, 0.99899881,
0.00100119, 0.59999905, 0.40000095, 0.60000890, 0.39999110,
0.60002688, 0.39997312, 0.60000805, 0.39999195, 0.59999707,
0.40000293, 0.60000000, 0.40000000, 0.04589901, 0.95410099,
0.09506804, 0.90493196, 0.21296098, 0.78703902, 0.65510365,
0.34489635, 0.89446013, 0.10553987, 0.04999988, 0.95000012,
0.20296843, 0.79703157, 0.24460834, 0.75539166, 0.30131863,
0.69868137, 0.64828754, 0.35171246, 0.89272412, 0.10727588,
0.19999975, 0.80000025, 0.36601239, 0.63398761, 0.50953936,

```

0.49046064, 0.58844400, 0.41155600, 0.73363523, 0.26636477,
0.88542717, 0.11457283, 0.49999954, 0.50000046, 0.73736084,
0.26263916, 0.79128323, 0.20871677, 0.82795922, 0.17204078,
0.87377052, 0.12622948, 0.92093372, 0.07906628, 0.74999875,
0.25000125, 0.92922541, 0.07077459, 0.93680316, 0.06319684,
0.94323496, 0.05676504, 0.94519934, 0.05480066, 0.94737705,
0.05262295, 0.94999977, 0.05000023, 0.49999545, 0.50000455,
0.49999523, 0.50000477, 0.50001445, 0.49998555, 0.50000574,
0.49999426, 0.49999450, 0.50000550, 0.50000000, 0.50000000,
0.00355242, 0.99644758, 0.10376618, 0.89623382, 0.40256823,
0.59743177, 0.79624682, 0.20375318, 0.98514132, 0.01485868,
0.00500173, 0.99499827, 0.09299359, 0.90700641, 0.13974059,
0.86025941, 0.22126391, 0.77873609, 0.55219457, 0.44780543,
0.95580529, 0.04419471, 0.09999967, 0.90000033, 0.21013927,
0.78986073, 0.34889955, 0.65110045, 0.46826059, 0.53173941,
0.83308042, 0.16691958, 0.94265709, 0.05734291, 0.19999947,
0.80000053, 0.59425293, 0.40574707, 0.74623867, 0.25376133,
0.90296107, 0.09703893, 0.96810896, 0.03189104, 0.98920019,
0.01079981, 0.60000239, 0.39999761, 0.81450009, 0.18549991,
0.84020855, 0.15979145, 0.89718504, 0.10281496, 0.88822116,
0.11177884, 0.88562722, 0.11437278, 0.85000159, 0.14999841,
0.29999827, 0.70000173, 0.30000635, 0.69999365, 0.50000824,
0.49999176, 0.69999933, 0.30000067, 0.90003937, 0.09996063,
0.30000000, 0.70000000, 0.00014145, 0.99985855, 0.01402143,
0.98597857, 0.20581080, 0.79418920, 0.58631814, 0.41368186,
0.85005224, 0.14994776, 0.20000270, 0.79999730, 0.01618371,
0.98381629, 0.08935140, 0.91064860, 0.17459392, 0.82540608,
0.46325530, 0.53674470, 0.85210344, 0.14789656, 0.04999932,
0.95000068, 0.09420466, 0.90579534, 0.29310670, 0.70689330,
0.27793647, 0.72206353, 0.61403710, 0.38596290, 0.85299456,
0.14700544, 0.24999929, 0.75000071, 0.46466247, 0.53533753,
0.56710324, 0.43289676, 0.67521543, 0.32478457, 0.76854729,
0.23145271, 0.94469376, 0.05530624, 0.50000517, 0.49999483,
0.89140569, 0.10859431, 0.93843105, 0.06156895, 0.99805948,
0.00194052, 0.99947176, 0.00052824, 0.99771604, 0.00228396,
0.95000234, 0.04999766, 0.19999737, 0.80000263, 0.20001577,
0.79998423, 0.35001338, 0.64998662, 0.64998994, 0.35001006,
0.85005604, 0.14994396, 0.20000000, 0.80000000, 0.23999998,
0.76000002, 0.30000002, 0.69999998, 0.40000012, 0.59999988,
0.65000005, 0.34999995, 0.95000012, 0.04999988, 0.40000000,
0.60000000, 0.30000005, 0.69999995, 0.39999994, 0.60000006,
0.49999972, 0.50000028, 0.64999981, 0.35000019, 0.94999994,
0.05000006, 0.50000000, 0.50000000, 0.60000026, 0.39999974,
0.60000020, 0.39999980, 0.59999956, 0.40000044, 0.69999963,
0.30000037, 0.94999963, 0.05000037, 0.60000000, 0.40000000,

```

    0.80000001, 0.19999999, 0.80000006, 0.19999994, 0.79999990,
    0.20000010, 0.79999978, 0.20000022, 0.94999962, 0.05000038,
    0.80000000, 0.20000000, 0.94999984, 0.05000016, 0.94999990,
    0.05000010, 0.95000004, 0.04999996, 0.95000003, 0.04999997,
    0.95000004, 0.04999996, 0.95000000, 0.05000000, 0.70000000,
    0.30000000, 0.70000000, 0.30000000, 0.70000000, 0.30000000,
    0.70000000, 0.30000000, 0.70000000, 0.30000000, 0.70000000,
    0.30000000);
};
};

node ShotSuccess
{
  PARENTS = (GoalDist, GoalHit);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.81785615, 0.18214385, 0.19083162, 0.80916838,
    0.59999876, 0.40000124, 0.86551337, 0.13448663, 0.12335149,
    0.87664851, 0.37000890, 0.62999110, 0.56069265, 0.43930735,
    0.08171471, 0.91828529, 0.10004607, 0.89995393, 0.23565930,
    0.76434070, 0.12839560, 0.87160440, 0.01009167, 0.98990833,
    0.59731848, 0.40268152, 0.27630616, 0.72369384, 0.15011271,
    0.84988729, 0.49999753, 0.50000247, 0.50000134, 0.49999866,
    0.50000000, 0.50000000);
  };
};

node Shoot
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);
  DEFINITION =
  {
    NAMESTATES = (ShootWithAccel, ShootSlow, Dribble);
    PROBABILITIES = (0.32395698, 0.04954856, 0.62649446, 0.51877109,
    0.07589894, 0.40532998, 0.81377356, 0.02090572, 0.16532072,
    0.58783615, 0.03558241, 0.37658145, 0.73601693, 0.02969043,
    0.23429263, 0.34697065, 0.04474968, 0.60827967, 0.79508290,
    0.18591212, 0.01900498, 0.60445883, 0.04961802, 0.34592314,
    0.01119569, 0.39945107, 0.58935325, 0.13389670, 0.14667819,
    0.71942512, 0.06027697, 0.92660019, 0.01312284, 0.04107800,
    0.83163314, 0.12728886, 0.25887801, 0.05939222, 0.68172977,
    0.38351182, 0.03836368, 0.57812451, 0.37326286, 0.53732385,
    0.08941330, 0.21178297, 0.40627511, 0.38194194);
  };
};

```

```
};
```

```
};
```

B.2.3 Netzwerk mit Bewertungsknoten, für statischen Torwart

Es sind nur die Unterschiede zum Netzwerk mit einem Entscheidungsknoten aufgeführt.

