Grundlagen der künstlichen Intelligenz

Vorlesungsskript

Sebastian Krings, et al.

This book and the corresponding slides are based on work of the students of our artifical intelligence course. We would like to thank all contributors.
The contributors are (in alphabetical order):

- Ahmadi, Sirat
- Batiukova, Olga
- Brecklinghaus, Anne
- Brenneis, Markus
- Brümmer, Ronny
- Disterhöft, Alexander
- Dreyer, Kevin
- Ebbinghaus, Björn
- Funke, Andreas
- Happe, Chistopher
- Kerkmann, Anna Maria
- Komander, Dominique
- Krzysztala, Denis
- Ludolf, Christoph
- Matussek, Tim
- Michels, Daniel
- Moreira Hamasaki, Thaís
- Nalecz, Rafael
- Oberländer, Nils
- Özen, Eyyüp
- Papenberg, Martin
- Petrasch, Jessica
- Sansalone, Sebastian
- Schmidt, Joshua
- Schuhmacher, Luisa
- Spohr, Philipp
- Seliger, Kerstin
- Sura, Sebastian
- Thelen, Alexander
- Ullrich, Julian
- Vukovic, Renato
- Wagner, Lukas
- Weck, Sandy
- Weiß, Fabian
- Weißenfeld, Anke Leonie

Inhaltsverzeichnis

VII Appendix

I Einleitung

1. Einleitung

Das Vorlesungsskript „Grundlagen der künstlichen Intelligenz“ entstand aus Beiträgen der Studierenden der gleichnamigen Veranstaltung an der Heinrich-Heine-Universität Düsseldorf. Im Rahmen eines Seminars sind die unterschiedlichen Kapitel erstellt und später durch die Dozenten zu dem vorliegenden Buch kombiniert worden. In den folgenden Semestern ist das Buch durch andere Studierende erweitert und verbessert worden. Wir hoffen, dass auch die kommenden Semester sich an dem Buch beteiligen und so zu dem Erfolg der Veranstaltung beitragen.

Vielen Dank für eure Mühe!

– Sebastian Krings und die anderen Dozenten

2. Allgemeine Definitionen

2.0.1 Graphen

Definition Ungerichteter Graph. Ein Graph $G = (V, E)$ besteht aus einer endlichen Menge von Knoten $V = u_1, ..., u_n$ und einer endlichen Menge E von Kanten. In einem ungerichteten Graphen ist jede Kante e ein Menge u, v von zwei verschiedenen Knoten $u, v \in V$.

Definition Weg. Ein Weg p in einem ungerichteten Graphen $G = (V, E)$ ist eine Folge von k Knoten $(v_1, ..., v_k)$ aus der Knotenmenge V des Graphen G, für die gilt, dass die Kanten v_i, v_{i+1} in der Kantenmenge E enthalten sind, für i aus $1, ..., k-1$.

Definition Weglänge. Die Weglänge Wf(p) eines Weges mit k Knoten $p = (v_1, ..., v_k)$ in einem Graphen $G = (V, E)$ mit der Kantenbewertung $f : E \to R$ die jeder Kante e aus der Kantenmenge E des Graphen G einen reellen Kostenwert zuordnet, ist die Summe der Kostenwerte der k Kanten des Weges p.

Definition Kürzester Weg. Die kürzeste Weglänge $k(m, n)$ von m nach n ist der Weg p, so dass es keinen Weg p' von m nach n gibt für den gilt, dass:

$$Wf(p') < Wf(p).$$

Wenn n von m aus nicht erreichbar ist, dann ist $k(m, n) = \infty$.

II Reasoning

3. Regelbasierte Expertensysteme

3.1 Einleitung

Ein Expertensystem *(kurz „XPS“)* ist ein Computerprogramm, welche das Wissen spezialisierter Fachleute nachbildet. Dadurch, dass sie das Wissen über einen spezifischen Sachbereich haben, können sie Experten bei ihren täglichen Aufgaben unterstützen und sind zudem in der Lage ihr Wissen jederzeit zur Verfügung zu stellen.

„Expertensysteme sind Programme, mit denen das Spezialwissen und die Schlußfolgerungsfähigkeit qualifizierter Fachleute auf eng begrenzten Aufgabengebieten nachgebildet werden soll.“ [**puppe**]

Es gibt unterschiedliche Arten von Expertensysteme, jedoch werden wir hier nur über die sogenannten regelbasierten Expertensysteme *(engl. „Rule-based expert systems“)* sprechen. Regelbasierte Expertensysteme schlussfolgern aus einer gegebenen Problembeschreibung eine passende Lösung.
Bei der Erstellung solch eines Systems sind drei wichtige Grundaufgaben zu beachten, die es lösen muss:

1. Bereitstellen von Wissen
2. Umgang mit Unsicherheit
3. Verhalten erklären

3.2 Bereitstellen von Wissen

Um Wissen bereitstellen zu können, müssen wir erst einmal definieren, wie Wissen dargestellt wird. Außerdem müssen wir Wissen abspeichern können, deshalb werden wir uns mit dem Aufbau eines regelbasierten Expertensystems befassen und die Funktionsweise seiner wichtigsten Komponenten erläutern.

3.2.1 Wie wird Wissen dargestellt?

Das Programm besteht aus lauter **WENN**-**DANN**-Regeln.

*„**WENN** Straße nass und **WENN** Himmel bewölkt, **DANN** hat es geregnet.“*

Die Regeln werden vom Experten bzw. von mehreren Experten eingegeben. Da es jedoch einigen Experten schwer fällt ihr komplexes Fachwissen in obiger Form umzuformulieren, wird oft auch ein sogenannter Knowledge Engineer hinzugezogen, der das Wissen der Experten dementsprechend formalisiert und in das System einfügt. Dieser regelbasierter Programmierstil, oft auch logischer Programmierstil genannt, steht im Kontrast zum traditionellen instruktionsbasiertem Programmierstil. Beim instruktionsbasiertem Programmierstil besteht das Programm aus

einer Abfolge von Instruktionen und Anfragen. Der Programmierer entscheidet hier was abgelaufen wird und in welcher Reihenfolge es abgelaufen wird. Beim regelbasiertem Programmierstil besteht das Programm aus einer Menge von Regeln. Hier entscheidet der Experte was getan werden soll und ein Regelinterpreter legt die Reihenfolge fest.

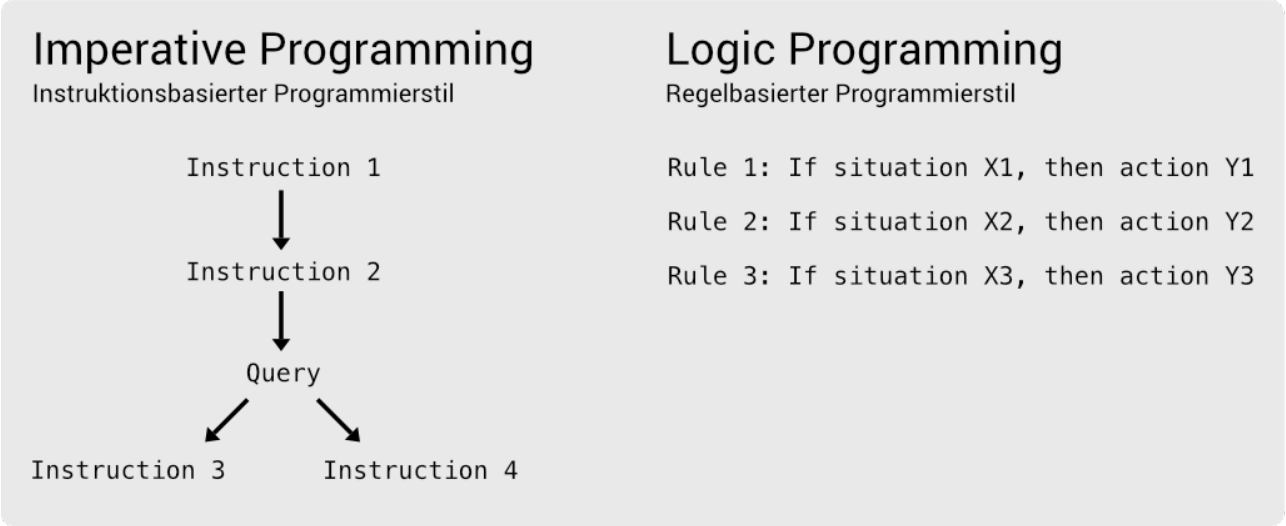


Abbildung 3.1: Imperativer Programmierstil vs. Regelbasierter Programmierstil

So ein regelbasierter Ansatz hat vier wichtige Eigenschaften, die den einzelnen Komponenten des Systems zu Gute kommen. Eine Aneinanderreihung von WENN-DANN-Regeln ist

- **Modular:** Das Wissen wird unabhängig voneinander in kleineren Regeln dargestellt.
- **Incrementable:** Aufgrund ihrer Modularität, können neue Regeln problemlos hinzugefügt werden.
- **Modifiable:** Das Ändern einer Regel beeinflusst nicht die restlichen Regeln.
- **Transparent:** Regeln können verwendet werden, um das Programmverhalten zu erklären.

3.2.2 Aufbau eines regelbasierten Expertensystems

Regelbasierte Expertensysteme bestehen aus vier verschiedenen Komponenten:

Knowledge Base:

Die Wissensbasis enthält das ganze Fachwissen der Experten.
Die ganzen WENN-DANN-Regeln sind hier gespeichert.

Inference Engine:

Der Regelinterpreter bzw. der Logikkern des ganzen Systems. Mit Hilfe der Regeln aus der Knowledge Base, wird hier das Problem interpretiert, verarbeitet und gelöst.

Working Storage:

Hier werden relevante Daten zur aktuellen Problemlösung zwischen gespeichert.

User Interface:

Die Schnittstelle zwischen dem Benutzer und der Inference Engine.

Der Working Storage, die Benutzerschnittstelle sowie die Inference Engine bilden die Hülle des Systems. Die Knowledge Base soll theoretisch austauschbar sein, jedoch ist dies in der Praxis nur sehr schwer umsetzbar, denn die Inference Engine ist zu sehr auf das Wissen der Knowledge Base zugeschnitten.

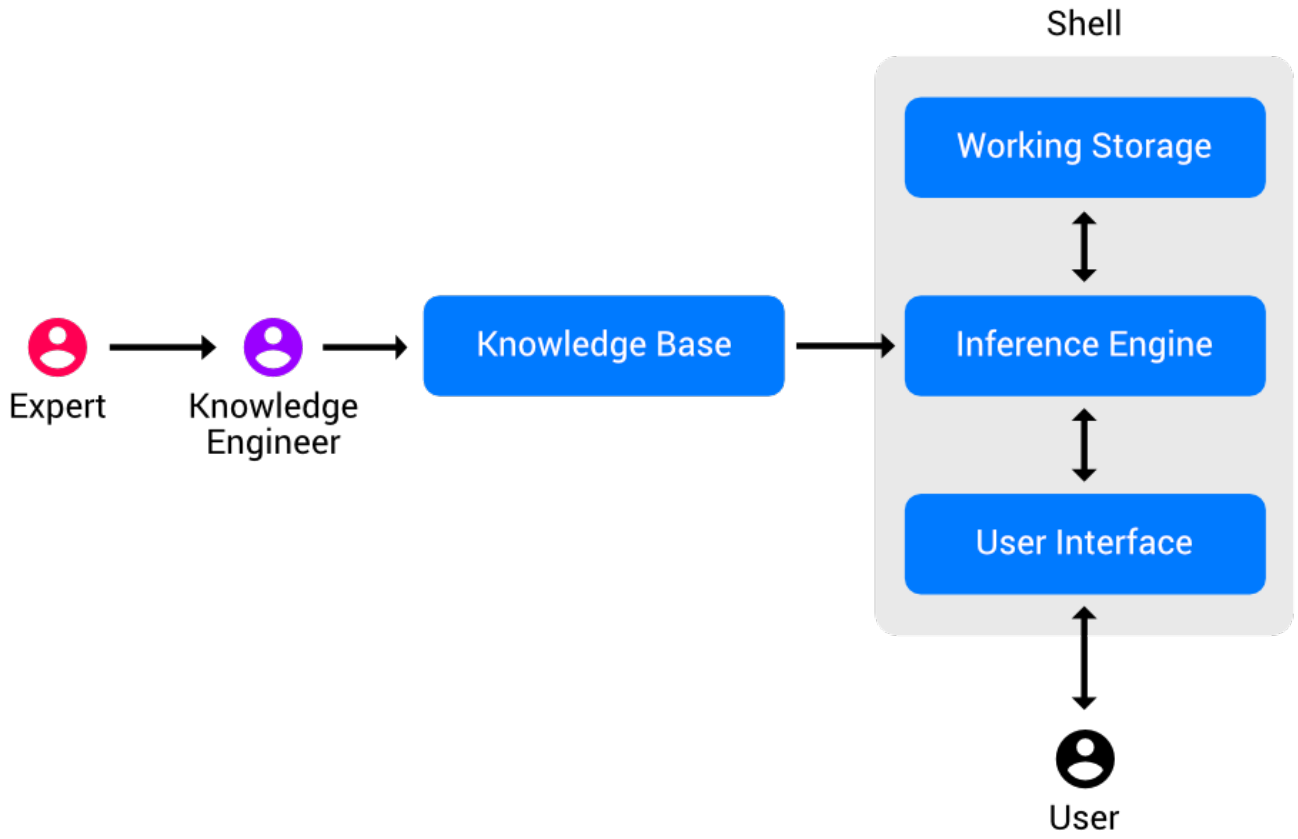


Abbildung 3.2: Aufbau eines regelbasierten Expertensystems

3.2.3 Wie wird Wissen abgearbeitet?

Die Inferenz der Regeln erfolgt intern über Goal Trees, auch AND-OR-Trees genannt. Ein Goal Tree weist die üblichen Baumeigenschaften auf. Das Topgoal ist die Wurzel des Baumes, dementsprechend sind die Subgoals die Teilbäume des Baumes, welche selbst wiederum Goal Trees sind. Es gibt zwei Arten von Knoten:

AND-Knoten: Hier müssen alle Teilbäume erfüllt werden.
OR-Knoten: Ein Teilbaum muss erfüllt werden.

Es gibt generell zwei verschiedene Ansätze um die Problemstellung zu lösen, entweder anhand des **Forward Chainings** oder seinem Gegenstück, anhand des **Backward Chainings**.

3.3 Forward Chaining

Das Forward Chaining wird auch **Data Driven Reasoning** genannt, denn man fängt mit einer gegebenen Faktenbasis *(den Basiszuständen)* an. Dies sind die Blätter des Goal Trees. Von hier aus schließt man mit Hilfe der Regeln aus der Knowledge Base neue Fakten und tastet sich immer weiter vor bis man schlussendlich aus den geschlossenen Fakten eine Diagnose tätigen kann. Dies ist unser Topgoal, also die Wurzel des Goal Trees.

Zum Veranschaulichen ein kleines Beispiel aus der Tierwelt:
In der freien Natur sehen wir ein Tier, was wir identifizieren wollen. Um nun unser Problem zu lösen benutzen wir ein Expertensystem, welches mit der Methode des Forward Chaining arbeitet. Wie oben bereits schon erwähnt fangen wir beim Forward Chaining mit einer Faktenbasis an. Auf unser Problem bezogen bedeutet dies, dass wir Beobachtungen aufstellen müssen. Zum Beispiel sehen wir, dass einige Tiere Federn haben, andere wiederum einen Schnabel. Einige leben in der Kälte, andere sind nachtaktiv. Mit Hilfe dieser Basiszustände versucht das System eine Lösung zu inferieren.

Abbildung 3.3: Beginnen mit unserer Faktenbasis, hier links

Mit den Regeln aus der Knowledge Base kann das System nun schrittweise zu Teillösungen voranschreiten. Die erste Regel könnte lauten: **WENN** Federn und **WENN** Schnabel, **DANN** Vogel.

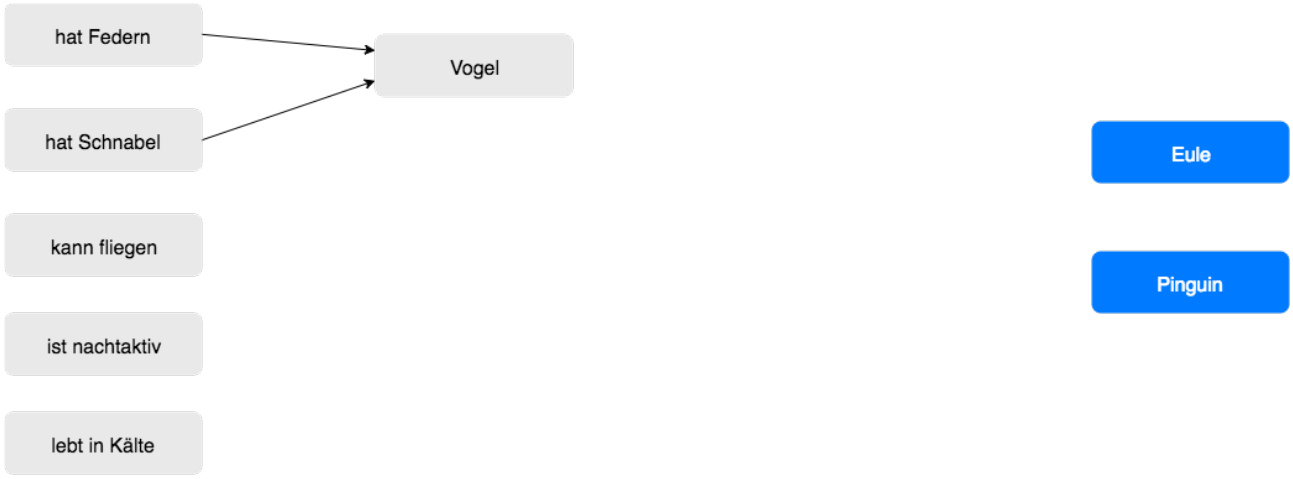


Abbildung 3.4: Neue Teillösungen inferieren

Eine andere Regel könnte lauten: **WENN** Vogel und **WENN** fliegen, **DANN** flugfähiger Vogel.