```
net DribbleOrShoot
{
  node WayFree
  {
    PARENTS = (ObstXDist2, ObstYDist2);
    DEFINITION =
    {
      NAMESTATES = (CompletelyFree, FreeIfAccel, FreeNotAccel, Blocked);
      PROBABILITIES = (0.00143472, 0.00129957, 0.00061264, 0.99665307,
        0.00997255, 0.00995874, 0.24797682, 0.73209190, 0.02260706,
        0.31466924, 0.06186320, 0.60086052, 0.39966298, 0.07834436,
        0.13228375, 0.38970896, 0.17740777, 0.01587861, 0.28214477,
        0.52456882, 0.04997387, 0.19997307, 0.20001753, 0.55003555,
        0.06217470, 0.08572048, 0.01065173, 0.84145309, 0.01602453,
        0.01640227, 0.00359520, 0.96397800, 0.01258825, 0.01235738,
        0.00305638, 0.97199797, 0.07638267, 0.11110121, 0.16288767,
        0.64962846, 0.00265557, 0.00180853, 0.37482909, 0.62070679,
        0.19996494, 0.39996398, 0.15002066, 0.25005039, 0.07835695,
        0.08338523, 0.01160526, 0.82665257, 0.38079439, 0.56949510,
        0.01229435, 0.03741616, 0.27722354, 0.66603358, 0.02208269,
        0.03466022, 0.46368236, 0.38065601, 0.01168448, 0.14397718,
        0.11807672, 0.03746450, 0.26828608, 0.57617273, 0.24999865,
        0.24999970, 0.24998657, 0.25001508, 0.00522841, 0.00318703,
        0.00454044, 0.98704412, 0.40387259, 0.48962532, 0.04324544,
        0.06325663, 0.34679383, 0.50096497, 0.06358488, 0.08865629,
        0.60842866, 0.32683127, 0.02743046, 0.03730963, 0.03130748,
        0.03005867, 0.26231421, 0.67631968, 0.60003271, 0.35003551,
        0.00095751, 0.04897426, 0.00243368, 0.00192241, 0.03881298,
        0.95683092, 0.58397188, 0.39609941, 0.01003368, 0.00989502,
        0.47839405, 0.50297603, 0.00405096, 0.01457893, 0.59013645,
        0.39410998, 0.00955422, 0.00619934, 0.00498967, 0.00468612,
        0.39137135, 0.59895285, 0.40001719, 0.30001819, 0.00998353,
        0.28998110, 0.04991055, 0.09990603, 0.20001471, 0.65016871,
        0.09987077, 0.29986086, 0.29994042, 0.30032793, 0.24981888,
```

```

    0.59980033, 0.04988256, 0.10049821, 0.49979666, 0.39978570,
    0.00105758, 0.09936009, 0.79940881, 0.15939439, 0.00147048,
    0.03972635, 0.29999997, 0.04999997, 0.25000002, 0.40000003);
};
};

node CanDribble
{
  PARENTS = (MySpeed, ObstXDist, ObstYDist);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.05333135, 0.94666865, 0.21581729, 0.78418271,
    0.53973860, 0.46026140, 0.88434834, 0.11565166, 0.99999398,
    0.00000602, 0.05001664, 0.94998336, 0.33310011, 0.66689989,
    0.48526488, 0.51473512, 0.68640619, 0.31359381, 0.99999220,
    0.00000780, 0.99997619, 0.00002381, 0.30002515, 0.69997485,
    0.51326808, 0.48673192, 0.56333426, 0.43666574, 0.74835411,
    0.25164589, 0.98318871, 0.01681129, 0.99989100, 0.00010900,
    0.50000964, 0.49999036, 0.66900035, 0.33099965, 0.50729533,
    0.49270467, 0.73927434, 0.26072566, 0.79545843, 0.20454157,
    0.74225431, 0.25774569, 0.89998926, 0.10001074, 0.97726817,
    0.02273183, 0.85862081, 0.14137919, 0.90389575, 0.09610425,
    0.93508953, 0.06491047, 0.93478717, 0.06521283, 0.99899757,
    0.00100243, 0.60000141, 0.39999859, 0.60001218, 0.39998782,
    0.60003773, 0.39996227, 0.60008562, 0.39991438, 0.60040156,
    0.39959844, 0.60000000, 0.40000000, 0.05888679, 0.94111321,
    0.13284848, 0.86715152, 0.29177254, 0.70822746, 0.75610003,
    0.24389997, 0.99999968, 0.00000032, 0.05001148, 0.94998852,
    0.25113435, 0.74886565, 0.43187008, 0.56812992, 0.67375949,
    0.32624051, 0.91522262, 0.08477738, 0.99997942, 0.00002058,
    0.20001662, 0.79998338, 0.41904512, 0.58095488, 0.86220471,
    0.13779529, 0.99983325, 0.00016675, 0.99998078, 0.00001922,
    0.99995235, 0.00004765, 0.50001624, 0.49998376, 0.95238780,
    0.04761220, 0.99996571, 0.00003429, 0.99994754, 0.00005246,
    0.99997206, 0.00002794, 0.99999956, 0.00000044, 0.75001902,
    0.24998098, 0.99990838, 0.00009162, 0.99992425, 0.00007575,
    0.99994223, 0.00005777, 0.99994159, 0.00005841, 0.99999973,
    0.00000027, 0.95001146, 0.04998854, 0.50000606, 0.49999394,
    0.50003181, 0.49996819, 0.50009424, 0.49990576, 0.50011208,
    0.49988792, 0.50027948, 0.49972052, 0.50000000, 0.50000000,
    0.02492651, 0.97507349, 0.20305823, 0.79694177, 0.62994624,
    0.37005376, 0.94073120, 0.05926880, 0.99918667, 0.00081333,
    0.00500995, 0.99499005, 0.19175861, 0.80824139, 0.41128860,
    0.58871140, 0.84976109, 0.15023891, 0.92384823, 0.07615177,

```