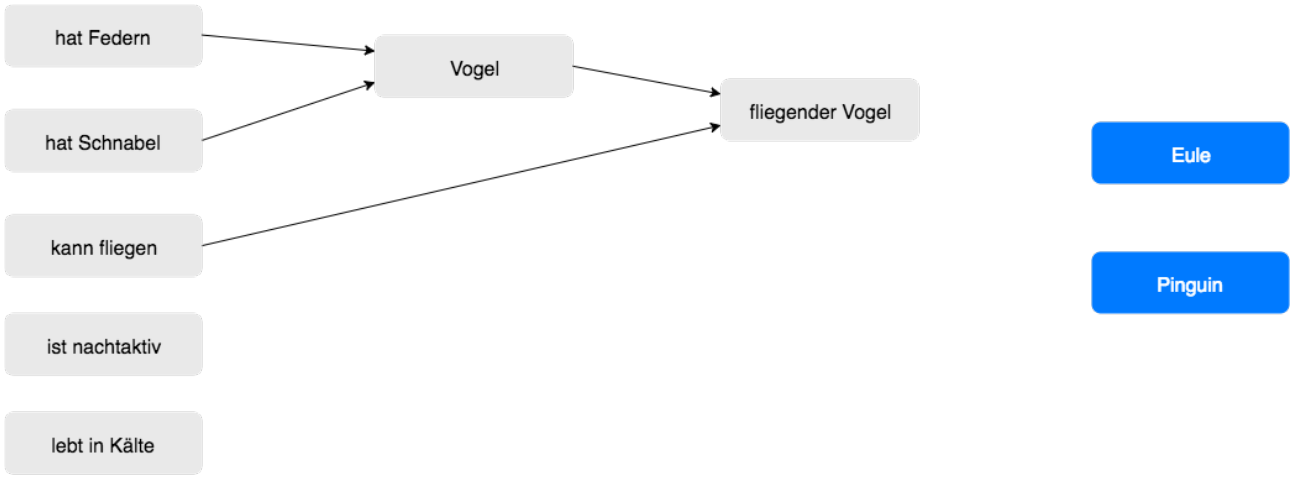


Abbildung 3.5: Weitere Teillösung inferieren

Eine weitere Regel könnte lauten: **WENN** fliegender Vogel und **WENN** nachtaktiv, **DANN** Eule. Somit hat unser Expertensystem von unseren anfänglichen Beobachtungen geschlussfolgert, dass es sich um eine Eule handeln muss.

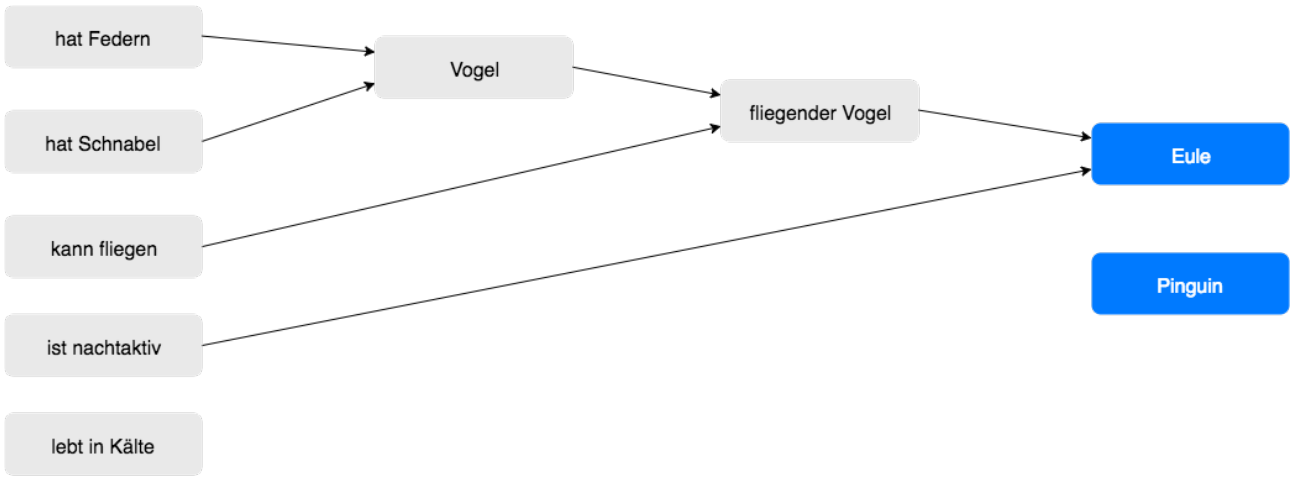


Abbildung 3.6: Topgoal erreicht

Würden wir nun ein weiteres Tier sehen und wieder unser Expertensystem fragen, könnte das ganze so aussehen:

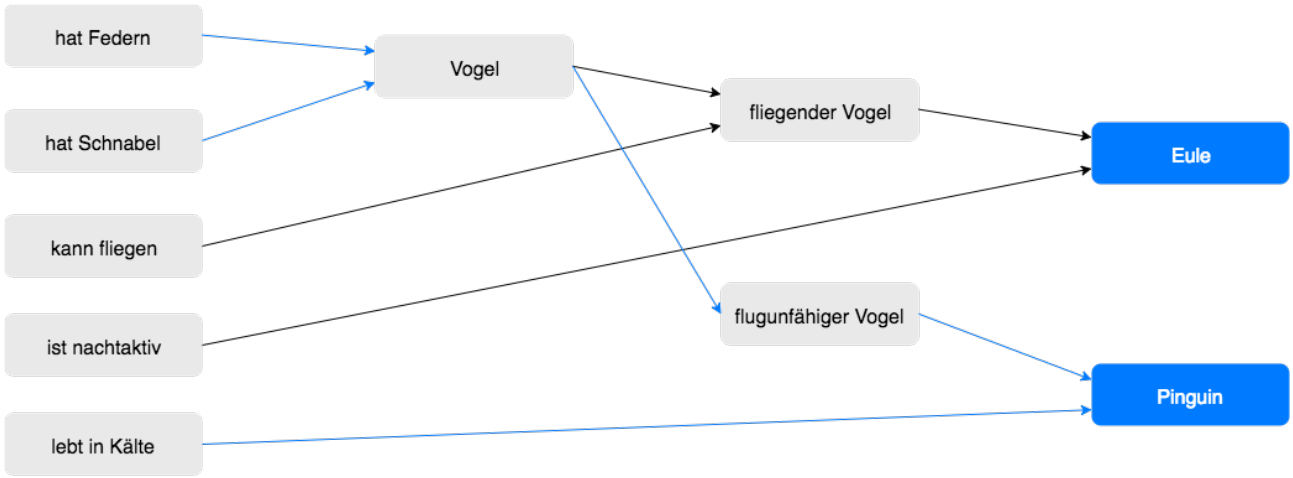


Abbildung 3.7: Von unserer Faktenbasis inferiert das System, dass es sich um einen Pinguion handelt.

Das Forward Chaining ist anwendbar auf Probleme mit nicht aufzählbaren Lösungen. In der Realität wird Forward Chaining daher in Design- und Konfigurationssysteme genutzt.

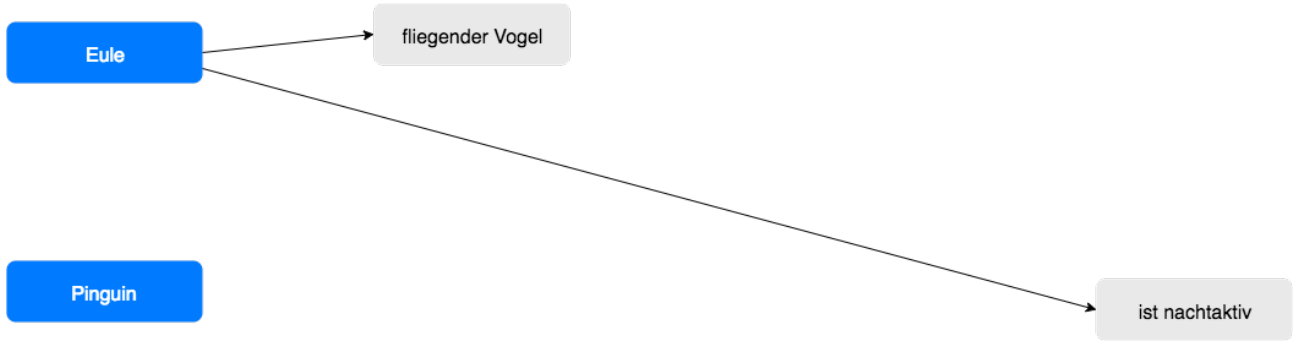
3.4 Backward Chaining

Backward Chaining ist wie vorher schon erwähnt das Gegenstück zum Forward Chaining. Hier beginnt man beim Topgoal und zerlegt dieses in die ursprünglichen Subgoals. Da die Subgoals prinzipiell wieder Topgoals sind zerlegt man diese auch wieder in Subgoal. Dies macht man solange bis man wieder bei den Blättern, also den Fakten angelangt ist. Man kann dazu auch sagen: Das System inferiert von den Hauptlösung zur Teillösung. Diese Teillösungen werden dabei im Working Storage gespeichert. Nun wird dies wieder an unserem bekannten Beispiel mit Eule und Pinguin illustriert.

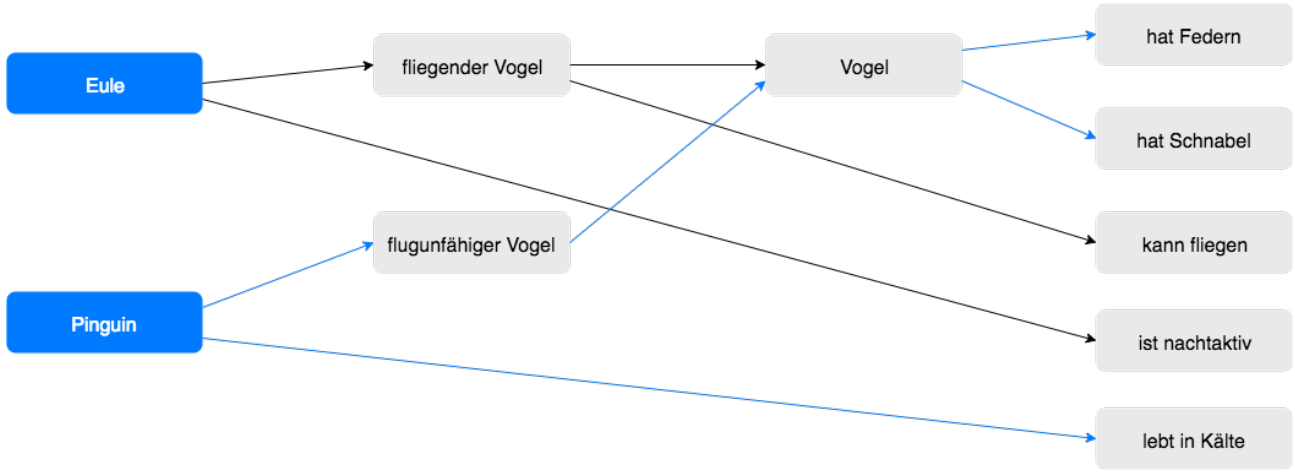
Beispiel

Wir haben nun die beiden Topgoals Eule und Pinguin gegeben. Nun möchten wir wieder zu den Teillösungen gelangen. Dies geschieht wie auch beim Forward Chaining mit den Wenn-Dann Regeln. In diesem Beispiel wäre die erste Regel:

Wenn Eule, dann kann fliegen und ist nachtaktiv



Dies macht man nun solange bis man wieder bei der Faktenbasis ist.



Diese Art von regelbasierten Expertensystem ist anwendbar auf Probleme mit strukturierter Selektion und auf Probleme, die aufzählbare Lösungen haben. Hier könnte sich die Frage stellen: Warum nur auf Probleme mit aufzählbaren Lösungen? Diese Frage beantwortet sich fast schon von allein, da jede Lösung mindestens eine Regel haben muss.
In der Realität wird Backward Chaining daher in Identifaktions- und Diagnosesysteme zum Einsatz.

3.5 Umgang mit Unsicherheit

Im regelbasierten Expertensystem gibt es nur absolute Angaben, also nur richtig oder falsch. Dies ist aber nicht Realitätsnah, da Benutzer nicht immer absolut sicher sind. Außerdem kann es auch sehr gut sein, dass Unsicherheiten

in den Regeln vorkommen. Durch all diese Unsicherheiten gibt es das Bedürfnis mit diesen Unsicherheiten umzugehen.

Eine Möglichkeit mit den Unsicherheiten sind die Certainty Factors (CF) zu deutsch Sicherheitsfaktoren.

3.5.1 Certainty Factors

Die CF's liegen zwischen -1 und 1. Dabei entspricht -1 falsch, 1 wahr und 0 kompletter Unwissenheit. Den CF des Topgoals kann man berechnen, in dem man den minimalen CF der Subgoals mit dem CF der abschließenden Regel multipliziert. Dies wird im nächsten Beispiel veranschaulicht. Desweiteren gilt:

- ist ein Subgoal mit -1 bewertet, so ist das Topgoal nicht erfüllbar
- erhält das Topgoal einen CF mit 1, so ist dieses mit Sicherheit erfüllbar.

In manchen Systemen kann es vorkommen, dass die Werte -1 und 1 als sehr mächtig interpretiert werden und man noch andere Werte zu lassen möchte. Daher kann man sich eigene Schwellen einrichten, so könnte man Beispielsweise sagen: „Wir erkennen schon einen Wert ab 0.75 als wahr an und einen -0.75 als falsch." Aber auch dies ist immer System und Benutzer abhängig.

Beispiel

Betrachten wir nun die Regel:

Wenn flugunfähiger Vogel und lebt in Kälte dann Pinguin.

- **flugunfähiger Vogel** ist ein Subgoal welches durch alle Regeln und Subgoals, die zu ihm führen mit einem CF=0.4 bewertet wurde.
- **lebt in Kälte** ist ein Fakt, welcher einen CF=-0.3 erhalten hat. Dieser wurde vom Benutzer eingegeben.
- Die **Wenn-Dann-Regel** ist eine sehr sichere Regel die mit 0.9 bewertet wurde. Man könnte sich noch vorstellen, dass der flugunfähige Vogel in der Kälte eine Möwe mit gebrochenen Flügel ist, aber das ist auch schon alles.

Der minimale Wert der Subgoals ist hier -0.3.

$\Rightarrow$ CF für Pinguin $-0.3 \cdot 0.9 = -0.27$

Die -0.27 sagen uns nun das wir eher nicht wissen, dass es ein Pinguin ist, aber wir schon sehr unsicher sind ob es wirklich kein Pinguin ist.

Es kann vorkommen, dass es in einem regelbasierten Expertensystem mehrere Wege gibt, um auf das gleiche Ziel zu gelangen. Wenn dieser Fall eintritt möchte man die CF's, die durch die verschiedenen Regeln für das gleiche Topgoal entstehen in irgendeiner Form kombinieren können. Dazu gibt es verschiedene Möglichkeiten. Eine dieser Möglichkeiten ist unter anderem MYCINs Certainty Algebra.

3.5.2 MYCINs Certainty Algebra

Als Voraussetzung für diese Algebra muss man mindestens 2 CF's haben, die man kombinieren möchte, also X und Y. Hat man dies gegeben, so kann man den kombinierten CF mit Hilfe der folgenden Algebra berechnen:

$$CF(X,Y) = \begin{cases} X + Y\,(1-X) & X \geq 0 \wedge Y \geq 0 \\ (X+Y)\,/\,(1-min\,(|X|,|Y|)) & X < 0 \vee Y < 0 \\ X + Y\,(1+X) & X < 0 \wedge Y < 0 \end{cases}$$

Wenn man mehr als 2 CF's gegeben hat, dann kombiniert man 2 und das Ergebnis dann mit den weiteren z.B.: a, b, c kombiniere erst d=CF(b,c) erg=CF(a,d)

$\Rightarrow$ Die Reihenfolge spielt hier also keine Rolle.

Beispiel zur Kombination von CF's mithilfe von MYCINs Certainty Algebra

Regeln:

1. flugunfähiger Vogel, isst Fisch, lebt auch in der Antarktis $\Rightarrow$ Pinguin CF=0.51
2. flugunfähiger Vogel, ist schwarz/weiß, kann schwimmen $\Rightarrow$ Pinguin CF=-0.3
3. Vogel, isst Fisch, kann schwimmen, lebt auch in der Antarktis $\Rightarrow$ Pinguin CF=0.4

Nun haben wir den Fall, dass wir 3 Regeln haben, die man kombinieren möchte. Zuerst sei X=0.51 und Y=-0.3, also haben wir den Fall, dass einer der Werte unter 0 ist und der andere über 0. Daher müssen wir $CF\,(X,Y) = (X+Y)\,/\,(1-min\,(|X|,|Y|))$ anwenden. Also:

$$CF\,(0.51,-0.3) = (0.51-0.3)\,/\,(1-min\,(|0.51|,|-0.3|))$$

Das ergibt 0.3. Nun muss noch die 0.3 mit der 0.4 kombiniert werden. Wenn man dies macht kommt ein Wert von 0.58.

3.6 Verhalten erklären

Expertensysteme können ihr Verhalten dem Nutzer erklären, sollte dieser eine simple Nachfrage stellen. Hierbei wird zwischen zwei Arten von Fragen unterschieden.

3.6.1 Why-Questions

Why-Questions sind Fragen die angeben, warum die genau diese Anfrage dem Nutzer gestellt wurde. In einem Goal Tree muss sich das System nur eine Ebene nach oben bewegen um die Antwort auf die gestellte Frage zu finden.

Beispielaufruf:
System: Ist fliegender Vogel?
User: why?
System: Eule $\Leftarrow$ fliegender Vogel, ist nachtaktiv

3.6.2 How-Questions

How-Questions beantworten warum diese spezifische Lösung inferiert wurde. Das Expertensystem hat sich seinen Arbeitsweg abgespeichert und ruft diesen Schritt für Schritt ab, falls der Nutzer Fragen stellt.

Beispielaufruf:
System: Tier ist Eule.
User: how Eule?
System: fliegender Vogel, ist nachtaktiv $\Rightarrow$ Eule
User: how fliegender Vogel?
System: Vogel, kann fliegen $\Rightarrow$ fliegender Vogel

3.7 Zusammenfassung

Expertensysteme simulieren menschliche Experten, die nur aus WENN-DANN-Regeln bestehen. Die Lösungen, die die Expertensysteme generieren sind basiert auf Regeln, die von der Problemstellung vorgegeben werden. Es kann mit Unsicherheiten umgehen, sei es Unsicherheiten in den Regeln oder auch von der Benutzereingabe her. Das Programm kann sich selbst erklären solange die Fragen How bzw. Why basiert sind.

	Forward Chaining	**Backward Chaining**
Ansatz	Data Driven Reasoning	Goal Driven Reasoning
Goal Tree	von Blättern aufgebaut	von Wurzel aufgebaut
Lösungen finden	Lösungszustände durch Regeln akzeptiert	Jede Lösung (mind.) ein Topgoal
Anwendung	nicht aufzählbare Lösungsmenge	aufzählbare Lösungsmenge

3.8 Kritikpunkte

Es gibt einige Kritikpunkte an den Expertensystemen, da die Knowledge Base aufwändig zu erstelle ist. Meistens benötigt man einen Experten und zustätzlich noch einen Knowledge Engineer, der das Expertenwissen in eine regelbasierte Computersprache umsetzen kann. Das Wissen des Expertensystems ist kein echtes Wissen, sondern nur regelbasiert, somit kann nur neues Wissen angelegt werden, wenn dieses wieder in die Regelform gebracht wurde. Es gibt zwar die Möglichkeit das System zu hinterfragen mit den How- und Why-Questions, jedoch gibt das System meistens nur die Position im Goal Tree an, aber kein Hintergrundwissen oder Begründungen für den Benutzer.

4. Theorembeweiser

4.1 Einleitung

Der Eine kennt sie überhaupt nicht, der Andere womöglich noch aus der Schule, aber wirklich jeder Student mit Mathematikkursen kennt sie - die Beweise. In der Mathematik werden seit jeher Theorien aufgestellt, bewiesen und mitunter auch widerlegt. Man kann das Beweisen dieser als eine Art Kunstform ansehen. So gibt es häufig nicht nur den einen Weg zum Ziel, sondern oftmals auch schnellere, schönere und/oder einfachere. Allen gemein ist allerdings, dass Außenstehende damit kaum etwas anfangen können. Nichtsdestotrotz resultieren die Bemühungen der Mathematiker nicht zuletzt in einen sichtbaren Nutzen. Als ein Beispiel sei hierbei das Vier-Farben-Problem genannt. Kurz und knapp geht es um die Frage: „Ist es möglich eine Landkarte mit lediglich vier verschiedenen Farben so einzufärben, so dass am Ende alle benachbarten Länder verschiedenfarbig sind?“ Die Frage entstand bereits im 19. Jahrhundert, als das Drucken von zusätzlichen Farben noch mit erhöhtem Aufwand einherging. 1852 formulierte Francis Guthrie erstmals dazu einen Satz als Vermutung. Dieser erlangte schnell Popularität und es folgten eine Reihe von Beweisen, welche dann früher oder später widerlegt wurden. Erstmals bewiesen wurde er 1976 von Appel und Haken, welche zu jener Zeit frisch entwickelte Methoden zum computergestützten Beweisen von Theoremen anwendeten. Dazu wurde das Problem zunächst auf die Ebene der Graphentheorie getragen und anschließend mit einer Mischung aus klassischen und computerbasierten Beweisen zu einem positiven Ergebnis gebracht. Das allgemeine Problem wurde dabei auf 1936 Fälle (später 1478) reduziert, welche der Computer dann einzeln durchprüfen konnte. Es gibt vor allem zwei Hauptkritikpunkte an Appel und Hakens Beweis:

1. Es wurde ein Computer benutzt, sodass nicht alles verifiziert werden kann.
2. Gerade der Teil, der per Hand geprüft werden müsste, ist so kompliziert und mühsam, dass ihn, soweit bekannt, niemand unabhängig geprüft hat.

Man wird schnell bemerken, dass dies ein durchaus stark diskutiertes Thema ist. So argumentieren Kritiker, dass Beweise stets von Menschenhand nachprüfbar und nachvollziehbar sein sollten, oder zweifeln gar die Korrektheit der maschinellen Beweise an. Nun, in der Tat verstecken sich häufig Fehler in Computerprogrammen, sodass die verwendeten Algorithmen mitunter inkonsistent arbeiten. 1996 bewiesen N. Robertson, D. P. Sanders, P. Seymour und R. Thomas den Vier-Farben-Satz mit der gleichen Beweisidee, reduzierten allerdings die Anzahl der Problemfälle auf 633. Mittlerweile gibt es noch weitere unabhängige Beweise dazu. Man kann zwar nicht ausschließen, dass der Mensch oder die Maschine in einem geführten Beweis Fehler gemacht haben, aber je häufiger ein Theorem unabhängig bewiesen wird, desto unwahrscheinlicher wird es, dass all diese Beweise fehlerhaft sind. Anhand der folgenden Tabelle von Thomas C. Hales[Hal08] sieht man, dass es seit längerem Bestrebungen gibt, mathematische Aussagen mit dem Computer zu beweisen.

Darüber hinaus haben sich über die Jahre hinweg verschiedene Systeme und auch verschiedene Anforderungen an diese entwickelt. Im folgenden möchte ich erklären was Theorembeweiser (kurz TB) sind, wie diese arbeiten

Jahr	Theorem	Beweissystem	Formaler Beweis	Klassischer Beweis
1986	First Incompleteness	Boyer-Moore	Shankar	Gödel
1990	Quadratic Reciprocity	Boyer-Moore	Russinoff	Eisenstein
1996	Fundamental - of Calculus	HOL Light	Harrison	Henstock
2000	Fundamental - of Algebra	Mizar	Milewski	Brynski
2000	Fundamental - of Algebra	Coq	Geuvers et al.	Kneser
2004	Four-Color	Coq	Gonthier	Robertson et al.
2004	Prime Number	Isabelle	Avigad et al.	Selberg-Erdös
2005	Jordan Curve	HOL Light	Hales	Thomassen
2005	Brouwer Fixed Point	HOL Light	Harrison	Kuhn
2006	Flyspeck I	Isabelle	Bauer-Nipkow	Hales
2007	Cauchy Residue	HOL Light	Harrison	classical
2008	Prime Number	HOL Light	Harrison	analytic proof

und wofür wir sie brauchen. Dazu werden der formale Beweis, Erfüllbarkeitsprobleme (kurz SAT, engl. für „satisfiability“), interaktive TB und automatisierte TB vorgestellt und zum Schluss gibt es einen Ausblick auf die Anwendungen und zukünftige Bestrebungen.

4.2 Der formale Beweis

Klassische Beweise sind in der Regel so gehalten, dass Mathematiker diese gut verstehen bzw. nachvollziehen können. Hinter Bezeichnungen und Abkürzungen verstecken sich dabei häufig sehr spezielle mathematische Ausdrücke in Form von Eigenschaften und Formeln, welche der Laie ohne Frage als Fremdsprache identifizieren kann. Darüber hinaus führen Mathematiker Beweise häufig schemenhaft oder auch anhand von Beispielen, die dann verallgemeinert werden können. So kommt es nicht selten vor, dass für eine Aussage mehrere Fälle gezeigt werden müssen, aber nur einer mit dem Zusatz vorgerechnet wird, dass die anderen Fälle analog behandelt werden. Ebenfalls gern verwendet werden Formulierungen wie „Ohne Beschränkung der Allgemeinheit“ (kurz OBdA) oder „trivial“, deren Bedeutung dem Mathematiker klar ist. Der klassische Beweis erfordert auch häufig kreative Wege und Beweisideen, die mitunter erst nach einigem Probieren, etwa Umformen und Umbenennen, ersichtlich werden. Ein formaler Beweis hingegen umfasst alles. Jede Implikation muss auf ihre Gültigkeit geprüft und jeder logische Schritt, bis zurück zu den fundamentalen Axiomen der Mathematik, muss vorhanden sein. Um zum Beispiel zu zeigen, dass ein Graph planar ist, reicht es dem Mathematiker, wenn der Graph entsprechend gezeichnet werden kann. Im formalen Beweis würde eine zusammenhängende Argumentationskette entstehen, die den theoretischen Kontext vollkommen abdeckt. Wie man sich leicht überlegt wird so etwas leicht unüberschaubar und umfangreich. Um dies ein wenig zu verdeutlichen möchte ich einen computergefertigten formalen Beweis vorstellen ohne dabei auf den mathematischen Hintergrund einzugehen. Als Beispiel sei die Robbins Algebra gewählt. Der Computer hat damals fast 8 Tage gebraucht um diese Lösung zu ermitteln. Brandon Fitelson benutz in seiner Arbeit „Using Mathematica to Understand the Computer Proof of the Robbins Conjecture“ [Fit98] eine andere Darstellungsform um den Beweis übersichtlicher und nachvollziehbarer zu gestalten. Er bemerkt, dass es trotzdem an einigen Stellen äußerst schwierig ist, zu verstehen, wie tatsächlich vorgegangen wird. Interessierten an den mathematischen Aspekten empfehle ich daher seine Aufarbeitung. Er nutzt die klassische Notation, in der die Negation in der typischen Boolschen Schreibweise dargestellt wird, wodurch dem Betrachter einige Erleichterungen beim Lesen verschafft werden. Ich zeige an dieser Stelle den eigentlichen Beweis von W. McCun, welcher mit dem Equational Theorem Prover (kurz EQP, ein automatisierter Theorembeweiser) angefertigt wurde und im Anschluss folgt eine aufgearbeitete Fassung von Thomas C. Hales [Hal08], in der übersichtlich zwischen Umformungen, Substitutionen und Anwendungen der Robbins Identität unterschieden wird.

Um beide Notationen in Einklang zu bringen sei gesagt, dass $n(\ldots)$, $[\ldots]$ und $\bar{}(x)$ die Negation darstellen und die Verknüpfung $(x,y) \to x+y$ aus dem 1. Teil durch $(x,y) \to xy$ in Hales Aufarbeitung ersetzt wurde. Der direkte Vergleich dieser beiden Beweise zeigt eindeutig, dass der Computerbeweis mitunter sehr schwer zu lesen ist und durchaus unhandlich sein kann. Heutzutage wird daher immer häufiger auf interaktive Beweise und vor allem auch eigene Ausgabestrukturen gesetzt, wie z.B. Isar (der Beweissprache von Isabelle). Nach Möglichkeit sollen diese sowohl der Maschine als auch dem Menschen ermöglichen, die Sprache gut zu lesen und zu verstehen. Wir können

diesen formalen Beweis nun als das Endergebnis verstehen, welches wir hier sehen, eine Kette von logischen Inferenzschritten.

4.3 Korrektheit eines computergefertigten Beweises

In diesem Abschnitt wird die Fragestellung diskutiert, in welchem Maße man auf die Korrektheit eines computergefertigten formalen Beweises vertrauen kann. Dazu wird zunächst darauf eingegangen, in wie weit menschliches Versagen die Korrektheit eines maschinellen Beweises gefährdet und anschließend wird das Problem abstrahiert, indem allgemeine Aussagen über die Korrektheit eines beliebigen Beweises gemacht werden.
Beschäftigt man sich mit der Frage, ob man eher einem computergefertigten Beweis als einem von Hand gefertigten Beweis vertraut, stößt man oft auf den ersten naiven Ansatz, dem Computer zu vertrauen, da diesem erfahrungsgemäß keine Flüchtigkeitsfehler unterlaufen. Dieser Ansatz ist auch durchaus berechtigt, es ist in der Vergangenheit schon oft vorgekommen, dass sich Beweise von viel diskutierten Theoremen erst einige Zeit nach ihrer Veröffentlichung als fehlerhaft herausgestellt haben. Jedoch ist zu beachten, dass auch maschinell erstellte Beweise von menschlichen Fehlern betroffen sein können. Man hat nämlich ebenfalls keine absolute Gewissheit, dass die Algorithmen, welche der Computer zum Beweisen benötigt, fehlerfrei sind, da diese auch vom Menschen geschaffen wurden. Hinzu kommt, dass maschinell erstellte Beweise meistens so kompliziert sind, dass sie nur sehr schwer per Hand nachgeprüft werden können. Also lässt sich zusammengefasst sagen, dass bei maschinell erstellten Beweisen und bei per Hand erstellten Beweisen die Gefahr von menschlichem Versagen besteht, aber die per Hand erstellten Beweise sich leichter überprüfen lassen.
Geht man nun davon aus, dass dem Menschen keine Fehler unterlaufen würden, hätte man leider trotzdem keine absolute Gewissheit, dass ein Beweis korrekt ist. Dies besagt der zweite Gödelsche Unvollständigkeitssatz, der besondere Bedeutung in der modernen Logik erlangt hat. Dieser Satz besagt:

Jedes hinreichend mchtige konsistente System kann die eigene Konsistenz nicht beweisen.

Im Folgenden wird eine stark vereinfachende Erklärung dieses Satzes vorgestellt, welche aber zum Verständnis der Problematik völlig ausreichend ist. Eine detaillierte und präzise Erläuterung wäre Aufgabe einer Vorlesung in mathematischer Logik und Mengenlehre.
Der Aufbau eines Beweises ist immer der selbe. Es sind mehrere Voraussetzungen gegeben und aus diesen Voraussetzungen werden so lange logische Schlussfolgerungen gezogen, bis man zu der zu beweisenden Aussage gelangt. Die Korrektheit eines Beweises hängt nun davon ab, ob die Schlussfolgerungen korrekt sind und ob die Voraussetzungen widerspruchsfrei sind. Der zweite Gödelsche Unvollständigkeitssatz besagt nun stark vereinfacht unter anderem, dass man die Widerspruchsfreiheit der Voraussetzungen nicht beweisen kann. Dies lässt sich auch leicht veranschaulichen. Angenommen, man wolle die Widerspruchsfreiheit der Voraussetzungen beweisen. Dazu würde man einen weiteren Beweis benötigen. Dieser hätte aber wieder andere Voraussetzungen, dessen Widerspruchsfreiheit man noch nicht bewiesen hat. Man kann nur die Widerspruchsfreiheit einer Menge von Annahmen zeigen, wenn diese Annahmen widersprüchlich sind. Aus einer widersprüchlichen Annahme lässt sich nämlich jede beliebige falsche Aussage ableiten, dazu gehört auch die Aussage, dass die widersprüchliche Annahme widerspruchsfrei ist.

Zusammenfassend lässt sich sagen, dass die allgemeine Korrektheit maschinell erstellter Beweise Raum für verschiedene Meinungen zur Verfügung stellt. Sowohl Beweise per Hand als auch maschinelle Beweise sind nicht vor Fehlern des Menschen geschützt. Fehler bei Beweisen, die vom Menschen erstellt wurden, lassen sich sich leichter entdecken, dafür arbeitet der Computer fehlerfrei, wenn die benötigten Algorithmen korrekt sind. Eine absolute Gewissheit, dass ein Beweis korrekt ist, wird man jedoch nie haben.

4.4 SAT / SMT

Hinter einem Erfüllbarkeitsproblem versteckt sich die Frage, ob es für eine aussagenlogische Formel eine Lösung gibt. Man stelle sich jetzt eine einfache mathematische Ungleichung vor, wie zum Beispiel $3+7<2\cdot x$. Auf die Frage ob eine natürliche Zahl x existiert, welche diese Formel erfüllt, werden die meisten im Schlaf eine Antwort geben können. Solche Formeln müssen natürlich nicht auf die natürlichen Zahlen, ja geschweige denn auf die

Mathematik beschränkt sein. Um eine Maschine anleiten zu können Theorien, bzw. Aussagen auf ihre Gültigkeit hin zu überprüfen oder gar diese zu beweisen, muss zunächst eine Struktur gefunden werden, auf deren Grundlage der Computer entscheiden kann, wann eine Aussage wahr oder falsch wird, eine Sprache. Ohne zu sehr ins Detail gehen zu wollen, sei an dieser Stelle gesagt, wir benötigen eine Prädikatenlogik. SAT formalisiert dieses Thema und führt Theorien ein, mithilfe derer wir die Aussagen logischer Konstruktionen, umformen bzw. Belegungen finden können, um diese zu lösen. Die „Satisfiability Modulo Theories“, kurz SMT, erweitert dieses System um die Möglichkeit Formeln und Terme um spezielle theoretische Aussagen zu erweitern. Dies sind in der Regel Terme, welche dann von dem SMT-Löser selbst gar nicht verarbeitet werden können und dessen Wahrheitsgehalt von einem sogenannten Theorie-Löser ermittelt werden muss. Bei den automatischen Theorembeweisern werde ich noch einmal darauf zurück kommen. Zusammenfassend ist SAT also die Theorie von der Lösbarkeit prädikatenlogischer Formeln und Terme und SMT eine Erweiterung dieser, durch theoretischen Kontext zunächst ungebundener Natur. Im TB stellt dieses Thema die Schnittstelle zwischen der Formulierung des Problems und deren Lösung dar.

4.5 Theorembeweiser

Wir unterscheiden vor allem zwei Sorten von Löser, die interaktiven und die automatisierten, wobei die Interaktiven heutzutage ebenfalls Methoden mitbringen, welche selbstständig nach dem nächsten Umformungsschritt suchen können. Der offensichtlichste Unterschied ist, dass die Automatisierten, wie der Name schon sagt, automatisch nach einer Lösung suchen. Das Programm wird mit allen nötigen Informationen gefüttert und den Rest übernimmt die Maschine.

4.5.1 Interaktive Theorembeweiser

Sie werden auch gerne Beweisassistenten genannt, da sie den Rahmen stellen um geordnet Beweise anzufertigen. Auf der einen Seite prüfen sie alle Inferenzschritte, können aber auf der anderen Seite auch eigenständige Lösungen mit einbringen. Ebenfalls hilfreich ist, dass sie dem Anwender aufzeigen an welchen Stellen noch Unklarheiten bestehen und welche Aussagen noch gezeigt werden müssen. Ein Nachteil ist, dass solche Beweiser zumeist ein hohes Maß an Verständnis für das Programm und seine Sprache vorraussetzen. In der Regel nutzen sie Logiken höherer Ordnung (kurz HOL, engl. „higher order logic“). Als Beispiele für diese Art der Beweiser möchte ich HOL Light, PVS, Coq, Mizar und Isabelle nennen. Im folgenden sehen wir 2 exemplarische Beweise mit Isabelle. Der Erste wird veranschaulicht, wie mit sogenannten „apply“-Scripts gearbeitet wird, das Zweite, wie mit Isar eine lesbare Form erreicht wird. Ursprünglich sollte hier ein eigenes kleines Beispiel in der zweiten Variante ausformuliert und mit allerhand Erklärungen ausgeschmückt werden, allerdings sprengt dies den Rahmen, sodass hier lediglich die Kenntnis von den beiden Varianten ausreichen soll.

Variante 1

Gegeben ist eine Listenstruktur mit zwei Funktionen, app und rev. Die Erste soll zwei Listen miteinander verknüpfen, die Zweite die Reihenfolge umkehren. Gezeigt werden soll, dass das doppelte Umkehren (der Reihenfolge der Elemente) einer Liste, die ursprüngliche Liste ergibt. In Abbildung 4.1 sieht man den Kopf des Beweises, mit den Definitionen des Datentyps dieser Listen und der beiden Funktionen app und rev.

Zunächst beginnen wir mit dem Theorem und arbeiten uns dann mit den Lemmas von unten nach oben, immer dann, wenn zusätzliche Aussagen gezeigt werden müssen. In |1 in Abbildung 4.2 haben wir Isabelle gesagt, dass eine Induktion über die Liste xs durchgeführt werden soll. In Abbildung 4.3 sieht man die Antwort darauf, die zu erfüllenden zwei Ziele:

- Induktionsanfang und
- Induktionsschritt.

Ähnlich sieht es bei den Lemmas aus. In Abbildung 4.4 sieht man, wie Isabelle den Beweis akzeptiert und eine Regel mit Platzhaltern zur Verfügung stellt, welche nun zur Vereinfachung genutzt werden kann. Lemma rev_app wird für beide Goals vom Theorem rev_rev gebraucht und die anderen beiden Lemmas für die beiden Goals von rev_app. „[sim]“ steht hier für eine Vereinfachung, d.h. Isabelle darf die Regeln anwenden um Termumformungen durchzuführen, z.B. durch „apply(simp)“.