```
0.99997206, 0.00002794, 0.10001741, 0.89998259, 0.52384626,  
0.47615374, 0.58289677, 0.41710323, 0.99877579, 0.00122421,  
0.99968680, 0.00031320, 0.99991239, 0.00008761, 0.20002353,  
0.79997647, 0.93361709, 0.06638291, 0.70499371, 0.29500629,  
0.99634806, 0.00365194, 0.99856338, 0.00143662, 0.99988211,  
0.00011789, 0.60002818, 0.39997182, 0.97195046, 0.02804954,  
0.75339817, 0.24660183, 0.80668169, 0.19331831, 0.99934949,  
0.00065051, 0.99977478, 0.00022522, 0.85001265, 0.14998735,  
0.30002646, 0.69997354, 0.30010162, 0.69989838, 0.50022568,  
0.49977432, 0.70014529, 0.29985471, 0.90019683, 0.09980317,  
0.30000000, 0.70000000, 0.02519677, 0.97480323, 0.11115312,  
0.88884688, 0.45987635, 0.54012365, 0.77114340, 0.22885660,  
0.99789783, 0.00210217, 0.20001418, 0.79998582, 0.11490650,  
0.88509350, 0.21318947, 0.78681053, 0.62696510, 0.37303490,  
0.80789614, 0.19210386, 0.99978987, 0.00021013, 0.05001972,  
0.94998028, 0.43175317, 0.56824683, 0.00108004, 0.99891996,  
0.04278051, 0.95721949, 0.79435193, 0.20564807, 0.99995936,  
0.00004064, 0.25001364, 0.74998636, 0.70413181, 0.29586819,  
0.00177297, 0.99822703, 0.00453196, 0.99546804, 0.55312402,  
0.44687598, 0.99987650, 0.00012350, 0.50000546, 0.49999454,  
0.99266699, 0.00733301, 0.28455223, 0.71544777, 0.00231862,  
0.99768138, 0.77494685, 0.22505315, 0.99924512, 0.00075488,  
0.95000122, 0.04999878, 0.20003444, 0.79996556, 0.20010281,  
0.79989719, 0.35024240, 0.64975760, 0.65017114, 0.34982886,  
0.85029510, 0.14970490, 0.20000000, 0.80000000, 0.24000012,  
0.75999988, 0.30000063, 0.69999937, 0.40000138, 0.59999862,  
0.65000111, 0.34999889, 0.95000254, 0.04999746, 0.40000000,  
0.60000000, 0.30000078, 0.69999922, 0.40000241, 0.59999759,  
0.50000460, 0.49999540, 0.65000297, 0.34999703, 0.95000375,  
0.04999625, 0.50000000, 0.50000000, 0.60000195, 0.39999805,  
0.60000402, 0.39999598, 0.60000804, 0.39999196, 0.70000514,  
0.29999486, 0.95000336, 0.04999664, 0.60000000, 0.40000000,  
0.80000230, 0.19999770, 0.80000343, 0.19999657, 0.80000829,  
0.19999171, 0.80000550, 0.19999450, 0.95000321, 0.04999679,  
0.80000000, 0.20000000, 0.95000088, 0.04999912, 0.95000100,  
0.04999900, 0.95000274, 0.04999726, 0.95000213, 0.04999787,  
0.95000213, 0.04999787, 0.95000000, 0.05000000, 0.70000000,  
0.30000000, 0.70000000, 0.30000000, 0.70000000, 0.30000000,  
0.70000000, 0.30000000, 0.70000000, 0.30000000, 0.70000000,  
0.30000000);  
};  
};  
  
node ShotSuccess  
{
```

```

PARENTS = (GoalDist, GoalHit);
DEFINITION =
{
  NAMESTATES = (Yes, No);
  PROBABILITIES = (0.66722491, 0.33277509, 0.00116724, 0.99883276,
0.59999959, 0.40000041, 0.64360914, 0.35639086, 0.00024584,
0.99975416, 0.37001003, 0.62998997, 0.51483243, 0.48516757,
0.00014539, 0.99985461, 0.10012002, 0.89987998, 0.25385368,
0.74614632, 0.31151362, 0.68848638, 0.01024568, 0.98975432,
0.60327816, 0.39672184, 0.55335130, 0.44664870, 0.15030153,
0.84969847, 0.49999806, 0.50000194, 0.49993832, 0.50006168,
0.50000000, 0.50000000);
};
};

node ShootFast
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.99808403, 0.00191597, 0.00107866, 0.99892134,
0.99817451, 0.00182549, 0.99876887, 0.00123113, 0.99795970,
0.00204030, 0.00064097, 0.99935903, 0.99744771, 0.00255229,
0.99869160, 0.00130840, 0.00006950, 0.99993050, 0.00015213,
0.99984787, 0.67912721, 0.32087279, 0.00006991, 0.99993009,
0.00031376, 0.99968624, 0.00093172, 0.99906828, 0.00512074,
0.99487926, 0.00166204, 0.99833796);
  };
};

node ShootSlow
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.23379356, 0.76620644, 0.00007856, 0.99992144,
0.16196515, 0.83803485, 0.03674885, 0.96325115, 0.32217815,
0.67782185, 0.00003681, 0.99996319, 0.19114893, 0.80885107,
0.22285669, 0.77714331, 0.00000124, 0.99999876, 0.00001161,
0.99998839, 0.99947942, 0.00052058, 0.22857953, 0.77142047,
0.00000933, 0.99999067, 0.00004514, 0.99995486, 0.00000143,
0.99999857, 0.00000461, 0.99999539);
  };
};

```