Variante 2

In Abbildung 4.5 sieht man ein Beispiel, in dem der Beweis eigenhändig durchgeführt wird. Gezeigt werden soll, das Abbildungen von einer Menge in ihre Potenzmenge niemals surjektiv sein können. Ohne genau zu wissen wie

Abbildung 4.1: Isabelle Proof 1

```
theory Example1
imports Main
begin

declare [[names_short]]
datatype 'a list = Nil | Cons 'a "'a list"

fun app :: "'a list => 'a list => 'a list"
where
"app Nil ys = ys" |
"app (Cons x xs) ys = Cons x (app xs
ys)"

fun rev :: "'a list => 'a list" where
"rev Nil = Nil" |
"rev (Cons x xs) = app (rev xs) (Cons x
Nil)"
...
```

Abbildung 4.2: Isabelle Proof 2

```
...
lemma app_assoc [simp]: "app (app xs
ys) zs
= app xs (app ys zs)"
apply(induction xs)
apply(auto)
done

lemma app_Nil2 [simp]: "app xs Nil = xs"
apply(induction xs)
apply(auto)
done

lemma rev_app [simp]: "rev(app xs ys) =
app (rev ys) (rev xs)"
apply(induction xs)
apply(auto)
done                              |2

theorem rev_rev [simp]: "rev(rev xs) =
xs"
apply(induction xs)               |1
apply(auto)
done

end
```

Abbildung 4.3: Isabelle Antworten

```
proof (prove)
goal (2 subgoals):
 1. rev (rev Nil) = Nil
 2. ⋀x1 xs.
       rev (rev xs) = xs ⟹
    rev (rev (Cons x1 xs)) =
    Cons x1 xs
```

Abbildung 4.4: Isabelle akzeptierter Beweis

```
theorem app_Nil2: app
?xs Nil = ?xs
```

Abbildung 4.5: Isabelle Beweis

```
theory Example2
imports Main
begin

lemma "¬ surj(f :: 'a ⇒ 'a set)"
proof
 assume 0: "surj f"                    |3
 from 0 have 1: "∀ A. ∃ a. A = f a" by(simp add:
surj_def )
 from 1 have 2: "∃ a. {x . x ∉ f x } = f a" by blast
 from 2 show "False" by blast          |4
qed

end
```

Abbildung 4.6: Isabelle Beweisstatus

```
proof (state)
this:
 surj f

goal (1 subgoal):
 1. surj f ⟹ False
```

die neuen Begriffe, wie assume, from, have und show anzuwenden sind, kann man den Beweis doch bereits recht leicht lesen. In Abbildung 4.6 sieht man, wie sich der Beweisstatus in |3 verändert hat, nach der Annahme, dass f surjektiv sei. Es wird ein f festgehalten, das Goal hat sich nicht verändert. In Abbildung 4.7 sieht man die Ausgabe von Isabelle, bevor das Lemma akzeptiert wird (|4).

4.5.2 Automatisierte Theorembeweiser und DPLL(T)

Nachdem wir nun die interaktiven Theorembeweiser kennen gelernt haben, schauen wir uns die automatisierten etwas genauer an. Wir unterscheiden unter anderem zwei Methoden bei der automatisierten Beweissuche, den Davis-Putnam-Logeman-Loveland Algorithmus (kurz DPLL, auch DPLL(T) wobei T für Theorie steht) und das Tableaukalkül (auch Baumkalkül, bzw. engl. Method of analytic tableaux). Beide Herangehensweisen arbeiten auf einer Prädikatenlogik, allerdings sei das Baumkalkül nur als weiterer Vertreter mit aufgeführt. Grundlegend wird bei diesem versucht die Gegenaussage zu widerlegen.

4.6 Maschinengestützte Beweistechniken

4.6.1 Resolution

Der Begriff der Resolution bezeichnet ein Verfahren der formalen Logik, mit dem eine logische Formel auf Gültigkeit getestet werden kann. Das Resolutionsverfahren, auch Resolutionskalkül genannt, ist ein Widerlegungsverfahren. Hierbei wird ein logischer Widerspruch aus der Negation einer Formel abgeleitet, statt die Allgemeingültigkeit der Formel auf direktem Wege zu zeigen.
Diese Ableitung erfolgt mithilfe eines Algorithmus auf rein formalem Weg und kann aus diesem Grund von einem

Abbildung 4.7: Isabelle Status vor Lemma

```
show False
Successful attempt to solve goal by
exported rule:
 (surj f) ⟹ False
proof (state)
this:
 False

goal:
No subgoals!
```

Computerprgramm durchgeführt werden.

4.6.2 Termersetzungssysteme

Termersetzungssysteme (TES) sind ein formales Berechnungsmodell in der Theoretischen Informatik. Sie bilden insbesondere die Grundlage der logischen und funktionalen Programmierung. Weiterhin spielen sie eine wichtige Rolle beim Wortproblem und der Terminierungsanalyse.
Termersetzungssysteme sind Mengen von so genannten Termersetzungsregeln. Diese Mengen kann man sich wie Gleichungssysteme von Termen vorstellen, bei denen die Gleichung nur von links nach rechts angewendet werden dürfen.
Termersetzungssystem sind turing-vollständig, deren Berechnungsstärke ist also in etwa auf einem Level mit anderen Formalismen wie etwa dem Lambda-Kalkül oder den Turingmaschinen.

4.6.3 Modellprüfung

Modellprüfung(engl. Model Checking) ist ein Methode zur vollautomatischen Verifikation einer Systembeschreibung(Modell) gegen eine Spezifikation(Formel). Der Begriff ist abgeleitet von der mathematischen Formulierung des Problems: Für eine gegebene Systembeschreibung M und eine gegebene logische Eigenschaft Φ, prüfe, ob M Modell für Φ ist (formal $M \models \Phi$).
Model Checking wird als ein vollautomatisches Verfahren bezeichnet, da es bei der Durchführung keine Benutzerinteraktion nötig ist, anders als etwa beim interaktiven Theorembeweisen. Die Systembeschreibung wird in einer formalen Sprache erstellt, etwa mit einem Programm, einem endlichen Automaten oder einem Transitionssystem. Die Spezifikation ist eine formale Eigenschaft des Systems, die nachzuweisen ist.
Will man nun eine Spezifikation für ein System nachweisen, so gibt man diese in einen Modellprüfer ein und dieser stoppt und gibt ein Korrektheitszertifikat aus, falls die Systembeschreibung die Spezifikation erfüllt. Andernfalls gibt der model checker ein Gegenbeispiel aus, das eine Verletzung der Verifikation nachweist.

4.6.4 Induktion

Die vollständige Induktion ist, wie aus Mathevorlesungen bekannt, eine Bewismethode, bei der eine Aussage für alle natürlichen Zahlen, die größer als ein Startwert sind, gezeigt wird.
Ein Induktionsbeweis wird in zwei Phasen durchgeführt. Zuerst erfolgt der Induktionsanfang, bei dem die Aussage für eine kleinste Zahl gezeigtn wird, um sie dann im Induktionsschritt für eine variable Zahl n für die nächste Zahl n+1 logisch abzuleiten.
Dieses Verfahren ist von grundlegender Bedeutung für alle Gebiete der Mathematik und somit auch für maschinengestütztes Beweisen, bei dem man dem Computer die formale Aussage übergibt und sie von diesem durchgeführt wird, wobei hierfür meist einige Hilffsätze vom Computer benötigt werden, die man ihm ebenfalls übergibt.

4.6.5 Binäre Entscheidungsdiagramme

Ein Binäres Entscheidungsdiagramm(BED; engl. binary decision diagram, BDD) ist eine Datenstruktur zur Repräsentaion Boolescher Funktionen. Diese Diagramme werden hauptsächlich im Bereich der Hardwaresynthese und -verifikation eingesetzt.
Ein BED ist ein Graph mit den Variablen der Funktion als Knoten und den Entscheidungsmöglichkeiten *Wahr* oder *Falsch* als gerichtete Kanten. So kann man die Variablenbelegungen einer Funktion durchgehen und schauen, welche den Wert *wahr* beziehungsweise den Wert *falsch* haben.
Solche Diagramme können dann von einem Computer genutzt werden, um zwei Funktionen auf Gleichheit zu überprüfen oder zu testen, ob es eine Variablenbelegung einer Funktion mit dem Funktionswert *wahr* gibt, also, ob die Funktion erfüllbar ist.

4.7 Schlusswort

Nachdem wir nun einen Einblick in der Welt der TB erhalten haben und in Grundzügen wissen was TB sind und wie sie arbeiten, sollen folgend einige Anwendungen vorgestellt werden. Da Computerprogramme in verschiedenen Sprachen geschrieben werden, welche ein Vokabular, eine Grammatik und eine Syntax haben, können

diese genauso für den theoretischen Inhalt herhalten, wie mathematische Probleme. Ein Beispiel dafür ist der Beweis des Vier-Farben-Satzes von Neil Robertson, Daniel P. Sanders, Paul Seymour und Robin Thomas, welcher genau genommen zwei Beweise beinhaltet. Der erste zeigt, dass der benutzte Algorithmus richtig arbeitet, also verifiziert die Korrektheit und der zweite, dass das Theorem gilt. In der Wirtschaft setzt man Theorembeweiser vor allem für die Verifikation von integrierten Schaltkreisen und Prozessoren ein, z. B. um kritische Operationen zu prüfen. Allerdings finden sie auch in fachübergreifenden Gebieten ihren Nutzen. Insgesamt scheinen sie bei der Programmverifikation einer zunehmenden Beliebtheit zu unterliegen. Der seL4 Mikrokernel, welcher in unzähligen Mobilgeräten vorkommt, wurde zum Beispiel mithilfe von Isabelle geprüft und seine Korrektheit nachgewiesen. In der Mathematik sind sie noch sehr umstritten und auch wenn die vorhandenen Beweiser mächtige Werkzeuge sind, so können sie ihren menschlichen Kollegen heute noch nicht das Wasser reichen. Eine naheliegende Anwendung diesbezüglich wäre nämlich das Erschaffen von neuen Theoremen und dabei bringen sie höchstens triviale Aussagen zustande. Mit „Automated Theorem Discovery“ [GGC14] richtet man den Blick auf das Schaffen von neuem Wissen, was allerdings noch in den Kinderschuhen steckt. Das sogenannte „Theory-Exploration“ hingegen versucht Möglichkeiten zu verwirklichen, um dem Anwender von interaktiven TB, im Falle des Feststeckens, mit Lemmas und Ideen weiter zu helfen. Man bedenke, heutige TB zeigen nur, was noch nicht bewiesen ist und nicht, wie man solches zeigen kann. Als Fazit kann man sagen, dass mit dieser Arbeit die Themengebiete gerade einmal angekratzt werden konnte, denn es gibt noch mehr als genug Ungesagtes zu den Theorembeweisern. Mir persönlich hat die Arbeit von Thomas C. Hales über den Formalen Beweis sehr gut gefallen, da sie zeitgleich einen guten Überblick verschafft. Ich hatte darüber hinaus ein großes Interesse daran Isabelle einmal kennenzulernen und einen interaktiven TB selbst auszuprobieren. Auch wenn die Zeit nicht gereicht hat um elegante Beweise zu führen, so konnte man doch zumindest einen Eindruck gewinnen. Der Internetauftritt von Isabelle [UT] bietet diesbezüglich gut verständliche Tutorials an, die für Interessierte in jedem Fall empfehlenswert sind.

5. Bayessche Netze

5.1 Intro

Ein Bayessches Netz stellt die Wahrscheinlichkeiten von Ereignissen und deren Abhängigkeit (bzw. Unabhängigkeit) zueinander dar.

Bayessche Netze finden immer dort Anwendungsmöglichkeit, wo Logik und Unwissenheit aufeinandertreffen. Sie dienen der Vereinfachung von komplexen Problemen mit Hilfe weniger stochastischer Regeln.

In der Praxis finden Bayessche Netze beispielsweise in der Spracherkennung, medizinischen Diagnose, Filtern von Spam, Bildverarbeitung, Analyse von Kaufverhalten und in vielen anderen Gebieten Anwendung.

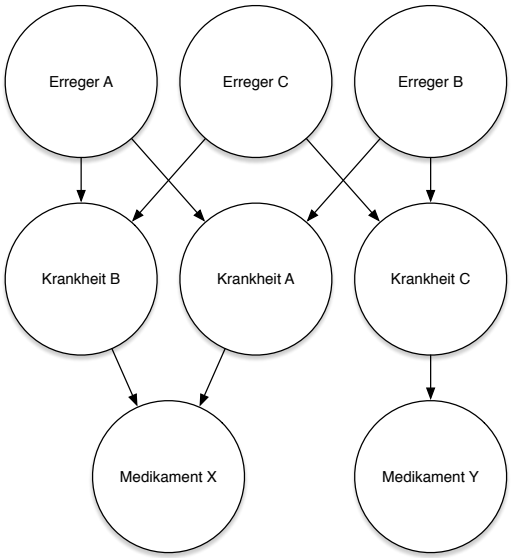


Abbildung 5.1: Beispiel eines Bayesschen Netzes zur Diagnose von Krankheiten (ohne Wahrscheinlichkeiten)

5.2 Grundlagen der Wahrscheinlichkeitsrechnung

Einem Ereignis A weisen wir eine Wahrscheinlichkeit p(A) zwischen 0 (tritt nie ein) und 1 (tritt immer ein) zu.

Als marginal probability bezeichnen wir eine Wahrscheinlichkeit p(A), welche keine Abhängigkeiten aufweist. Beispiel: A = Eine aus einem Skatdeck gezogene Karte ist Kreuz. p(A) = 1/4 (oder 25 %)

Als joint probability bezeichnen wir Wahrscheinlichkeiten, welche nebeneinander auftreten. p(A,B) Beispiel: A (von oben) und B: Die Karte ist ein Bube

Da A und B voneinander unabhängige Ereignisse sind folgt: $p(A,B) = p(A) \cdot p(B) = 1/8 \cdot 1/4 = 1/32$

Sind Ereignisse jedoch nicht unabhängig, so müssen wir die Wahrscheinlichkeit von p(A,B) bereits bestimmt haben. Möchten wir nun für eine große Anzahl Ereignisse Wahrscheinlichkeiten berechnen, so benötigen wir extrem viele Daten.

Sonne scheint	Werktag	Stau	Rasensprinkler	Tage im Jahr
F	F	F	F	4
F	F	F	T	5
F	F	T	F	2
F	F	T	T	1
F	T	F	F	13
F	T	F	T	2
F	T	T	F	66
F	T	T	T	...
T	F	F	F	...
T	F	F	T	
T	F	T	F	
T	F	T	T	
T	T	F	F	
T	T	F	T	
T	T	T	F	
T	T	T	T	Errechenbar

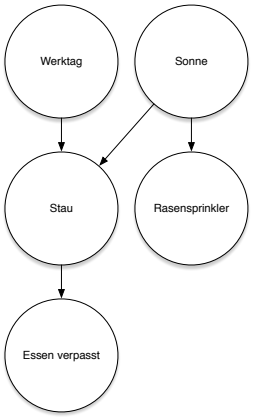
Bis auf 1 errechenbares Ergebnis benötigen wir die direkten Werte aller Kombinationen. Wir benötigen also Für N Ereignisse $2^N - 1$ Datensätze. Gerade in der Medizin aus dem Anfangsbeispiel gibt es aber oft extrem viele Ereignisse.

Oftmals ist vieles davon aber auch nicht interessant, bzw. ableitbar. Die Wahrscheinlichkeit, dass die Sonne scheint ist beispielsweise unabhängig davon ob der Tag ein Werktag ist.

Um die benötigten Datensätze möglichst gering zu halten betrachtet man bedingte Wahrscheinlichkeiten (conditional probability).

5.3 Bedingte Wahrscheinlichkeit

Bedingte Wahrscheinlichkeiten beruhen auf Annahmen. Man nimmt etwa an, dass die Sonne keinen Einfluss auf die Frage hat, ob es nun ein Werktag ist, sehr wohl jedoch auf die Frage, ob der Sprinkler angeschaltet ist. Gehen wir weiterhin davon aus, dass an sonnigen Tagen mehr Staus zustande kommen, da z.B. Fahrer geblendet werden. Zudem gehen wir davon aus, dass werktags mehr Staus auftreten. Fügen wir zudem noch das Verpassen des Essens hinzu, welches lediglich vom Stau abhängt.



Unsere benötigten Datensätze mit fiktivem Inhalt:
$p(Sonne) = 0,8$
$p(Werktag) = 0,7$
$p(Rasensprinkler|Sonne) = 0,95$
$p(Rasensprinkler|!Sonne) = 0,20$
$p(Stau|Sonne, Werktag) = 0,8$
$p(Stau|!Sonne, Werktag) = 0,75$
$p(Stau|Sonne, !Werktag) = 0,15$
$p(Stau|!Sonne, !Werktag) = 0,05$
$p(Essenverpasst|Stau) = 0,5$
$p(Essenverpasst|!Stau) = 0,05$

Von 31 Datensätzen in einer Tabelle mit joint probabilities haben wir nun nur noch 10 benötigt Wahrscheinlichkeiten und können alle anderen nun problemlos errechnen. (Beispiel folgt)

Um bedingte Wahrscheinlichkeiten effektiv nutzen zu können benötigen wir einige Definitionen:

$$p(A,B) = p(A|B) \cdot p(B)$$

Daraus folgt direkt:

$$p(A,B,C) = p(A|B,C) \cdot p(B,C) = p(A|B,C) \cdot p(B|C) \cdot p(C)$$

Dies lässt sich für N Ereignisse wie folgt darstellen (Chain Rule):

$$P\left(\bigcap_{k=1}^{N} A_k\right) = \prod_{k=1}^{N} \mathrm{P}\left(A_k \,\middle|\, \bigcap_{j=1}^{k-1} A_j\right)$$

Zudem bezeichnen wir A als von B unabhängig, wenn gilt:

$$p(A|B) = p(A)$$

und A als von B unabhängig unter der Prämisse C, wenn gilt:

$$p(A|C) \cdot p(B|C) = p(A,B|C)$$

Mit Hilfe der Chain-Rule können wir nun für das Beispiel von oben jede mögliche joint probability ausrechnen.