```

};

node Dribble
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);
  DEFINITION =
  {
    NAMESTATES = (State0, State1);
    PROBABILITIES = (0.00193595, 0.99806405, 0.99983869, 0.00016131,
    0.00004124, 0.99995876, 0.00040782, 0.99959218, 0.00127014,
    0.99872986, 0.99954036, 0.00045964, 0.00199336, 0.99800664,
    0.00228393, 0.99771607, 0.99995795, 0.00004205, 0.99996386,
    0.00003614, 0.00014193, 0.99985807, 0.90630231, 0.09369769,
    0.99978194, 0.00021806, 0.99954578, 0.00045422, 0.99885301,
    0.00114699, 0.99996319, 0.00003681);
  };
};

node SFRating
{
  PARENTS = (ShootFast);
  DEFINITION =
  {
    NAMESTATES = (VeryGood, QuiteGood, Invariant, QuiteBad, VeryBad);
    PROBABILITIES = (0.40000000, 0.35000000, 0.19000000, 0.05000000,
    0.01000000, 0.01000000, 0.05000000, 0.19000000, 0.35000000,
    0.40000000);
  };
};

node SSIRating
{
  PARENTS = (ShootSlow);
  DEFINITION =
  {
    NAMESTATES = (VeryGood, QuiteGood, Invariant, QuiteBad, VeryBad);
    PROBABILITIES = (0.40000000, 0.35000000, 0.19000000, 0.05000000,
    0.01000000, 0.01000000, 0.05000000, 0.19000000, 0.35000000,
    0.40000000);
  };
};

node DRating
{
  PARENTS = (Dribble);

```

```

DEFINITION =
{
  NAMESTATES = (VeryGood, QuiteGood, Invariant, QuiteBad, VeryBad);
  PROBABILITIES = (0.40000000, 0.35000000, 0.19000000, 0.05000000,
0.01000000, 0.01000000, 0.05000000, 0.19000000, 0.35000000,
0.40000000);
};
};
};

```

B.2.4 Netzwerk mit Bewertungsknoten, für sich regelförmig bewegenden Torwart

Es sind nur die Unterschiede zum Netzwerk für den statischen Torwart aufgeführt.

```

net DribbleOrShoot
{
  node WayFree
  {
    PARENTS = (ObstXDist2, ObstYDist2);
    DEFINITION =
    {
      NAMESTATES = (CompletelyFree, FreeIfAccel, FreeNotAccel, Blocked);
      PROBABILITIES = (0.00101527, 0.00103451, 0.0186702, 0.97928, 0.000675448,
0.000540601, 0.471516, 0.527269, 0.0793435, 0.485821,
0.277561, 0.157275, 0.443659, 0.13867, 0.218419,
0.199252, 0.00668113, 0.00678507, 0.391973, 0.59456,
0.0499532, 0.199952, 0.200042, 0.550051, 0.213074,
0.304277, 0.0472258, 0.435423, 0.220303, 0.425588,
0.175507, 0.178604, 0.0280529, 0.0996162, 0.577676,
0.294654, 0.255684, 0.342129, 0.260868, 0.141318,
0.0493722, 0.0497894, 0.384898, 0.515941, 0.199964,
0.399965, 0.150033, 0.250038, 0.447264, 0.549259,
0.000621019, 0.00285587, 0.400187, 0.597731, 0.00125003,
0.000832345, 0.296855, 0.702512, 0.000264005, 0.000370048,
0.191041, 0.130725, 0.427036, 0.251199, 0.484072,
0.0566087, 0.179267, 0.280054, 0.250006, 0.250008,
0.24999, 0.249993, 0.456074, 0.543316, 0.000597082,
1.22851e-05, 0.524168, 0.471358, 0.00216891, 0.00230583,
0.527343, 0.46691, 0.00189329, 0.00385396, 0.547799,
0.204433, 0.106545, 0.141223, 0.13651, 0.000943088,
0.350854, 0.511695, 0.600017, 0.350018, 0.000976874,
0.0489882, 0.575289, 0.422072, 0.00115478, 0.00148296,
0.625678, 0.371887, 0.000932455, 0.00150235, 0.547289,

```

```

0.449254, 0.00086796, 0.00258878, 0.535377, 0.300311,
0.0565401, 0.107771, 0.197608, 0.000430512, 0.340608,
0.461354, 0.400003, 0.300001, 0.00999419, 0.290001,
0.0499599, 0.0999695, 0.200037, 0.650033, 0.0997878,
0.299808, 0.300215, 0.30019, 0.249838, 0.599847,
0.050164, 0.100151, 0.499797, 0.399801, 0.00118518,
0.0992158, 0.798885, 0.158859, 0.00201681, 0.0402397,
0.3, 0.05, 0.25, 0.4);
};
};

```