5.3.1 Rechenbeispiel

Wir suchen eine Möglichkeit folgenden Ausdruck zu berechnen:

$$p(Essen\ verpasst, Stau, Werktag, Rasensprenger, Sonne)$$

Nach Anwendung der Chain Rule haben wir:

$$\begin{aligned} &p(Essen\ verpasst|Stau, Werktag, Essen\ verpasst, Sonne) \\ &\cdot p(Stau|Werktag, Rasensprinkler, Sonne) \\ &\cdot p(Werktag|Rasensprinkler, Sonne) \\ &\cdot p(Rasensprinkler|Sonne) \\ &\cdot p(Sonne) \end{aligned}$$

Wichtig ist hier den Aufbau des ersten Ausdrucks entlang der Abhängigkeiten zu formulieren. Die Reihenfolge der Ereignisse für die joint probabilty muss also eine umgedrehte topologische Anordnung unseres Graphen darstellen bzw. ein Ereignis A kann nicht vor einem Ereignis B stehen, wenn B abhängig von A ist.
Nach dem Kürzen der Unabhängigkeiten erhalten wir:

$$p(Essen\ verpasst|Stau) \cdot (Stau|Werktag, Sonne) \cdot p(Werktag) \cdot p(Rasensprinkler|Sonne) \cdot p(Sonne)$$

Man nennt dies auch die Faktorisierung.
Die gesuchten Wahrscheinlichkeiten kennen wir bereits und können sie einsetzen:

$$0,5 \cdot 0,8 \cdot 0,7 \cdot 0,95 \cdot 0,8 = 0,218.$$

Es ist also durch simples Einsetzen unserer 10 Werte möglich sämtliche 31 Werte für die joint probability Tabelle zu berechnen. Gleichzeitig sind Lösungen für relevante Fragen zur bedingten Wahrscheinlichkeit, welche nicht alle Ereignisse umfassen, mit geringem Aufwand zu lösen. In der Regel sind jedoch nicht alle Angaben bekannt. Nehmen wir nun an, dass wir als hart arbeitender Mensch an einem Nicht-Werktag zur Arbeit fahren und es sonnig ist. Wir möchten nun wissen, wie wahrscheinlich es ist, dass man es abends bei der Rückfahrt rechtzeitig zum Essen schafft. Wir suchen also:

$$p(!Essen\ verpasst | !Werktag, Sonne)$$

Dazu berechnen wir:

$$p(Essen\ verpasst|Stau) \cdot p(Stau|Sonne,Werktag) + p(Essen\ verpasst|Stau) \cdot p(Stau|Sonne,Werktag)$$

Die passenden Werte kennen wir entweder oder können sie direkt ableiten:

$$\begin{aligned}
&p(Essen\ verpasst|Stau) = 0,5 \rightarrow p(Essen\ verpasst|Stau) = 0.5\\
&p(Essen\ verpasst|Stau) = 0,05 \rightarrow p(Essen\ verpasst|Stau) = 0.95\\
&p(Stau|Sonne,Werktag) = 0,15 \rightarrow p(Stau|Sonne,Werktag) = 0.85
\end{aligned}$$

$$0,5 \cdot 0,15 + 0,95 \cdot 0,85 = 0,8825$$

bzw. die Wahrscheinlichkeit am Wochenende pünktlich zum Essen zu kommen beträgt 88%.
Was jedoch, wenn ich in die andere Richtung rechnen möchte? Beispiel: Der Lebensgefährte kommt pünktlich zum Essen und ich möchte wissen, wie wahrscheinlich es ist, dass er im Stau gesteckt hat. Wir kennen bereits die Formel:

$$p(A,B) = p(A|B) \cdot p(B)$$

offensichtlich gilt auch:

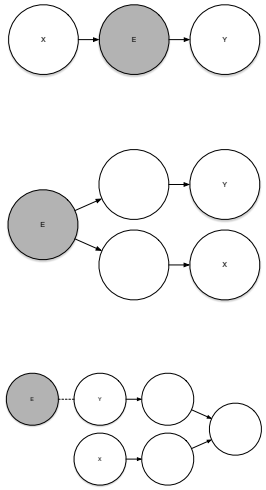
$$\begin{aligned}
&p(B,A) = p(B|A) \cdot p(A)\\
\Rightarrow &p(A|B) = p(A,B)/P(B) = p(B,A)/P(B) = p(B|A) \cdot p(A)/p(B) \text{ (Satz von Bayes)}
\end{aligned}$$

Wir wissen also:

$$\begin{aligned}
&p(Stau|!Essen\ verpasst) = p(!Essen\ verpasst|Stau) * P(Stau)/p(!Essen\ verpasst)\\
&p(!Essen\ verpasst|Stau) = 0,5\\
p(Stau) = &p(Stau|Sonne,Werktag) \cdot p(Sonne) \cdot p(Werktag) +\\
&p(Stau|Sonne,!Werktag) \cdot p(Sonne) \cdot p(!Werktag) +\\
&p(Stau|!Sonne,Werktag) \cdot p(!Sonne) \cdot p(Werktag) +\\
&p(Stau|!Sonne,!Werktag) \cdot p(!Sonne) \cdot p(!Werktag) = 0,382\\
&p(!Essen\ verpasst) =\\
&p(!Essen\ verpasst|Stau) \cdot p(Stau) + p(!Essen\ verpasst|!Stau) \cdot p(!Stau) = 0,7781\\
&\Rightarrow p(Stau|!Essen\ verpasst) = 0,5 \cdot 0,382/0,7781 = 0,2454...
\end{aligned}$$

Ist die Person also pünktlich, so gab es mit einer Wahrscheinlichkeit von 75% keinen Stau. An dieser Stelle ist anzufügen, dass das ganze Problem auch so hätte modelliert werden können, dass die Pünktlichkeit beim Essen ebenfalls vom Werktag abhängig ist, dann jedoch wäre die Abhängigkeit zum Stau zu hinterfragen, wenn es denn keinen Werktag gibt. Es gibt also durchaus komplexere Problemstellungen, die wir mit unseren einfachen Methoden nicht so einfach behandeln können und eventuell verschiedene Modelle für Abhängigkeiten die je nach der Menge der Daten möglicherweise nicht optimal sind. Diese neue Methode hilft uns vor allen Dingen damit mit einem einzelnen Modell gleichzeitig z.B. Krankheiten und Symptome in beide Richtungen zu bestimmen. Ohne große Umstände kann eine Krankheit aus Symptomen bestimmt werden und direkt von der Wahrscheinlichkeit der Krankheit kann die Wahrscheinlichkeit für weitere Symptome berechnet werden.

5.4 Graphische Darstellung



Wir nennen zwei Ereignisse X und Y bedingt unabhängig gegeben E, wenn im Graphen kein Weg von X zu Y existiert, welcher nicht E nicht beinhaltet. Dies wir auch d-Separation genannt. Wir schreiben hierfür $X \perp Y|E$ Ausgehend vom vorherigen Beispiel heißt das, dass das Verpassen des Essens zwar vom Werktag abhängt, jedoch wenn bekannt ist, ob es einen Stau gab, diese Abhängigkeit aufgehoben wird. Für 3 verknüpfte Ereignisse gibt es die Folgenden Möglichkeiten:

(a) $A \rightarrow B \rightarrow C$ impliziert $A \perp C|B$
(b) $A \leftarrow B \leftarrow C$ impliziert $A \perp C|B$
(c) $A \leftarrow B \rightarrow C$ impliziert $A \perp C|B$
(d) $A \rightarrow B \leftarrow C$ impliziert $A \perp C$

Das Beispiel (d) beschreibt hierbei eine sogenannte V-Struktur.

Betrachten wir hierzu außerdem die Verschiedenen Faktorisierungen:

(a) p(A,B,C) = p(C|B) · p(B|A) · p(A)
(b) p(A,B,C) = p(A|B) · p(B|C) · p(C)
(c) p(A,B,C) = p(A|B) · p(C|B) · p(B)
(d) p(A,B,C) = p(B|A,C) · p(A) · p(C)

5.5 IC-Algorithmus

Der IC Algorithmus ist eine Möglichkeit aus gegebenen Unabhängigkeiten ein Bayessches Netz zu erstellen.

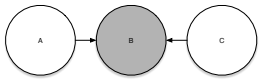
1. Konstruiere einen ungerichteten Graphen; füge jede mögliche Kante (X,Y), für die es keine (bedingte) Abhängigkeit $X \perp Y|E$ bzw. $X \perp Y$ gibt, in den Graphen ein.
2. Gilt für zwei Nachbarn X,Y von einem Knoten E die bedingte Unabhängigkeit $X \perp Y|E$ nicht, dann füge die Kanten (X,E) und (Y,E) hinzu. (V-Struktur)
3. Orientiere die verbleibenden Kanten beliebig, aber ohne neue V-Strukturen entstehen zu lassen.

5.6 Erstellung eines Bayesschen Netzes

Nach unseren theoretischen Überlegungen sind wir zur Erkenntnis gelangt, dass uns Bayessche Netze nicht nur ein intuitives Verständnis bzgl. bedingten Wahrscheinlichkeiten vermitteln, indem Abhängigkeitsverhältnisse zwischen Zufallsvariablen explizit angegeben werden, sondern auch langwierige Rechnungen erheblich vereinfachen können. Allerdings stellt sich für uns immer noch die Frage, wie wir aus einem probabilistischen Problem ein geeignetes Netz entwickeln können.

5.6.1 Aufbau eines Bayesschen Netzes aus Expertenwissen

Eine Möglichkeit wäre die Hilfe von Experten in Anspruch zu nehmen. Ihre Aufgabe besteht darin Abhängigkeitsverhältnisse anzugeben und somit eine Topologie zu entwickeln sowie die entsprechenden Parameter der Knoten anzupassen. Diese Herangehensweise ist jedoch meist nur für kleine bzw. unveränderliche Probleme geeignet, da



sie zahlreiche Schwierigkeiten mit sich bringt. Zum einen existiert ein Komplexitätsproblem. Offensichtlich kann allein die schiere Anzahl von Zufallsvariablen diesen Lösungsversuch zunichtemachen. Aber wie ist es mit einer begrenzten, übersichtlichen Anzahl an Zufallsvariablen?

Erinnern wir uns an die Kanten des Graphen. Jede Kante beschreibt eine Wahrscheinlichkeit und kann einen Wert zwischen 0 und 1 annehmen. Somit ist jede Wahrscheinlichkeit maximal eine gute Approximation der Realität. Da es oft der Fall ist, dass jede Zufallsvariable von mehreren Eltern abhängt und ihrerseits die Wahrscheinlichkeit ihrer Kinder bestimmt, können bereits kleinste Fehler zu großem Schaden führen und das Bayes Netz somit unbrauchbar machen. Dies lässt die Schlussfolgerung zu, dass ein konstruiertes Netz von verschiedenen Experten auf Fehler untersucht werden sollte, sodass der ganze Entwicklungsprozess sehr langwierig wird. Dennoch kann das Ergebnis nicht verifiziert werden.

5.6.2 Aufbau eines Bayesschen Netzes aus selbstlernenden Algorithmen

Da der Aufbau eines Bayesschen Netzes aus Expertenwissen einige Probleme mit sich bringt, haben Wissenschaftler eine weitere Methode entwickelt. Sie besteht darin, dass Netz in die Lage zu versetzen, anfallende Daten zu analysieren und auf Grundlage des Ergebnis Justierungen vorzunehmen. Da solche Algorithmen sehr komplex sind, werden im folgenden nur die Grundzüge der Verfahren vorgestellt und einhergehende Schlüsselwörter genannt. Grundsätzlich gibt es zwei Möglichkeiten ein Netz zu erstellen bzw. zu verbessern.

Anpassung der Parameter

Ein Experte erstellt eine sinnvolle Topologie und fügt Parameter ein. Diese Parameter beruhen auf A-priori Wahrscheinlichkeiten und sollten eine gute Approximation darstellen. Solch ein Netz wird „augmented Bayesian network“ genannt. Die Idee besteht darin, dass die Werte der Parameter kontinuierlich an die anfallenden Daten angepasst werden. Dies hat zur Folge, dass dasNetz umso bessere Vorhersagen treffen kann, je mehr Daten bereits verarbeitet wurden. In diesem Kontext spielt besonders dierelative Häufigkeit eine wichtige Rolle. Darunter versteht man wie oft ein gewisses Ereignis in einer Zeitspanne eintrat.
Beispiel: Wir werfen eine Münze. Wir schätzen, dass die Wahrscheinlichkeit 50 % beträgt, dass das Ereignis Kopf eintritt. Nach dem 10.000 Wurf stellen wir fest, dass 7000 mal das Ereignis Kopf eingetreten ist. Somit beträgt die relative Häufigkeit 70 % und wir müssen unseren anfangs gewählten Wert anpassen.
Diese Art von Berechnung ist jedoch nur möglich, wenn die relative Häufigkeit eines Parameters mit [0,1] gleich wahrscheinlich ist. In vielen Fällen muss jedoch ein anderer Weg eingeschlagen werden, da diese Art von Verteilung nicht angenommen werden kann. Somit benötigen wir eine Verteilungsfunktion, die in der Lage ist solche Verteilungen darzustellen. Zu diesem Zweck bietet sich besonders die Familie der Beta Wahrscheinlichkeitsdichtefunktionen an, da diese über die Eigenschaft verfügen, nach Dateneingang dynamisch und kontinuierlich angepasst werden zu können.

Erstellung der Topologie

Die zweite Möglichkeit ein Bayes Netz durch Algorithmen konfigurieren zu lassen, besteht darin die Struktur eines Bayesschen Netzes entsprechend der Datenlage anzupassen. Eine populäre Methode ist der „score-based approach“. Dabei wird jeder Struktur ein Score zugewiesen, der angeben soll, wie geeignet gegebene Daten durch eine beliebige Struktur beschrieben werden. Der Score wird folgendermaßen berechnet:

P(G|D) = Score(G, D), mithilfe der Bayesschen Formel zu

$$P(G|D) = \frac{P(D|G) * P(G)}{P(D)} \tag{5.1}$$

Da uns die zur Verfügung gestellten Daten nicht weiter interessieren, konzentrieren wir uns einzig auf den Zähler, da wir das Ziel haben diesen Ausdruck zu maximieren. P(G) kann durch verschiedene Möglichkeiten mithilfe von „prior information“ bestimmt werden. Prior information geben Wahrscheinlichkeitsverteilungen an, bevor es Beweise für deren Richtigkeit gibt. Sie beruhen meist auf früheren Beobachtungen oder entstehen durch Vergleich mit ähnlichen Problemen. Somit liegt der Fokus in der Maximierung von P(D|G). Es wurden viele

verschiedene Möglichkeiten entwickelt dieses Problem zu lösen, eine Standardmethode beinhaltet meistens die Likelihood-Berechnung, ein Schätzverfahren, dass die Maximierung eines Ausdrucks anstrebt durch Schätzen der Parameter anstrebt.

5.6.3 Vorteile von Bayesschen Netzen

Bayessche Netze ersparen ihren Anwender enorm viel Arbeit. Sie reduzieren die benötigten Wahrscheinlichkeitstabellen auf ein Minimum, sodass manche Anwendungen laufzeit-technisch erst möglich werden. Beispielsweise benötigt die Berechnung der Joint probability eine Laufzeit aus $O(2^n)$, unter der Bedingung, dass die gewählten Zufallsvariablen binär sind, mithilfe von unserem Netz eine Laufzeit aus $O(2^{pa(x)})$ wobei pa(x) für die Anzahl aller Eltern eines Knotens x steht.

Ein weiterer Vorteil von einem Bayes Netz liegt darin, dass der Aufbau des Netzes eine übersichtliche grafische Darstellung ermöglicht und im Gegensatz zu neuronalen Netzen nicht als Black-Boxen für viele Benutzer wahrgenommen werden. Es wird durch sie intuitiv erkennbar, welche Abhängigkeiten bestehen und mit welcher Wahrscheinlichkeit Ereignisse eintreten werden. Besonders für Fachfremde wird das Verständnis enorm verbessert, was direkte Folgen für die Lösung der Probleme hat. Ein Arzt wird der Interpretation von Symptomen einer Künstlichen Intelligenz mehr Vertrauen schenken, wenn er weiß, wie sie funktioniert.

Außerdem sollte nicht unterschätzt werden, dass ein bereits angelegtes Bayessches Netz nicht auf einen einzigen Typus einer Problemstellung reduziert werden muss. Nennen wir als Beispiel ein Netz, dass die verschiedenen Mikrocontroller innerhalb eines PKWs verknüpft, sodass diese im Zusammenspiel entscheiden können, wann die automatische Notbremsung eingeleitet werden soll. Produziert man nun ein ähnliches Modell mit einer unterschiedlichen Bremsvorrichtung, so muss man lediglich das Subnetz der Bremsen kappen um das neue System eingliedern zu können. Dadurch können Bayessche Netze als flexibel interpretiert werden, da man bis zu einem bestimmtem Grad das System an seine Bedürfnisse anpassen kann.

5.6.4 Nachteile von Bayesschen Netzen

In der Praxis treten einige Probleme mit Bayesschen Netzen auf. In unseren theoretischen Überlegungen finden wir das Axiom der zyklenfreiheit unseres Graphen. Dies ist jedoch oft nicht vorhersagbar, da unbekannte Abhängigkeiten zwischen Zufallsvariablen existieren können.

Es treten insbesondere vermehrt Probleme auf, wenn das Netz selbst lernend ist. Zum einen besteht die Chance von Manipulation, wenn es von User-Inputs abhängt. Gibt ein User gezielt gefälschte Daten ein, so kann es passieren, dass die Wahrscheinlichkeitsberechnungen der einzelnen Knoten gestört werden, was zur Folge hat, dass auch ihre Kind-Knoten beeinflusst werden können. Da oft mehrere verschiedene Kind-Knoten pro Knoten existieren, geben diese dann ihrerseits den Input weiter, sodass oft ein großer Teil des Netzes manipulierbar ist.

Auch der automatische Aufbau der Topologie erweist sich als problematisch. Um die best mögliche Anordnung zu erreichen, muss jede Anordnung berechnet werden. Dies kann sehr zeitintensiv werden und bei einer höheren Anzahl an Zufallsvariablen unter Umständen unmöglich. Dieses Problem ist NP-schwer.

5.7 Quellen

- Open Course „Artifical Intelligence“ am MIT
- https://upload.wikimedia.org/math/b/5/a/b5a87dba9ec79dd6a93628c85fab ca97.png
- http://www.informatik.uni-bremen.de/tdki/lehre/ss12/bayes/Intro.pdf
- http://www.fil.ion.ucl.ac.uk/ wpenny/bdb/bayes.pdf
- Bayesian Artificial Intelligence, Second Edition
- https://www.cs.cmu.edu/ dmarg/Papers/PhD-Thesis-Margaritis.pdf
- https://en.wikipedia.org/wiki/Prior_probability
- http://www.markmeloon.com/some-advantages-of-bayesian-networks/
- http://niedermayer.ca/book/export/html/29
- http://www.cs.technion.ac.il/ dang/books/Learning%20Bayesian%20Networks(Neapolitan,%20Richard).pdf

III Suche

6. Tiefensuche, Beam-Search und Hill Climbing

6.1 Motivation

Die Hauptaufgabe von Rechnern ist, Probleme zu lösen. Oft kann man die Probleme graphisch darstellen, mit einem Start- und einem Zielknoten und die Wege zwischen den beiden. Hierfür benötigt man nicht nur eine Darstellung des aktuellen Zustandes, sondern auch die Möglichkeiten, die zur Verfügung stehen, um das Ziel zu erreichen. Meistens gibt es keine einfache Lösung, denn nicht alle Probleme sind Addition von zwei positiven ganzen Zahlen. Daher benötigt man Suchverfahren. Hier beschränken wir uns auf die Depth-First Search, Breadth-First Search, Hill Climbing und Best-First Suchverfahren.

6.2 Suchalgorithmen

Wenn man vom Problem und der Darstellung abstrahiert, kann man die Suche nach einer Lösung in einem gerichteten Graphen betrachten. Der gerichtete Graph wird nicht gegeben, sondern wird durch Erzeugungsregeln abgeleitet. Der Graph kann auch unendlich groß sein und daher ist die Suche und partielle Erzeugung des Graphen verflochten.

Unser Suchgraph besteht hier aus Knoten, die die Zustände beschreiben, und Kanten, in Form von Funktionen, die die Nachfolgerknoten berechnen. Außerdem definieren wir eine Anfangssituation und ein Ziel.