```

node CanDribble
{
  PARENTS = (MySpeed, ObstXDist, ObstYDist);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.0480176, 0.951982, 0.218602, 0.781398, 0.570307,
0.429693, 0.839465, 0.160535, 0.997667, 0.00233338,
0.0500023, 0.95, 0.297215, 0.702785, 0.407804,
0.592196, 0.595867, 0.404133, 0.863381, 0.136619,
0.887221, 0.11278, 0.299997, 0.700003, 0.484924,
0.515076, 0.546396, 0.453604, 0.518018, 0.481982,
0.681228, 0.318772, 0.700307, 0.299693, 0.499988,
0.500012, 0.665095, 0.334905, 0.779419, 0.220581,
0.672021, 0.327979, 0.565046, 0.434954, 0.530231,
0.469769, 0.899984, 0.100016, 0.982211, 0.0177902,
0.992673, 0.00732688, 0.914961, 0.0850388, 0.873676,
0.126325, 0.876902, 0.123098, 0.998995, 0.00100456,
0.599999, 0.400001, 0.60002, 0.39998, 0.600072,
0.399928, 0.60004, 0.39996, 0.600045, 0.399955,
0.6, 0.4, 0.0399208, 0.960078, 0.130491,
0.869509, 0.399111, 0.600889, 0.775805, 0.224195,
0.931258, 0.0687429, 0.0500027, 0.949998, 0.213816,
0.786184, 0.311947, 0.688053, 0.556997, 0.443003,
0.822187, 0.177813, 0.965898, 0.0341018, 0.200005,
0.799995, 0.363909, 0.636091, 0.622067, 0.377933,
0.888236, 0.111764, 0.91047, 0.0895303, 0.966057,
0.0339426, 0.500005, 0.499995, 0.625841, 0.374159,
0.853157, 0.146843, 0.998999, 0.00100005, 0.999892,
0.000108415, 0.936421, 0.0635794, 0.75, 0.25,
0.875665, 0.124336, 0.945794, 0.0542057, 0.999458,
0.000541397, 0.999852, 0.000148069, 0.997785, 0.00221517,
0.950005, 0.0499966, 0.49999, 0.50001, 0.500033,
0.499967, 0.500211, 0.499789, 0.500132, 0.499868,

```

0.500037, 0.499963, 0.5, 0.5, 0.000218901,
0.999781, 0.162516, 0.837484, 0.617997, 0.382003,
0.997603, 0.00239644, 0.998757, 0.00124336, 0.0050387,
0.994961, 0.0217185, 0.978283, 0.287417, 0.712583,
0.591621, 0.408379, 0.815445, 0.184555, 0.986075,
0.0139256, 0.100035, 0.899965, 0.00500337, 0.994997,
0.650029, 0.349971, 0.985143, 0.0148566, 0.990612,
0.00938796, 0.979306, 0.0206935, 0.20002, 0.79998,
0.379548, 0.620452, 0.930741, 0.0692589, 0.99847,
0.00152959, 0.999655, 0.000345263, 0.999622, 0.000378282,
0.600026, 0.399974, 0.645943, 0.354057, 0.765294,
0.234706, 0.98137, 0.0186304, 0.999893, 0.000107515,
0.999809, 0.000191414, 0.850015, 0.149985, 0.299979,
0.700021, 0.300065, 0.699935, 0.500245, 0.499755,
0.700226, 0.299774, 0.900915, 0.0990848, 0.3,
0.7, 0.000271341, 0.999729, 0.05153, 0.94847,
0.37179, 0.62821, 0.784311, 0.215689, 0.998006,
0.00199528, 0.200051, 0.799949, 0.00833243, 0.991667,
0.201226, 0.798774, 0.488357, 0.511643, 0.699509,
0.300491, 0.989067, 0.0109328, 0.0500419, 0.949958,
0.000764183, 0.999236, 0.581685, 0.418315, 0.97235,
0.0276502, 0.982563, 0.0174368, 0.993078, 0.00692263,
0.250031, 0.749969, 0.0748913, 0.925109, 0.829137,
0.170863, 0.995515, 0.00448431, 0.998844, 0.00115549,
0.999233, 0.000766369, 0.500049, 0.499951, 0.610632,
0.389368, 0.812811, 0.187189, 0.956988, 0.0430115,
0.994, 0.00600032, 0.999507, 0.000492489, 0.950018,
0.0499817, 0.199961, 0.800039, 0.200049, 0.799951,
0.350205, 0.649795, 0.650205, 0.349795, 0.851215,
0.148785, 0.2, 0.8, 0.24, 0.76,
0.3, 0.7, 0.4, 0.6, 0.65,
0.35, 0.950005, 0.0499964, 0.4, 0.6,
0.3, 0.7, 0.4, 0.6, 0.500002,
0.499998, 0.65, 0.35, 0.950005, 0.049997,
0.5, 0.5, 0.6, 0.4, 0.6,
0.4, 0.600005, 0.399995, 0.700001, 0.299999,
0.95, 0.05, 0.6, 0.4, 0.8,
0.2, 0.8, 0.2, 0.800005, 0.199995,
0.8, 0.2, 0.95, 0.0500005, 0.8,
0.2, 0.95, 0.0500008, 0.95, 0.0500002,
0.95, 0.0499992, 0.95, 0.0499992, 0.95,
0.0499985, 0.95, 0.05, 0.7, 0.3,
0.7, 0.3, 0.7, 0.3, 0.7,
0.3, 0.7, 0.3, 0.7, 0.3);
};

```

};

node ShotSuccess
{
  PARENTS = (GoalDist, GoalHit);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.995762, 0.00423786, 0.00931228, 0.990688, 0.600024,
0.399976, 0.996585, 0.00341475, 0.00142232, 0.998578,
0.370029, 0.629971, 0.095101, 0.904899, 0.0346371,
0.965363, 0.100191, 0.899809, 0.100625, 0.899375,
0.352005, 0.647995, 0.0103966, 0.989603, 0.588408,
0.411592, 0.813148, 0.186852, 0.15049, 0.84951,
0.500025, 0.499975, 0.499935, 0.500065, 0.5,
0.5);
  };
};

node ShootFast
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.998297, 0.00170241, 0.00478135, 0.995218, 0.9999,
9.99786e-05, 0.998345, 0.00165524, 0.998074, 0.00192582,
0.000736127, 0.999263, 0.999674, 0.000326751, 0.991063,
0.00893604, 0.00297172, 0.997028, 0.000226343, 0.999773,
0.235271, 0.764729, 0.00530636, 0.994694, 0.00142868,
0.998572, 0.000360441, 0.999639, 0.477361, 0.522639,
0.00523702, 0.994763);
  };
};

node ShootSlow
{
  PARENTS = (WayFree, CanDribble, ShotSuccess);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.366619, 0.633381, 0.00769898, 0.992301, 0.259569,
0.740431, 0.422496, 0.577504, 0.473603, 0.526397,
0.0620598, 0.93794, 0.326588, 0.673412, 0.702954,
0.297046, 0.0418839, 0.958116, 0.0211911, 0.978809,

```

```

0.999704, 0.000296268, 0.929619, 0.0703808, 0.0320925,
0.967908, 0.00321799, 0.996782, 0.981987, 0.0180128,
0.991204, 0.00879628);
    };
};

node Dribble
{
    PARENTS = (WayFree, CanDribble, ShotSuccess);
    DEFINITION =
    {
        NAMESTATES = (State0, State1);
        PROBABILITIES = (0.00223472, 0.997764, 0.997763, 0.00223653, 0.000495043,
0.999504, 0.000143111, 0.999857, 0.00177884, 0.998222,
0.997437, 0.00256384, 0.000463436, 0.999537, 8.13768e-05,
0.999919, 0.999986, 1.46889e-05, 0.999685, 0.00031452,
0.000760803, 0.999238, 0.691144, 0.308856, 0.999849,
0.000151355, 0.999785, 0.000214431, 0.111506, 0.888494,
0.609069, 0.390931);
    };
};

};

```

B.3 Passspiel

B.3.1 Netzwerk für den Nutzen eines Passes zu einer Position

```

net UsePassPosition
{
    node WayFreeToPos
    {
        PARENTS = ();
        DEFINITION =
        {
            NAMESTATES = (Free, Blocked, Risky);
            PROBABILITIES = (0.50000000, 0.20000000, 0.30000000);
        };
    };

    node DistanceToPos
    {
        PARENTS = ();
    };
};

```

```

DEFINITION =
{
  NAMESTATES = (VerySmall, Small, Medium, High);
  PROBABILITIES = (0.10000000, 0.20000000, 0.39900000, 0.30100000);
};
};

```

```

node CanUsePos
{
  PARENTS = (WayFreeToPos, DistanceToPos);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.10000000, 0.90000000, 0.35000000, 0.65000000,
0.90000000, 0.10000000, 0.75000000, 0.25000000, 0.05000000,
0.95000000, 0.10000000, 0.90000000, 0.15000000, 0.85000000,
0.10000000, 0.90000000, 0.07000000, 0.93000000, 0.30000000,
0.70000000, 0.70000000, 0.30000000, 0.40000000, 0.60000000);
  };
};

```

```

node S_WayFreeToPos
{
  PARENTS = (WayFreeToPos);
  DEFINITION =
  {
    NAMESTATES = (Free, Blocked, Risky, Unknown, NoValue);
    PROBABILITIES = (0.55900000, 0.05000000, 0.14000000, 0.25000000,
0.00100000, 0.02000000, 0.57900000, 0.15000000, 0.25000000,
0.00100000, 0.20000000, 0.20000000, 0.34900000, 0.25000000,
0.00100000);
  };
};

```

```

node S_DistanceToPos
{
  PARENTS = (DistanceToPos);
  DEFINITION =
  {
    NAMESTATES = (VerySmall, Small, Medium, High, NoValue);
    PROBABILITIES = (0.92600000, 0.07000000, 0.00100000, 0.00100000,
0.00200000, 0.07000000, 0.85700000, 0.07000000, 0.00100000,
0.00200000, 0.00100000, 0.07000000, 0.85700000, 0.07000000,
0.00200000, 0.00100000, 0.00100000, 0.07000000, 0.92600000,
0.00200000);
  };
};

```

```

    };
};

node PartnerDistToPos1
{
    PARENTS = ();
    DEFINITION =
    {
        NAMESTATES = (VerySmall, Small, Medium, High);
        PROBABILITIES = (0.10000000, 0.20000000, 0.40000000, 0.30000000);
    };
};

node S_PartnerDist1
{
    PARENTS = (PartnerDistToPos1);
    DEFINITION =
    {
        NAMESTATES = (VerySmall, Small, Medium, High, NoValue);
        PROBABILITIES = (0.92700000, 0.07000000, 0.00100000, 0.00100000,
        0.00100000, 0.07000000, 0.85800000, 0.07000000, 0.00100000,
        0.00100000, 0.00100000, 0.07000000, 0.85800000, 0.07000000,
        0.00100000, 0.00100000, 0.00100000, 0.07000000, 0.82800000,
        0.10000000);
    };
};

node PartnerDistToPos2
{
    PARENTS = ();
    DEFINITION =
    {
        NAMESTATES = (VerySmall, Small, Medium, High);
        PROBABILITIES = (0.10000000, 0.20000000, 0.40000000, 0.30000000);
    };
};

node MinPartnerDist
{
    PARENTS = (PartnerDistToPos1, PartnerDistToPos2);
    DEFINITION =
    {
        NAMESTATES = (VerySmall, Small, Medium, High);
        PROBABILITIES = (0.99700000, 0.00100000, 0.00100000, 0.00100000,
        0.99700000, 0.00100000, 0.00100000, 0.00100000, 0.99700000,

```

```

    0.00100000, 0.00100000, 0.00100000, 0.99700000, 0.00100000,
    0.00100000, 0.00100000, 0.99700000, 0.00100000, 0.00100000,
    0.00100000, 0.00100000, 0.99700000, 0.00100000, 0.00100000,
    0.00100000, 0.99700000, 0.00100000, 0.00100000, 0.00100000,
    0.99700000, 0.00100000, 0.00100000, 0.99700000, 0.00100000,
    0.00100000, 0.00100000, 0.00100000, 0.99700000, 0.00100000,
    0.00100000, 0.00100000, 0.00100000, 0.99700000, 0.00100000,
    0.00100000, 0.00100000, 0.99700000, 0.00100000, 0.99700000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.99700000,
    0.00100000, 0.00100000, 0.00100000, 0.00100000, 0.99700000);
};
};

node ShallUsePos
{
  PARENTS = (CanUsePos, MinPartnerDist);
  DEFINITION =
  {
    NAMESTATES = (Yes, No);
    PROBABILITIES = (0.96000000, 0.04000000, 0.70000000, 0.30000000,
    0.40000000, 0.60000000, 0.05000000, 0.95000000, 0.10000000,
    0.90000000, 0.07000000, 0.93000000, 0.04000000, 0.96000000,
    0.01000000, 0.99000000);
  };
};

node S_PartnerDist2
{
  PARENTS = (PartnerDistToPos2);
  DEFINITION =
  {
    NAMESTATES = (VerySmall, Small, Medium, High, NoValue);
    PROBABILITIES = (0.92700000, 0.07000000, 0.00100000, 0.00100000,
    0.00100000, 0.07000000, 0.85800000, 0.07000000, 0.00100000,
    0.00100000, 0.00100000, 0.07000000, 0.85800000, 0.07000000,
    0.00100000, 0.00100000, 0.00100000, 0.07000000, 0.82800000,
    0.10000000);
  };
};
};
};
```

B.3.2 Netzwerk für den Nutzen des Dribblings

```

net CanDribble
{
  node DribbleWayFree
  {
    PARENTS = ();
    DEFINITION =
    {
      NAMESTATES = (Free, Blocked, Risky);
      PROBABILITIES = (0.40000000, 0.25000000, 0.35000000);
    };
  };

  node S_WayFree
  {
    PARENTS = (DribbleWayFree);
    DEFINITION =
    {
      NAMESTATES = (Free, Blocked, Risky, Unknown, NoValue);
      PROBABILITIES = (0.70900000, 0.05000000, 0.14000000, 0.10000000,
        0.00100000, 0.02000000, 0.52900000, 0.15000000, 0.30000000,
        0.00100000, 0.20000000, 0.20000000, 0.34900000, 0.25000000,
        0.00100000);
    };
  };