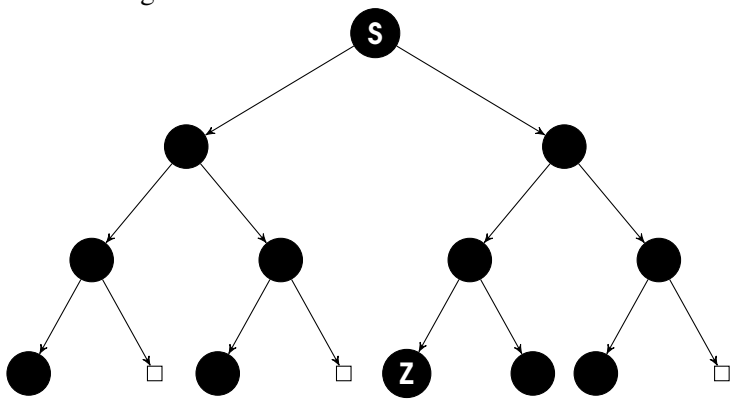


Fig1: Graphische Darstellung eines Problems
S: Startknoten, Z: Zielknoten

6.3 Uninformierte Suche

Wir betrachten zunächst die nicht informierte Suche. Hierbei wird bei der Suche nur der Graph und die Nachfolgerfunktion berücksichtigt, aber keine anderen Informationen über die Knoten, Kanten usw. dürfen verwendet werden.

Die Parameter für unseren Graph sind

- Menge der initialen Knoten
- Menge der Zielknoten, bzw. eindeutige Festlegung der Eigenschaft der Zielknoten
- Nachfolgerfunktion N

Damit lässt sich folgender Algorithmus schreiben:

```
1: function GENERAL-SEARCH(problem,strategy)
2:     returns a solution, or failure
3:     initialize the search tree using the initial state of problem
4:     loop
5:         choose a leaf node for expansion according to strategy if there are no candidates for expansion then
           failure
           end
           if the node contains a goal state then
           the corresponding solution
           end
           else
           expand the node and add the resulting nodes to the search tree
           end
6:     end loop
7: end function
```

Algorithm 1: General-Search Algorithm

Wir betrachten im Folgenden Varianten der blinden Suche, die insbesondere das Wählen des Knotens eindeutig durchführt.

6.3.1 Depth-First Search

Die Depth-First Suche verwendet eine Liste von Knoten und wählt den als nächsten zu betrachtenden Knoten aus dieser Liste. Außerdem werden neue Nachfolger vorne in die Liste eingefügt, was zur Charakteristik führt, dass zunächst in der Tiefe gesucht wird.

```
1: function DEPTH-FIRST-SEARCH(problem)
2:     returns a solution, or failure
3:     function GENERAL-SEARCH(problem,Enqueue-At-Front)
4:     end function
5: end function
```

Algorithm 2: Depth-First Search Algorithm

Die Depth-First Suche eignet sich damit besonders für Suchbäume, deren Äste eine vertretbare Länge nicht überschreiten.

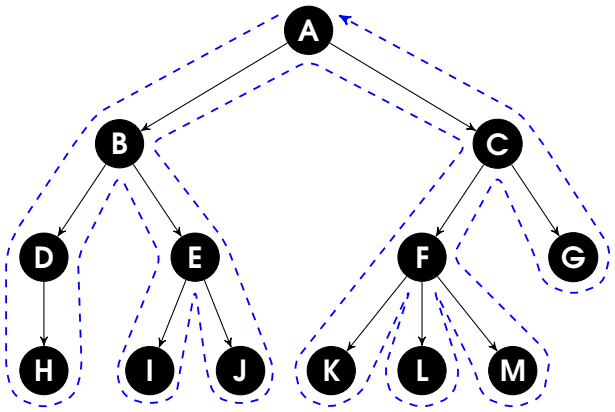


Fig2: Graphische Darstellung der Depth-First Suche

Eine mögliche Implementierung der Depth-First Suche in Prolog ist:

```
1  loesung(Start,Ziel):-
2    weg(Start,Ziel,[],Pfad),
3    write('Pfad:␣'),  % Aufruf der Tiefensuche
4    write(Pfad).
5
6  % weg(Startknoten, Zielknoten, Liste der
7  %          besuchten Knoten, Ergebnispfad)
8
9  weg(Knoten,Knoten,Liste,Pfad):-
10   reverse(Liste,Pfad).
11
12 weg(Start,Ziel,Liste,Pfad):-
13   kante(Start,Knoten),
14   not(member(Knoten,Liste)),
15   weg(Knoten,Ziel,[Knoten|Liste],Pfad).
```

Und in Haskell:

```
1  dfs ::(a -> Bool)     -- Zieltest (goal)
2    -> (a -> [a])       -- Nachfolgerfunktion (succ)
3    -> [a]              -- Startknoten
4    -> Maybe (a, [a]) -- Ergebnis: Just (Zielknoten,Pfad)
5                        --       oder Nothing
6  dfs goal succ stack =
7    -- Einfuegen der Anfangspfade, dann mit iterieren mit go
8    go [(k,[k]) | k <- stack]
9   where
10   go [] = Nothing          -- Alles abgesucht, nichts gefunden
11   go ((k,p):r)
12     | goal k = Just (k,p) -- Ziel gefunden
13     | otherwise = go ([(k',k':p) | k' <- succ k] ++ r)
```

Die explizite Speicherung der Pfade sowohl in Haskell als auch in Prolog kostet nicht viel Effizienz. Man kann den Algorithmus auch in imperativen Sprachen implementieren, indem jeder Eintrag einen Knoten mit Zeiger auf den nächsten Nachbarknoten zugeordnet wird. Die Komplexität des Verfahrens (worst-case) für die Zeit ist entsprechend der Anzahl der besuchten Knoten exponentiell in der Tiefe und für den Speicherplatz linear bei fester Verzweigungsrate.

Das Problem bei der Depth-First Suche ist, dass der Algorithmus nicht vollständig ist. Wenn der Suchgraph unendlich groß ist, kann die Suche am Ziel vorbeilaufen und für immer in einem unendlichen langen Pfad laufen.

Um diesem Problem vorzubeugen kann man eine Tiefenbeschränkung (k) einbauen. Wenn die vorgegebene Tiefenschranke (k) überschritten wird, werden keine Nachfolger dieser Knoten mehr erzeugt. Die Depth-First Suche findet in diesem Fall jeden Zielknoten, der höchstens Tiefe k hat.

Wenn der gerichtete Graph kein Baum ist, kann man auch die schon untersuchten Knoten in einer Hash-Tabelle speichern und somit werden die Knoten nicht nochmal untersucht, sondern redundant weiterverfolgt.

Hierfür ist der Platzbedarf entsprechend der Anzahl der besuchten Knoten und die Berechnungszeit gleich $n \cdot log(n)$ mit n = Anzahl der untersuchten Knoten.

Backtracking

Die Arbeitsweise der Depth-First Suche ist: Wenn Knoten K keine Nachfolger hat (oder eine Tiefenbeschränkung) überschritten wurde, dann mache weiter mit dem nächsten Bruder von K. Ist der Pfad nicht erfolgreich, führt die Depth-First Suche Backtracking durch. Dies wird als chronologisches Zurücksetzen bezeichnet.

Es gibt auch Suchprobleme, bei denen kein Backtracking erforderlich ist (greedy Verfahren ist möglich) zum Beispiel, wenn jeder Knoten noch einen Zielknoten als Nachfolger hat. Die Depth-First Suche reicht dann aus, wenn die Tiefe der Zielknoten in alle Richtungen unterhalb jedes Knotens beschränkt ist. Anderenfalls reicht Depth-First Suche nicht aus, da ein Ast immer ausgesucht werden kann, in dem der Zielknoten weiter weg ist.

6.3.2 Breadth-First Search

Die Breadth-First Suche verwendet auch eine Liste von Knoten wie bei der Depth-First Suche. Der Unterschied hierbei ist, dass die neuen Nachfolger hinten in die Liste eingefügt werden, was zur Charakteristik führt, dass zunächst in der Breite gesucht wird.

```
1: function BREADTH-FIRST-SEARCH(problem)
2:     returns a solution, or failure
3:     function GENERAL-SEARCH(problem,Enqueue-At-End)
4:     end function
5: end function
```

Algorithm 3: Breadth-First Search Algorithm

Die Breadth-First Suche untersucht somit ausgehend von Startknoten alle Nachbarknoten. Ist der Zielknoten noch nicht erreicht, werden alle Nachbarknoten der bisher untersuchten Knoten betrachtet.

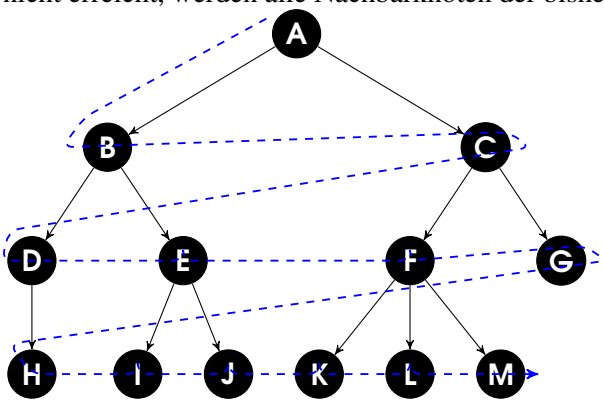


Fig3: Graphische Darstellung der Breadth-First Suche

Eine mögliche Implementierung der Breadth-First Suche in Prolog ist:

```
1  loesung(Start,Ziel):-
2      weg([Start],Ziel,[],Pfad),% Aufruf der Breitensuche,
3      write('Pfad:␣'),
4      write(Pfad).
5
6  % weg(Startknotenliste, Zielknoten, Liste der besuchten
7  %Knoten, Ergebnispfad)
8
9  weg(Startliste,Ziel,_,Pfad):-
10             Startliste=[Ziel|_],
11             reverse(Startliste,Pfad).
12
```

```
13 weg(Startliste,Ziel,Liste,Pfad):-
14          Startliste=[Start|_],
15          findall([Knoten|Startliste],
16             (kante(Start,Knoten),
17              not(member(Knoten,Startliste))),
18             Knotenliste),
19          append(Liste,Knotenliste,Listeneu),
20          Listeneu=[PfadN|RestPfad],
21          weg(PfadN,Ziel,RestPfad,Pfad).
```

Und in Haskell:

```
1  bfs goal succ start =
2    go [(k,[k]) | k <- start] -- Pfade erzeugen
3   where
4    go [] = Nothing -- nichts gefunden
5    go rs =
6      case filter (goal . fst) rs of -- ein Zielknoten enthalten?
7        -- Nein, mache weiter mit allen Nachfolgern
8        [] -> go [(k',k':p) | (k,p) <- rs, k' <- succ k]
9        -- Ja, dann stoppe:
10       (r:rs) -> Just r
```

Die Vor- und Nachteile der Breadth-First Suche lassen sich wie folgt zusammenfassen:

- Die Breadth-First Suche ist vollständig. Das bedeutet, wenn es eine Lösung gibt, wird sie mit Sicherheit gefunden.
- Der kürzeste Weg wird gefunden, dafür wird aber viel Speicherplatz benötigt.
- Es werden Knoten besucht, die für den optimalen Pfad nicht besucht werden müssten. Damit wird auch der Zeitaufwand erhöht. Dies beträgt nämlich $n + n * log(n)$, für n = Anzahl der Knoten.

6.3.3 Reverse Search

Bei der Rückwärtssuche wird von einem (bekannten) Zielknoten ausgegangen in die Richtung des Anfangsknoten.

Voraussetzungen für die Rückwärtssuche sind:

- Der Zielknoten kann explizit angeben werden (nicht nur eine Eigenschaft, die der Zielknoten erfüllen muss).
- Von jedem Knoten ist es möglich, die direkten Vorgänger zu berechnen.

Reverse Suche ist aber nur besser als Vorwärtssuche, wenn die Verzweigungsrate in Rückwärtsrichtung kleiner ist als die Verzweigungsrate in Vorwärtsrichtung. Das Problem an der Reverse Suche ist oft, dass die Vorgängerfunktion schwer zu finden ist.

6.4 Informierte Suche

Die Algorithmen, die bis jetzt betrachtet wurden, erzeugen neue Zustände und vergleichen sie immer wieder mit dem Zielzustand, bis sie übereinstimmen. Diese Strategie funktioniert, ist aber oft sehr kostenaufwendig. Wir betrachten im folgenden heuristischen Algorithmen.

Die Suche wird als „heuristisch“ oder „informiert“ bezeichnet, wenn es zusätzlich zu den Knoten und zu der Datenstruktur eine Schätzfunktion gibt, die als Schätzung des Abstands zum Zielknoten interpretieren werden kann. Der Zielknoten sollte ein Maximum bzw. Minimum der Schätzfunktion sein.

Eine Heuristik ist dann eine Methode, die oft ihren Zweck erreicht, aber nicht mit Sicherheit. Man spricht von heuristischer Suche, wenn die Schätzfunktion in vielen praktisch brauchbaren Fällen die richtige Richtung zu einem Ziel angibt, aber möglicherweise manchmal versagt.

Wir suchen dann das Maximum (oder Mininum) einer Knotenfunktion auf einem gerichteten Graphen.

6.4.1 Hill Climbing

Der Hill Climbing Suchalgorithm ist auch als Gradientenaufstieg bekannt, da die Suche immer in Richtung der Vergrößerung einer Funktion läuft.

Parameter, die notwendig für den Algorithmus sind:

- die Menge der initialen Knoten,
- die Nachfolgerfunktion,
- die Schätzfunktion,
- und der Zieltest.

Der Algorithmus verhält sich wie die Depth-First Suche, wobei der Nachfolger, der zu expandieren ist, nach der Schätzfunktion ausgesucht wird.

1: **function** HILL-CLIMBING(problem)
2: **returns** a solution state
Input:
problem ▷ Ein Problem
current ▷ Ein Knoten
next ▷ Ein Knoten
3: $current \leftarrow Make-Node(Initial-State[problem])$
4: **loop**
5: $next \leftarrow$ größter Wert aus *current* **if** $VALUE[next] < VALUE[current]$ **then**
return current
end
6: $current \leftarrow next$
7: **end loop**
8: **end function**

Algorithm 4: Hill Climbing Algorithm

Das Problem bei dem Hill Climbing Algorithmus ist, dass er bei lokalen Maxima abbricht. Es ist möglich, in dem Hill-Clinbing Algorithmus Zyklen einzubauen, damit Backtracking durchgeführt wird, aber er bleibt eine Zeit lang in den Maxima hängen.

Wenn mehrere Nachfolger die gleichen Schätzwerte haben, ist der Weg, der ausgesucht wird, nicht eindeutig.

Eine mögliche Implementierung der Hill Climbing Suche in Haskell ist:

```
1  hillclimbing cmp heuristic goal successor start =
2    let -- sortiere die Startknoten
3    list = map (\k -> (k,[k])) (sortByHeuristic start)
4    in go list []
5   where
6    go ((k,path):r) mem
7      | goal k
8        = Just (k,path) -- Zielknoten erreicht
9      | otherwise =
10       let -- Berechne die Nachfolger (nur neue Knoten)
11         nf = (successor k) \\ mem
12         -- Sortiere die Nachfolger entsprechend der Heuristik
13         l' = map (\k -> (k,k:path)) (sortByHeuristic nf)
14         in go (l' ++ r) (k:mem)
15   sortByHeuristic = sortBy (\a b -> cmp (heuristic a) (heuristic b))
16    -- cmp ist die compare-Funktion und ermoeglicht zu
17    -- maximieren oder zu minimieren
```

6.4.2 Best-First-Search

Die Best-First-Suche wählt immer den Knoten aus, der den besten Wert bzgl. der Schätzfunktion hat. Der Knoten wird dann überprüft und, wenn dieser nicht das Ziel ist, expandiert.

Eine Implementierung der Best-First Suche in Haskell ist:

```
1  bestFirstSearchMitSharing cmp heuristic goal successor start =
2    let -- sortiere die Startknoten
3      list = sortByHeuristic (map (\k -> (k,[k])) (start))
4    in go list []
```

```
1: function BEST-FIRST-SEARCH(problem,EVAL-FN)
2:     returns a solution sequence
Input:
    problem                                          ▷ Ein Problem
    Eval − Fn                                        ▷ Eine Evaluationsfunktion
3:     Queueing − Fn ← a function that orders nodes by EVAL-FN
4:     return GENERAL-SEARCH(problem, Queueing-Fn)
5: end function
```

Algorithm 5: Best-First-Search Algorithm

```
5  where
6   go ((k,path):r) mem
7     | goal k     = Just (k,path) -- Zielknoten erreicht
8     | otherwise =
9         let
10          -- Berechne die Nachfolger und nehme nur neue Knoten
11          nf = (successor k) \\ mem
12          -- aktualisiere Pfade
13          l' = map (\k -> (k,k:path)) nf
14          -- Sortiere alle Knoten nach der Heuristik
15          l'' = sortByHeuristic (l' ++ r)
16        in go l'' (k:mem)
17   sortByHeuristic =
18     sortBy (\(a,_) (b,_)-> cmp (heuristic a) (heuristic b))
```

Die Best-First-Suche ist mit der Depth-First Suche eng verbunden. Der Unterschied ist, dass bei der Best-First Suche der nächste Knoten, der überprüft und expandiert wird, mithilfe der Schätzfunktion ausgesucht wird. Dabei werden alle Knoten im Stack bewertet und dadurch können lokale Maxima schnell verlassen werden.

Wie bei der Depth-First Suche ist die Best-First Suche hier nicht vollständig und die Platzkosten steigen exponentiell in der Tiefe.

6.5 Fazit & Bewertung

Zunächst wurden das Suchproblem und der Begriff „Suche“ erläutert. Die nicht informierten Suchverfahren wurden präsentiert und es wurde gezeigt, dass ein General Search Algorithmus alle Probleme löst, wenn die richtige Strategie ausgesucht wird. Tiefensuche bedeutet, dass man beginnend im Startknoten so weit wie möglich entlang der bestehenden Kanten in die Tiefe geht, ehe man zurückläuft und dann in bislang unbesuchte Teilbäume absteigt. Breitensuche bedeutet, dass man beginnend im Startknoten alle direkt verbundenen Knoten besucht, bevor die nächst tiefere Ebene überprüft wird. Die Vollständigkeit und die Komplexität der gängigen uninformierten Suchalgorithmen wurden erläutert. Dabei ist klar geworden, dass die Zeit- und Speicherplatzkosten hoch sind.

Um die Kosten zu minimieren werden die bekannte Suchverfahren weiterentwickelt. Daraus folgte zum Beispiel die Best-First Suche. Diese ist eine General Search, wobei nur die minimale Anzahl der Knoten berücksichtigt wird.

Solche Algorithmen, wobei die Kosten minimiert werden, nennen wir **greedy Algorithmen**. Der Algorithmus ist aber immer noch nicht optimal.