  node ShallDribble
  {
    PARENTS = (DribbleWayFree);
    DEFINITION =
    {
      NAMESTATES = (Dribble, Pass);
      PROBABILITIES = (0.98000000, 0.02000000, 0.10000000, 0.90000000,
        0.50000000, 0.50000000);
    };
  };
};

```


Glossar

Bayessches Netz Siehe Bayessches Netzwerk.

Bayessches Netzwerk Eine Repräsentation einer Wahrscheinlichkeitsfunktion in Form eines Graphen, wobei zu jeder Variable ein Knoten existiert und die Kanten die Abhängigkeiten zwischen den Variablen modellieren. Es kann zu verschiedenen Arten von Inferenz benutzt werden. Englisch: *bayesian network* oder *belief network*.

Cliquenbaum Hier: Repräsentation eines Bayesschen Netzwerks in Form eines Baumes aus Cliquen eines aus dem Netzwerk bestimmten Graphen. Englisch: *junction tree* oder *join tree*.

Conditional Probability Table Tabelle, in der zu jeder Kombination der Zustände der Eltern eines Knotens in einem Bayesschen Netz die Wahrscheinlichkeiten abgelegt sind, mit der er einen seiner Zustände annimmt.

d-Separation Eigenschaft zweier Knotenmengen in einem Bayesschen Netzwerk. Entspricht der Unabhängigkeit der korrespondierenden Variablen.

Dynamisches Bayessches Netzwerk Spezielles Bayessches Netzwerk, das besonders für zeitbehaftete Modellierungen geeignet ist.

Evidenz Sicherheit über den Zustand einer Wahrscheinlichkeitsvariable: die Wahrscheinlichkeit, dass sie diesen einen bestimmten Zustand annimmt, ist eins; die Wahrscheinlichkeit aller anderen Zustände null.

Inferenz Probabilistisches Schließen durch Berechnung der Werte von Wahrscheinlichkeitsvariablen, wobei Evidenz in Betracht gezogen werden kann. Je nach Richtung des Schlusses wird sie als kausal, interkausal oder diagnostisch bezeichnet.

Polybaum Graph, in dem keine (ungerichteten) Zyklen auftreten.

Verstärkungslernen Automatisiertes Lernen, bei dem ein Agent anhand von Bewertungen (Belohnungen / Bestrafungen) trainiert wird. Beim aktiven Verstärkungslernen probiert der Agent selbst verschiedene Verhaltensweisen aus. Englisch: *reinforcement learning*.

Abbildungsverzeichnis

1.1	Roboter des Teams CoPS Stuttgart	2
1.2	Softwarearchitektur im Team CoPS	3
2.1	Beispiel eines Bayesschen Netzwerkes	8
2.2	Beispiele zur d-Separation	9
2.3	Beispiel für eine Normalisierung	11
2.4	Beispiel eines Entscheidungsnetzwerkes	14
2.5	Generelle Struktur eines DBN (nach [RN95])	14
3.1	Basisarchitektur der Bibliotheken	24
3.2	Struktur der „BV“ - Bibliothek	25
3.3	Adapter zwischen Weltmodell und Bayesschen Netzen	26
3.4	Ausschnitt aus einem Bayesschen Netz in Polybaumstruktur	28
3.5	Erzeugung eines Cliquenbaumes	31
3.6	Framework zum Trainieren von Bayesschen Netzen	42
4.1	Beispiel eines Sensorknotens	45
4.2	Erzeugung von Baumstrukturen	47
4.3	Beispiel zu Entscheidungen: Ursprüngliches Netz	48
4.4	Beispiel zu Entscheidungen: Reduziertes Netz	49
4.5	Beispiel für Bewertungsknoten	50
4.6	Netz zur Ermittlung des Ballbesitzes	51
4.7	Anteil der von verschiedenen Netzen korrekt berechneten Beispiele	54
4.8	Trainingsergebnisse beim letzten Schritt	55
4.9	Ursprüngliches Netzwerk zur Schussentscheidung	56
4.10	Netzwerk zur Schussentscheidung	57
4.11	Variante des Netzwerks zur Schussentscheidung	59

4.12	Trainingsergebnisse mit statischem Torwart	61
4.13	Trainingsergebnisse mit sich regelförmig bewegendem Torwart	62
4.14	Beispielsituationen zur Passentscheidung	64
4.15	Netze zur Entscheidung über Pass oder Dribbling	65
A.1	Neue Architektur des Spielers	78

Tabellenverzeichnis

4.1	CPT des Knotens „Habe ich den Ball?“	52
4.2	CPT „Torerfolg?“: Training	62
4.3	CPT „Schuss?“: Training	63
B.1	Legende zu den Klassendiagrammen	130

Symbolverzeichnis

Allgemeiner Hinweis

Diskrete Variablen werden mit kleinen lateinischen Buchstaben (z.B. x) bezeichnet, kontinuierliche Variablen dagegen mit kleinen griechischen Buchstaben (z.B. ξ). Große lateinische Buchstaben (z.B. F) stehen für Knoten oder Mengen (mit der Ausnahme von P und M).

Skalare Größen, Vektoren und Matrizen werden nicht durch ihre Bezeichnung von einander unterschieden, die jeweilige Definition der Variablen gibt darüber Aufschluss. Zur Unterscheidungshilfe sind Vektoren in Fettschrift abgedruckt.

Kontinuierliche Variablen

α, β Hilfsgrößen zur Normalisierung

Diskrete Variablen

x, y, z, u Zustände von Wahrscheinlichkeitsknoten
 i, j, k, l Zählervariablen
 n Anzahl Knoten eines Netzes
 e Anzahl Kanten eines Netzes
 c Anzahl Variablen in den CPTs eines Netzes

Funktionen

P Wahrscheinlichkeit
 ψ CPT eines Knoten oder einer Clique
 M Nachricht von einer Clique zu einer anderen Clique
 λ, π Nachrichten zur Wurzel hin bzw. von der Wurzel weg

Mengen

E, D	Evidenzen
C	Clique
S	Separator zwischen zwei Cliques
R	Differenz zwischen Separator und Clique
U	Elternknoten eines Knotens

Elemente aus Mengen

X, Y, Z	Knoten in einem Netz
w	Eintrag in einer CPT (entspricht einem Funktionswert einer CPT)

Vektoren aus Skalaren

\mathbf{z}, \mathbf{u}	Zustandskombinationen von Knoten
--------------------------	----------------------------------

Legende zu den Klassendiagrammen

Die in dieser Arbeit verwendeten Klassendiagramme benutzen die bekannte Symbolik der UML (siehe z.B. [Obj]). Dennoch gibt es in der Praxis häufig Unterschiede in deren Verwendung. Insbesondere die Symbole, die die Sichtbarkeit einzelner Klassen oder Member anzeigen, sind nicht klar standardisiert. Daher folgt hier in Tabelle B.1 eine Legende, in der die verwendeten Symbole erläutert sind.












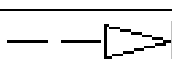

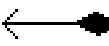
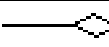

Symbol (Beispiel)	Bedeutung
 BVChanceNode	Klasse
 Network	Abstrakte Klasse
 NetworkListener	Schnittstellen-Klasse
 CPT	Paketinterne Klasse
 <code>getName()</code>	<code>public</code> Methode