6.6 Quellen und Literatur

- Stuart J. Russel and Peter Norvig, “Artificial Intelligence - A Modern Approach”, Pretince Hall, Englewood Cliffs, New Jersey 2010
- Ivan Bratko, “Prolog Programming for Artificial Intelligence”, Addison-Wesley Yugoslavia 1990
- R. Schenk and R.P. Abelson, “Scripts, Plans and Knowledge”, International Joint Conference on Artificial Intelligence, 1975
- F.C. Pereira and S.M.Schieber, “Prolog and Natural-Language Analysis”, CSLI, 1987
- F.M. Donini, M. Lenzerini and others, “The complexity of concept languages”, Inf. Comput., 1997
- I. Wegener, “Highlights aus der Informatik”, Springer, Berlin, 1996

- MIT Open Course - Artificial Intelligence http://ocw.mit.edu/courses/electrical-engineering-and-compu 6-034-artificial-intelligence-fall-2010/, Zugriff 28.04.2016

7. Informierte Suche

7.1 Schwächen uninformierter Suche

Im folgenden zeigen wir noch einmal die, bereits bekannte, uninformierte Suche, um die Vorteile von informierter Suche besser darstellen zu können. Wir beginnen mit folgendem Graphen (es könnte eine Abstraktion einer Straßenkarte sein).

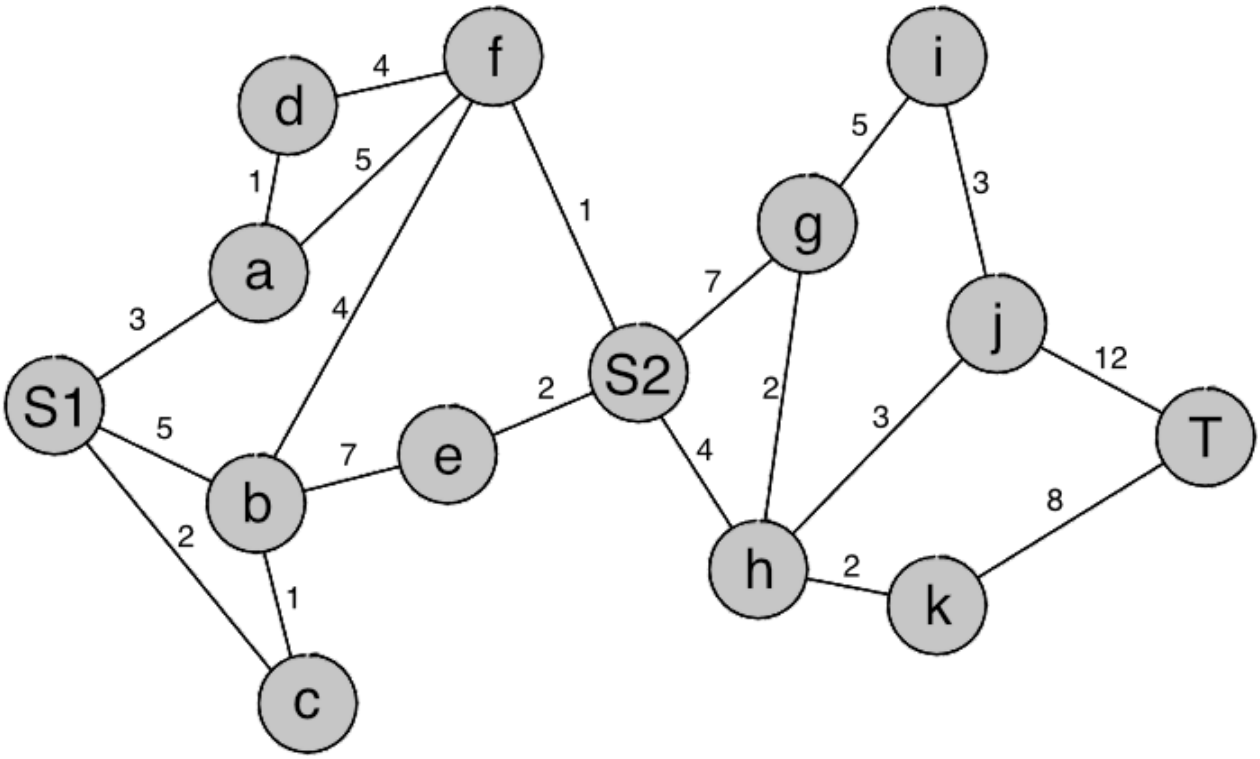


Abbildung 7.1: Beispiel Graph

Wir sehen unsere Startknoten $S1$ und $S2$, sowie den Zielknoten T. Außerdem sind alle Kanten zwischen den Knoten mit einer Länge versehen. Im folgenden werden wir mit Hilfe der Breitensuche erst den Weg von $S1$ zu T und dann den Weg von $S2$ zu T suchen. In der folgenden Grafik sind die Zwischenknoten mit Kleinbuchstaben beschriftet, des weiteren sind Start und Ziel in gelb eingefärbt. In jedem Schritt werden die aktuell zu untersuchenden

Knoten grün und die bereits besuchten Knoten blau markiert.

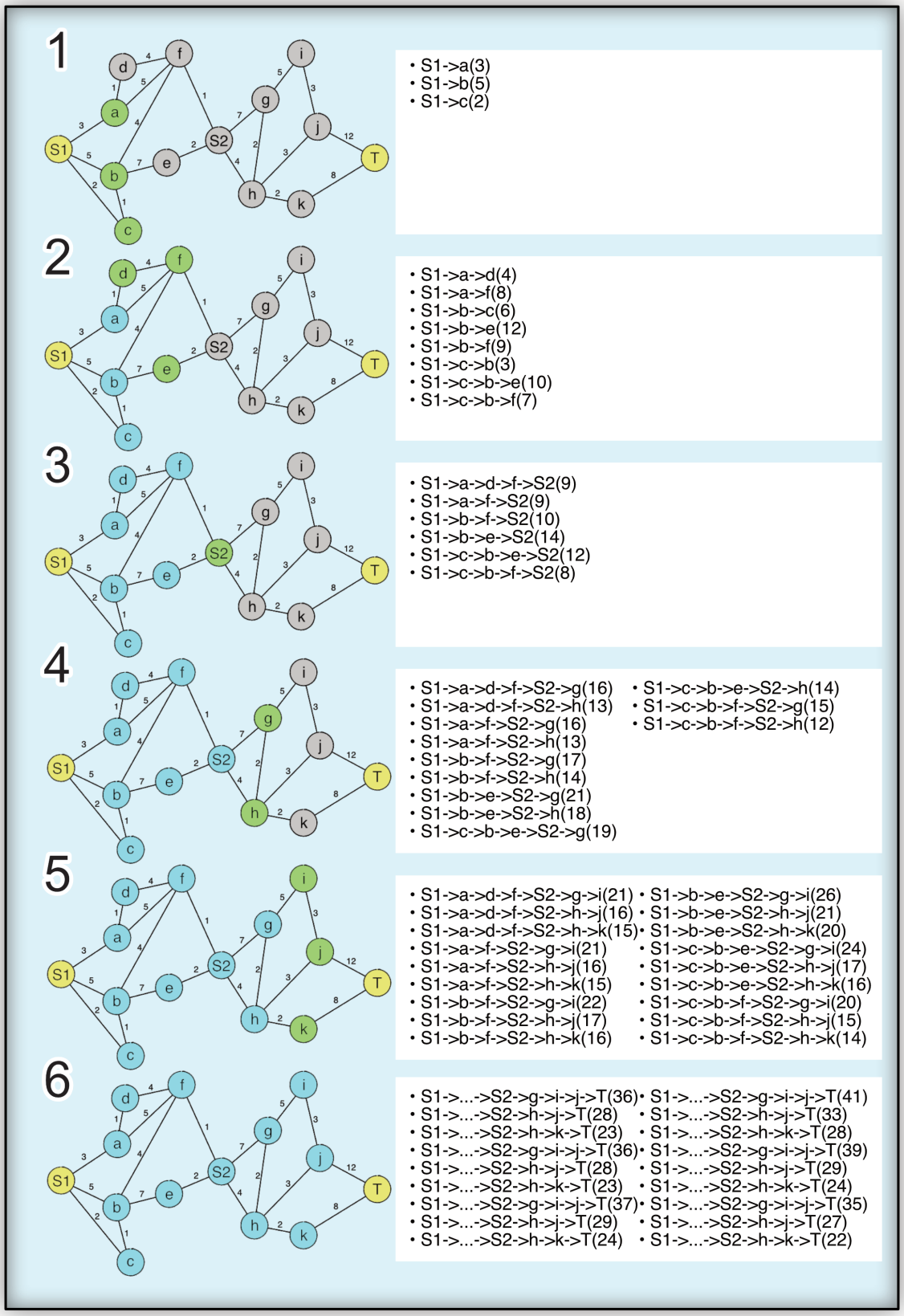


Abbildung 7.2: Beispiel Graph mit Breitensuche. Zu jedem Graphen sind die besuchten Wege angegeben. Die durch "..." zusammengefassten Wegstücke der Wege des letzten Graphen entsprechen denen der Wegen von $S1$ nach $S2$ des vorletzten Graphens.

1. Wir besuchen zuerst Knoten a, b und c und stellen offensichtlich fest, dass wir T noch nicht erreicht haben.
2. Wir besuchen nun d, e und f. Wir haben T immer noch nicht erreicht.
3. In diesem Schritt gibt es nur $S2$ zu besuchen.
4. Ab hier Suchen wir effektiv den kürzesten Weg von $S2$ zu T, weshalb dieses Beispiel später nicht mehr extra erläutert wird. Hier besuchen wir die Knoten g und h.
5. Dann besuchen wir noch i, j und k.
6. Jetzt bilden wir noch die letzen Wege und diese enden beim Knoten T. Also sind wir fertig.

Der kürzeste Weg von $S1$ zu T ist also:

- $S1 \rightarrow c \rightarrow b \rightarrow f \rightarrow S2 \rightarrow h \rightarrow k \rightarrow T$

Und daraus folgt der kürzeste Weg von S2 zu T ist:

- $S2 \rightarrow h \rightarrow k \rightarrow T$

Fazit:

Wie man sieht ist das Problem gut lösbar mit uninformierte Suche, aber es fällt sofort auf das wir alle Knoten besuchen mussten und alle Wege bilden mussten. Dies hat nicht nur hohe Speicherkosten zur Folge, sondern auch eine hohe Laufzeit. Bereits bei diesen simplen Beispiel sieht man die enorme Menge von Wegen die man speichern muss. Als Mensch würde man bei dieser Suche weniger Wege betrachten:

- Wäre es nicht einfacher Wege auszulassen die offensichtlich zu lang sind?
- Oder wäre es nicht besser Wege wie $S1 \rightarrow b$ durch den kürzeren $S1 \rightarrow c \rightarrow b$ zu ersetzen?

Unterbewusst benutzt man eine Menge Vereinfachungen die der Computer vielleicht auch anwenden sollte. Im folgenden Kapitel werden wir versuchen diese Suche intuitiv zu optimieren. /newpage

7.2 Verbesserungen

7.2.1 Intuitiver Algorithmus

Statt alle Knoten zu besuchen wie bisher, werden wir im folgenden unser Wissen über den Graphen nutzen um schneller zum Ergebnis zu kommen. Folgende Regeln werden angewandt:

1. Wir besuchen immer den Knoten mit der niedrigsten Kantenlänge zuerst.
2. Wenn wir einen neuen Weg zu einem Knoten finden und dieser kürzer ist als der vorherige, brauchen wir nur noch diesen neuen Weg zu betrachten. Ansonsten kann der neue Weg ignoriert werden.
3. Wenn wir zwei Wege zur Auswahl haben, die gleichlang sind, wählen wir zufällig.
4. Wenn ein Weg keine nicht besuchten Kinder mehr hat gehen wir zurück zum letzten Knoten der noch offene Kinder hatte.
5. Aufgrund der oben genannten Regeln muss der erste Weg zum Ziel der kürzeste sein. (Um das Beispiel nicht unnötig lang zu machen, wird dies an dieser Stelle nicht bewiesen)

7.2.2 Anwendung

Wir beginnen mit dem Graphen aus Kapitel 1. Die Knoten werden genau wie bereits bekannt eingefärbt werden.

1. Wir beginnen mit Knoten c da mit einer Länge von 2 seine Kante am kürzesten ist $(Regel1)$. Wir haben also den Weg:
   - $S1 \rightarrow c(2)$
2. Nun besuchen wir b und erhalten den Weg:
   - $S1 \rightarrow c \rightarrow b(3)$
3. Jetzt müssen wir erst a betrachten, da die Kante mit einer Länge von 3 kürzer ist als $b \rightarrow f$ mit 4. Wir haben also die Wege:
   - $S1 \rightarrow c \rightarrow b(3)$
   - $S1 \rightarrow a(3)$
4. Als nächstes besuchen wir d, da $a \rightarrow d$ am kürzesten ist. Es werden die Wege:
   - $S1 \rightarrow c \rightarrow b(3)$
   - $S1 \rightarrow a \rightarrow d(4)$

   gespeichert.
5. Wir mussten uns zwischen $d \rightarrow f$ und $b \rightarrow f$ entscheiden und haben zufällig $d \rightarrow f$ gewählt $(Regel3)$. Damit ergeben sich die Wege:
   - $S1 \rightarrow c \rightarrow b(3)$

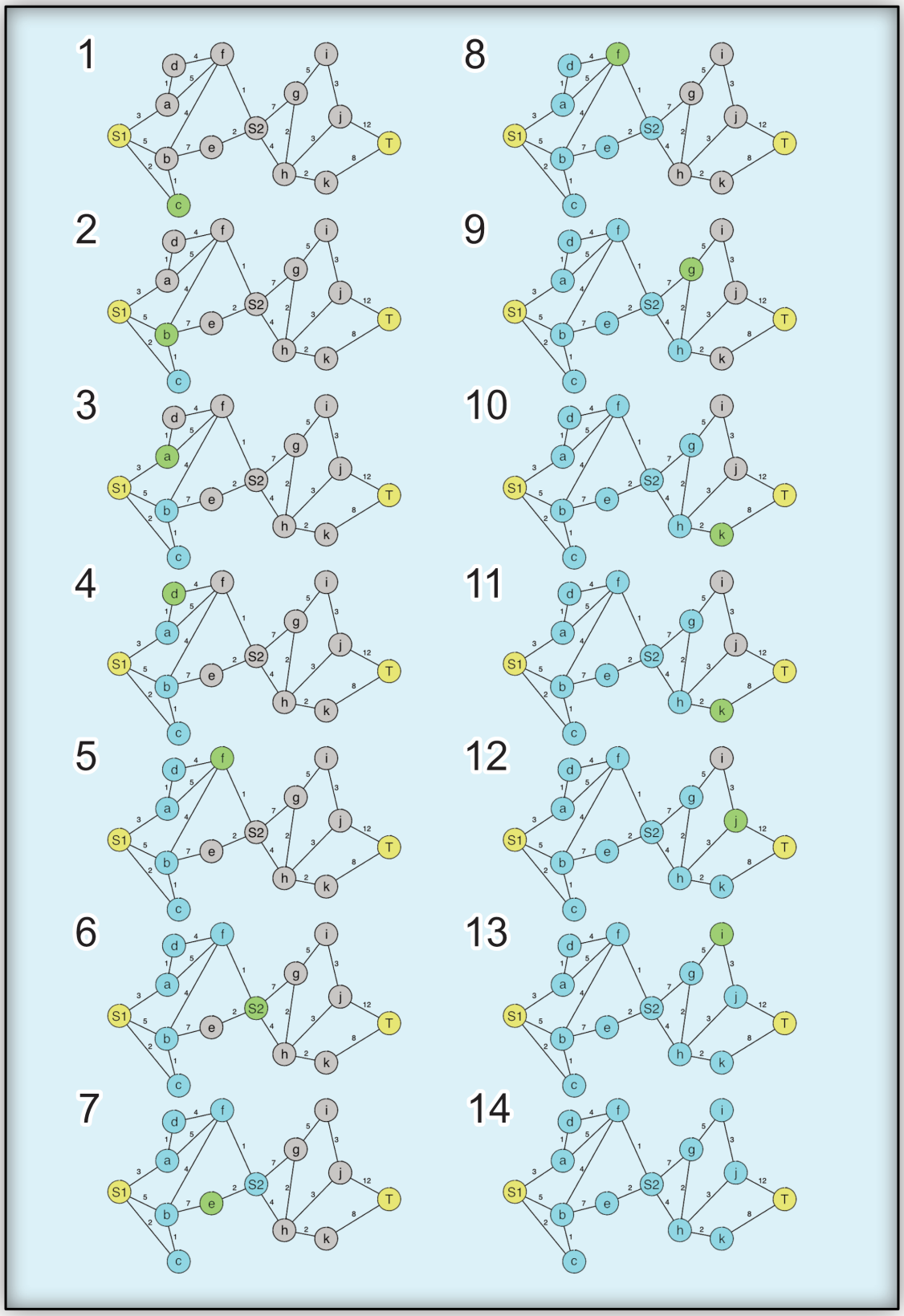


Abbildung 7.3: Beispiel Graph mit intuitiven Algorithmus.

- $S1 \rightarrow a \rightarrow d \rightarrow f(8)$

6. Im nächsten Schritt wählen wir $f \rightarrow S2$. Die Wege sind:

- $S1 \to c \to b(3)$
- $S1 \to a \to d \to f \to S2(9)$

7. Jetzt besuchen wir e. Wir haben also die Wege:
   - $S1 \to c \to b(3)$
   - $S1 \to a \to d \to f \to S2 \to e(11)$

   Stellen aber fest, dass e für den nächsten Schritt keine unbesuchten Kinder hat. Also gehen wir zurück zu $S2$ da dieser noch Kinder hatte $(Regel4)$. Die Wege werden wieder zu:
   - $S1 \to c \to b(3)$
   - $S1 \to a \to d \to f \to S2(9)$

   Wir besuchen noch einmal f, dieses Mal als $b \to f$ (zufällig gewählt aus $b \to f$ und $S2 \to h$). Damit haben wir einen neuen Weg zu f, der kürzer ist als $S1 \to a \to d \to f$. Wir ersetzen ihn $(Regel2)$. Damit ergibt sich der Weg:
   - $S1 \to c \to b \to f \to S2(8)$
8. Wir wählen h und erhalten den Weg:
   - $S1 \to c \to b \to f \to S2 \to h(12)$
9. Aus $h \to g$ und $h \to k$ wählen wir g. Der Weg ist:
   - $S1 \to c \to b \to f \to S2 \to h \to g(14)$
10. Wir besuchen k. Damit ergeben sich die Wege:
    - $S1 \to c \to b \to f \to S2 \to h \to g(14)$
    - $S1 \to c \to b \to f \to S2 \to h \to k(14)$
11. Wir wählen j und erhalten die Wege:
    - $S1 \to c \to b \to f \to S2 \to h \to g(14)$
    - $S1 \to c \to b \to f \to S2 \to h \to k(14)$
    - $S1 \to c \to b \to f \to S2 \to h \to j(15)$
12. Jetzt besuchen wir erst $j \to i$.
13. Dann besuchen wir sofort $g \to i$, aber wir haben über j bereits einen kürzeren Weg zu i und können den neuen ignorieren. Es folgen die Wege:
    - $S1 \to c \to b \to f \to S2 \to h \to k(14)$
    - $S1 \to c \to b \to f \to S2 \to h \to j \to i(18)$

    Und wir sehen, dass i keine Kinder mehr hat:
    - $S1 \to c \to b \to f \to S2 \to h \to k(14)$
    - $S1 \to c \to b \to f \to S2 \to h \to j(15)$
14. Als letztes nehmen wir noch $k \to T$. Wir bekommen also den Weg:
    - $S1 \to c \to b \to f \to S2 \to h \to k \to T(22)$

    und aus $Regel5$ folgt, dass dies der kürzeste Weg sein muss.