 <code>calcEvidenceStates()</code>	<code>protected</code> Methode
 <code>setProbability()</code>	Paketinterne Methode
 <code>hasCycle()</code>	<code>private</code> Methode
 <code>stateAdded()</code>	Abstrakte Methode
 <code>getInstance()</code>	Statische Methode
	Vererbung
	Schnittstellenimplementierung
	Assoziation
	Komposition
	Aggregation
	Allg. Benutzung / Abhängigkeit

Tabelle B.1: Legende zu den Klassendiagrammen

Literaturverzeichnis

- [aib] Aibo robot. <http://www.aibo.com/>.
- [BKLL03] T. Buchheim, G. Kindermann, R. Lafrenz, und P. Levi. A Dynamic Environment Modelling Framework for Selective Attention. In *IJCAI*, 2003.
- [BKRK97] John Binder, Daphne Koller, Stuart J. Russell, und Keiji Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2-3):213–244, 1997. <http://citeseer.nj.nec.com/article/-binder97adaptive.html>.
- [bns] Software packages for bayesian networks. <http://www.ai.mit.edu/~%7emurphyk/Bayes/bnsoft.html>.
- [Bun94] W. Buntine. Operations for Learning with Graphical Models. *Journal for artificial intelligence*, 1994.
- [Bun96] W. Buntine. A guide to the literature on learning probabilistic networks from data. *IEEE Trans. On Knowledge And Data Engineering*, 8:195–210, 1996. <http://citeseer.nj.nec.com/buntine96guide.html>.
- [CDS96] R.G. Cowell, A.P. Dawid, und P. Sebastiani. A comparison of sequential learning methods for incomplete data. In *Bayesian Statistics 5*. Oxford University Press, 1996.
- [CGH97] Enrique Castillo, José Manuel Gutiérrez, und Ali S. Hadi. *Expert Systems and Probabilistic Network Models*. Springer, New York, 1997.
- [Coo90] G.F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42, 1990.
- [DLR77] A.P. Dempster, N.M. Laird, und D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39, 1977.
- [FBT99] Dieter Fox, Wolfram Burgard, und Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999. <http://citeseer.nj.nec.com/-fox99markov.html>.

- [FHKR95] Jeff Forbes, Timothy Huang, Keiji Kanazawa, und Stuart J. Russell. The BATmobile: Towards a Bayesian Automated Taxi. In *IJCAI*, pages 1878–1885, 1995. <http://citeseer.nj.nec.com/forbes95batmobile.html>.
- [For02] Jeffrey Forbes. *Reinforcement Learning for Autonomous Vehicles*. Dissertation, University of California, Berkeley, 2002. <http://www.cs.duke.edu/~%7eforbes/thesis/thesis.pdf>.
- [GHJV96] Erich Gamma, Richard Helm, Ralph Johnson, und John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, München, 1996.
- [GMS00] Michael Goossens, Frank Mittelbach, und Alexander Samarin. *Der L^AT_EX Begleiter*. Addison-Wesley, 2000.
- [Ham01] Jürgen Hammelmann. Darstellung der Machbarkeit einer Prädiktion des Verhaltens von Verkehrsteilnehmern durch Kombination von Zeitnetzen mit probabilistischen Methoden. Diplomarbeit, Institut für Parallele und Verteilte Höchstleistungsrechner, Universität Stuttgart, Stuttgart, Germany, Juli 2001.
- [HD96] C. Huang und A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.
- [Hec95] David Heckerman. A tutorial on learning with bayesian networks, 1995. <http://citeseer.nj.nec.com/heckerman96tutorial.html>.
- [hug] Hugin expert systems. <http://www.hugin.com/>.
- [Jen01] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2001.
- [jvb] Javabayes. <http://www-2.cs.cmu.edu/~7ejavabayes/Home/>.
- [Kra98] P. Krause. Learning probabilistic networks, 1998. <http://citeseer.nj.nec.com/krause98learning.html>.
- [LS88] Steffen Lauritzen und David Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50(2), 1988.
- [Nea90] Richard E. Neapolitan. *Probabilistic Reasoning in Expert Systems*. John Wiley & Sons, Inc., 1990.
- [NMR] Daniel Nikovski, Dimitris Margaritis, und Roseli Romero. A Tour-Giving Learning Robot that Follows People. http://www-2.cs.cmu.edu/~7erll/overview/danieln_02/.

- [NN99] D. Nikovski und I. Nourbakhsh. Learning discrete Bayesian models for autonomous agent navigation. In *Proceedings of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'99.*, pages 137–143. IEEE, Monterey, CA, 1999. <http://citeseer.nj.nec.com/nikovski99learning.html>.
- [ntc] Norsys software corporation. <http://www.norsys.com/>.
- [Obj] Object Management Group. OMG Unified Modeling Language Specification. <http://www.omg.org/technology/documents/formal/uml.htm>.
- [Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [Pri92] W.H. Price. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 1992.
- [pul] Pulcinella. <http://iridia.ulb.ac.be/pulcinella/Welcome.html>.
- [rbc] Die RoboCup Initiative. <http://www.robocup.org/>.
- [RBKK95] Stuart J. Russell, John Binder, Daphne Koller, und Keiji Kanazawa. Local learning in probabilistic networks with hidden variables. In *IJCAI*, pages 1146–1152, 1995. <http://citeseer.nj.nec.com/russell95local.html>.
- [RN95] Stuart J. Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall International, London, 1995.
- [Sah] Mehran Sahami. An autonomous mobile robot architecture using belief networks and neural networks. <http://citeseer.nj.nec.com/178359.html>.
- [smi] Genie und smile. <http://www2.sis.pitt.edu/%7egenie/>.
- [Sto98] Peter Stone. *Layered Learning in Multi-Agent Systems*. Dissertation, Carnegie Mellon University, Pittsburgh, 1998. <http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-187.pdf>.
- [Str00] Bjarne Stroustrup. *Die C++ Programmiersprache*. Addison-Wesley, München, 2000.
- [You98] Håkan L. Younes. Current tools for assisting intelligent agents in real-time decision making. Diplomarbeit, Department of Computer and Systems Sciences, Royal Institute of Technology and Stockholm University, Stockholm, Sweden, December 1998. <http://citeseer.nj.nec.com/younes98current.html>.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Jörg Rüdener)