Fazit:

Zuallererst unser Ergebnis ist identisch mit dem aus Kapitel 1, wir haben also ebenfalls den besten Weg gefunden. Es es leicht sichtbar, dass unser intuitiver Algorithmus nicht perfekt ist. Wir mussten trotzdem alle Knoten besuchen, genau wie die Breitensuche. Aber man sieht ebenfalls sofort, dass wir deutlich weniger Wege betrachten und speichern mussten als zuvor. Und besonders hervorzuheben ist, dass wir die Suche beenden konnten als wir den ersten Weg gefunden hatten. Das wirkt sich natürlich positiv auf Laufzeit und Speicherverbrauch aus. In den folgenden Kapiteln wollen wir die hier intuitiv erarbeiteten Konzepte formal definieren, die echten Algorithmen der informierten Suche vorstellen und deren Eigenschaften überprüfen.

7.3 Einleitung

Statt blind alle Möglichkeiten zu probieren um die Beste zu erhalten, zieht man, wie ein Mensch es intuitiv machen würde, weitere Informationen über das Problem zurate (Heuristiken) und ignoriert Lösungen die offensichtlich schlechter sind als bereits gefundene (Branch and Bound). Damit ergibt sich eine deutlich bessere Laufzeit, wie sie in Echtzeitanwendungen (z.B. in Spielen) nötig ist. Natürlich stellt sich auch die Frage, ob diese einfacher erhaltene Lösung immer noch die optimale Lösung ist (Optimalität). Im folgenden werden die Verschiedenen Arten (Best-First Search und A*) informierter Suche von und deren Vor- und Nachteile erklärt.

7.4 Definitionen

7.4.1 Branch and Bound

Branch and Bound (häufig abgekürzt mit B&B[1]) ist eine Optimierungsstrategie für Algorithmen, hier im speziellen Suchverfahren. Um nicht alle Knoten eines Baumes durchsuchen zu müssen, kann man, wie bereits intuitiv oben gesehen, die Aufgabe in Teile aufteilen ("Branch" engl. "to branch"verzweigen[2]) und diese Teile nach Relevanz für die Lösung einteilen ("Bound engl. "to bound"beschränken[3]). Damit man dies bei Suchalgorithmen bewerkstelligen kann benutzt man eine Abschätzung, intuitiv war dies "kürzeste Luftlinie zum Ziel" auch Heuristik genannt.[4]

7.4.2 Extended List

Bei der Anwendung von Branch and Bound kann es dazu kommen, dass Wege p'=(. . . , A) zu einen Knoten A weiter verfolgt werden, für die es einen Weg p=(. . . , A) gibt, so dass gilt:

$$Wf(p) < Wf(p')$$

Dies führt dazu, dass der Suchbaum unnötig großwird. Um zu erreichen, dass nicht die längeren Wege weiter verfolgt werden, wird die Extended List eingeführt. In der Extended List wird für jeden Knoten zum Zeitpunkt an dem dieser besucht wird, die Knoten Identifikation und die Weglänge zu diesem gespeichert. Führt nun im Verlauf der Suche ein Weg zu einem Knoten, der bereits in der Extended List gespeichert ist, so wird die gespeicherte mit der Aktuellen Weglänge zum Knoten verglichen. Ist die Aktuelle Weglänge größer, muss der Weg nicht weiter verfolgt werden, es gibt bereits einen kürzeren Weg zu dem Knoten. Ist sie geringer, wird der neue Wert eingetragen und der ältere Weg über diesen Knoten muss nicht weiter betrachtet werden.[5] Mit Branch and Bound + Extended List ergibt sich in unserem Beispiel Graphen:

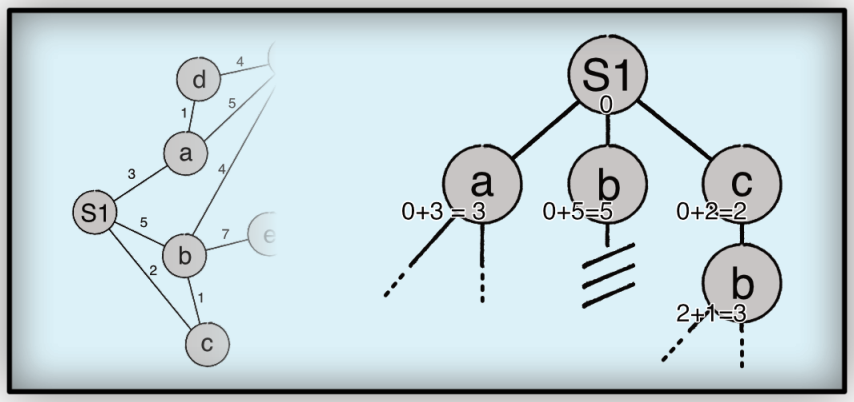


Abbildung 7.4: Links ist der linke Teil des Beispiel Graphen abgebildet, rechts ein Teil des Suchbaums von Branch and Bound + Extended List. Wie abgebildet wird der Weg (S1, b, . . .) nicht weiter verfolgt, da, $Wf((S1,c,b)) < Wf(S1,b)$.

[1] Branch and Bound Algorithms - Principles and Examples. Jens Clausen, March 12, 1999

[2] www.dict.cc

[3] www.dict.cc

[4] Anwendungen von Branch and Bound, Frederik Wollny und Philipp Schmid, Hochschule Aalen, 2016

[5] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-5-search-optimal-branch-and-bound-a/

7.5 Best-First Search

Best-First Search ist eine Klasse von informierten Such-Algorithmen, welche nachdem Prinzip arbeiten, dass sie stets denjenigen Knoten zuerst untersuchen, welcher für den Erfolg der Suche am "vielversprechendsten" erscheint. Die Wahl dieses "vielversprechenden" Knotens geschieht anhand einer gewissen Heuristik. Der wohl bekannteste Algorithmus aus diesem Bereich ist A*, welcher auch als eine Erweiterung von Branch and Bound gesehen werden kann. An ihm wollen wir im Folgenden das Prinzip von Best-First Search verdeutlichen.

7.6 Heuristik

Die Heuristik ist im Falle der Informierten Suche die Representation der menschlichen Lösungsfindung im Such-Algorithmus. Sie weist jedem Knoten einen geschätzten Wert zu, der darüber Auskunft gibt, ob es sich lohnen könnte, einen Weg über diesen Knoten einzuschlagen. Bei der Findung des kürzesten Weges gilt, desto höher der durch die Heuristik bestimmte Wert für einen Knoten ist, desto unwahrscheinlicher ist es, dass der Kürzeste Weg über diesen Knoten verläuft. Aufgrund der Vielfalt von Gegebenheiten unter denen eine Suche stattfinden kann, ist es offensichtlich, dass eine Heuristik die ein Problem Löst nicht ohne weiteres auf andere Probleme übertragen werden kann. Deswegen werden zunächst einige wichtige Begrifflichkeiten geklärt, die beim Umgang mit Heuristiken helfen können.[6]

7.6.1 Eigenschaften von Heuristiken

Im folgenden werden einige Eigenschaften definiert, die eine Heuristikfunktion erfüllen kann. Diese Eigenschaften können dabei helfen ein Urteil, über die Zulässigkeit einer Heuristik für eine Problemstellung, zu fällen.

Definition. Eine heuristische Funktion $h(n)$ wird als admissible (zulässig) bezeichnet, wenn für alle Knoten n erfüllt ist:

$$h(m) \leq k(m,n)$$

mit $k(m,n)$ als kürzeste Weglänge von m nach n und für alle Knoten $t \in G$, die die Endbedingung erfüllen, gilt:

$$h(t) = 0.$$[7]

Admissible Heuristiken reichen bereits aus, um auf Landkarten o.ä. nach der kürzesten Route zu suchen. Damit eine Heuristik auch für Suchen verwendet werden kann die nicht auf geografischer Ebene stattfinden, werden die Eigenschaften konsistent und monoton definiert. Die Definitionen wurden aus [Kai13] übernommen.

Definition. Eine heuristische Funktion $h(n)$ ist konsistent, wenn für alle Knotenpaare $m,n \in G$ folgendes erfüllt ist:

$$h(m) \leq k(m,n) + h(n)$$

mit $k(m,n)$ als kürzeste Weglänge von m nach n und für alle Knoten $t \in G$, die die Endbedingung erfüllen, gilt:

$$h(t) = 0.$$

Definition. Eine heuristische Funktion $h(n)$ ist monoton, wenn für alle Knotenpaare $m,n \in G$ mit n als unmittelbarem Nachbar von m folgendes erfüllt ist:

$$h(m) \leq c(m,n) + h(n)$$

mit $c(m,n)$ als Kosten der Kante von m nach n; und für alle Knoten $t \in G$, die die Endbedingung erfüllen, gilt:

$$h(t) = 0.$$

Theorem Die Eigenschaften konsistent und monoton sind äquivalent.

Theorem Für jede konsistente Funktion h gilt: $h(n) \leq h*(n)$ für alle Knoten $n \in G$.

Um den unterschied zwischen einer Heuristik die admissible ist und einer die konsistent ist zu verdeutlichen, sei ein Beispiel angeführt:

[6] http://www.duden.de/rechtschreibung/Heuristik

[7] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-5-search-optimal-branch-and-bound-a/

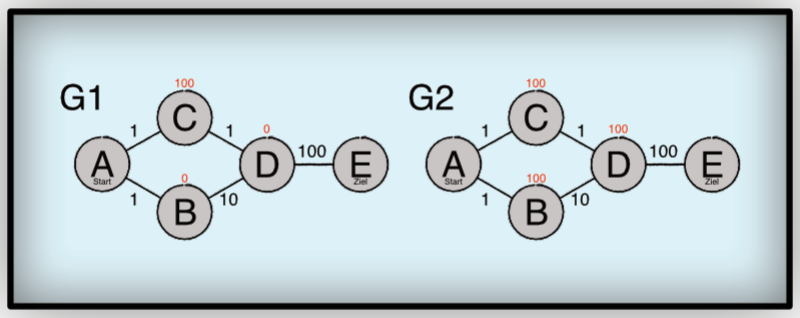


Abbildung 7.5: Die Abbildungen G1 und G2 beschreiben das selbe Problem mit zwei verschiedenen Heuristiken. Ziel ist es den kürzesten Weg von Knoten A nach Knoten E zu finden. G1 stellt die Verwendung einer admissible Heuristik und G2 die einer konsistenten Heuristik dar. Die roten Zahlen geben die Werte der Heuristikfunktionen an den relevanten Knoten an. Wird nun eine Suche, mit Branch and Bound + Extended List + Heuristik durchgeführt, liefert die Suche für G1 den Weg(A,B,D,E) und für G2 den Weg(A,C,D,E). Entscheidend für diese Lösung ist, dass bei G1 zunächst A,B, und D betrachtet werden und somit aufgrund der Extended List der Weg (A,C,D,E) wegfällt. Im Gegensatz dazu beginnt die Suche bei G2 mit einem Weg über C, da die geschätzten Kosten von C mit 101 kleiner als die von B sind.

7.6.2 Verschiedene Heuristiken

In diesem Abschnitt werden drei Heuristiken vorgestellt, angefangen mit der Hamming-Distanz. Aufgrund dessen, dass in einigen Kapiteln Bezug auf die folgenden Heuristiken genommen wird, wurden einige Bezeichnungen in den Definitionen entsprechend angepasst.
Die Hamming-Distanz hh(m,n) beschreibt wie großder Unterschied zwischen den Zeichenketten m und n ist und ist wie folgt definiert:

Definition. $\sum$ sei ein endliches Alphabet sowie $m = (m_1, \ldots, m_k)$ und $y = (n_1, \ldots, n_k)$ zwei k Zeichen lange Worte aus $\sum^k$. Die Hamming-Distanz zwischen m und n ist definiert als:

$$h_h(m,n) := |j \in \{1, \ldots, k\} | m_j \neq n_j|.$$

Eine Heuristik wäre z.B. $h(n) = h_h(n,x)$, mit x als Ziel-Konstellation der Suche.[8]

Definition. Die Manhattan Distanz $h_m(m,n)$ für die Punkte n und m beschreibt die Distanz, die zustande kommt, wenn nur horizontale und vertikale Wege genommen werden, um n von m aus zu erreichen. Die Manhattan-Distanz ist wie folgt definiert:

$$h_m(m,n) = \sum_i |m_i - n_i|.$$

Eine Heuristik wäre z.B $h(n) = h_m(n,x)$, mit x als Ziel-Knoten der Suche.[9]

Definition. Der Euklidischer Abstand $h_e(m,n)$ von m und n entspricht der "Luftlinießwischen diesen. Der Euklidische Abstand für zwei Vektoren im k-Dimensionalen Raum ist wie folgt definiert:

$$h_e(m,n) = \sqrt{((m_1 - n_1)^2 + \cdots + (m_k - n_k)^2)}.$$

Eine Heuristik wäre z.B $h(n) = h_e(n,x)$, mit x als Ziel-Knoten der Suche.[10]

[8] http://sb.fluomedia.org/hamming/, https://de.wikipedia.org/wiki/Hamming-Abstand
[9] http://mathworld.wolfram.com/TaxicabMetric.html
[10] http://mathworld.wolfram.com/EuclideanMetric.html

7.6.3 Findung von Bewertungsfunktionen

Die Frage Lautet:"Wie findet sich die richtige Heuristikfunktion für die gegebene Problemstellung". Um einen Lösungsansatz für eine allgemeine Vorgehensweise zu finden, wird zunächst das Beispiel aus 5.1 betrachtet. Die Aufgabe im Beispiel ist es, den kürzesten Weg von dem Knoten S1 bzw. S2 zum Zielknoten T im Graphen zu finden. Die Problemstellung lässt sich in eine Konjunktion von zwei Bedingungen aufteilen:

1. Die zurückgelegte Strecke von S1 bzw. S2 nach T soll minimal sein.
2. Die Strecke entspricht einem Weg (Graphentheorie).

Würde die zweite Bedingung weggelassen, würde immer eine Strecke entsprechend der Luftlinie von $S1$ bzw. $S2$ nach T gewählt werden. Es liegt also nahe $h(n) = h_e(n,x)$ zu wählen, um einem Knoten nach seiner Distanz zu T zu bewerten. Wird Bedingung zwei nun hinzugezogen, werden mit der gewählten Heuristik immer die erreichbaren Knoten bevorzugt, die am nächsten zum Ziel liegen. So ist eine Akzeptable Heuristik für die Problemstellung gefunden. Dieses Prinzip des Konstruieren von vereinfachten Modellen durch systematisches Weglassen von Bedingungen der Konjunktion der Problemstellung scheint auch allgemein eine gute Herangehensweise zu sein, um leicht heuristische Schätzungen zu ermitteln.

7.7 A* Algorithmus

A* beschreibt eine Gruppe von Algorithmen, die auf alle Kniffe der Informierten Suche, die bis jetzt vorgestellt wurden, zurückgreifen:[11]

- Branch and Bound
- Extended List
- Heuristik

Für ein gegebenes Problem gibt es innerhalb dieser Gruppe bessere und schlechtere Algorithmen. Um zwei A* Algorithmen zu vergleichen hilft das folgende Theorem:

Theorem A*1 und A*2 seien zwei Versionen von A* mit $h1(n)?h*(n)$ und $h2(n)?h*(n)$ für alle Knoten $n \in G$ und mit $h1(n)?h2(n)$ für alle Knoten $n \in G\,t|terflltdieEndbedingung$. (A*2 sei "besser informiertäls A*1.) Falls es eine Lösung gibt, so wird bis zur Terminierung jeder von A*2 expandierte Knoten auch von A*1 expandiert, (A*2 "dominiertÄ*1.)[12]

In den folgenden Abschnitten wird auf weitere Eigenschaften des A* eingegangen und anschließend wird der A* durch Beispiele veranschaulicht.

7.7.1 Zulässigkeit

Welche Bedingungen muss eine Heuristik erfüllen, damit unser Such-Verfahren ordnungsgemäßfunktioniert? Dazu sollten wir zuerst definieren, wie wir uns ein korrekte Funktionsweise vorstellen.

- Zulässigkeit Ein Such-Verfahren heißt zulässig, falls es stets mit einer optimalen Lösung terminiert, falls diese existiert.

Da bei unserem heuristischen Such-Verfahren A* die Wahl der Heuristik bisher nicht klar eingeschränkt ist, liegt es nahe, dass in dieser Wahl ein entscheidender Punkt für die Zulässigkeit unseres Algorithmus liegt. Man kann sich leicht Heuristiken überlegen, welche die Funktionsweise von A* stören und dafür sorgen, dass der Algorithmus keine optimale Lösung findet oder womöglich gar nicht erst terminiert. Wir sollten uns also genauer mit der Beschaffenheit von Heuristikfunktionen auseinandersetzen. Eine erste wichtige Eigenschaft einer Heuristik h zur Schätzung von h* ist, ob diese optimistisch schätzt.

- optimistische Schätzung Eine Schätzung h von h* ist genau dann eine optimistische Schätzung, falls sie die Kosten eines optimalen Pfades nie überschätzt.

Das heisst:

- $h(n) \leq h*(n)$ für alle Knoten $n \in GP$

Diese Eigenschaft liefert uns nun die ausschlaggebende Bedingung für die Zulässigkeit von A*. Zulässigkeit von A*. Falls die verwendete Heuristik h eine optimistische Schätzung von h* ist, dann ist A* zulässig. Der Beweis dieses Satzes und einige genauere Erörterungen können in [Kai13] nachgelesen werden.

[11] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-5-search-optimal-branch-and-bound-a/

[12] [Kai13]

7.7.2 Komplexität

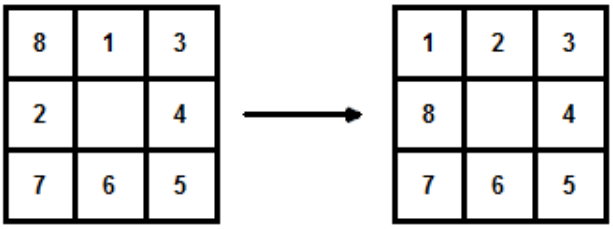
Es lässt sich leicht erkennen, dass A* in der Regel eine weit bessere Komplexität hat, als blinde Such-Verfahren. Die Anzahl der expandierten Knoten ist bei A* oft sogar um ein Vielfaches geringer. Sei nun die Länge einer optimalen Lösung mit d bezeichnet. Des Weiteren kann man von Einheitskosten ausgehen und die Existenz genau einer optimalen Lösung voraussetzen. Je nachdem wie optimistisch nun der Fehler von h bezüglich h* angenommen wird, kommt man auf eine quadratische oder sogar lineare Komplexität bezüglich d. Im eher realistischen Fall lässt sich jedoch nur auf eine exponentielle Komplexität schliessen.

7.7.3 Optimalität

Nun interessieren wir uns für die Optimalität von A* im Hinblick auf die Anzahl der expandierten Knoten. Der Begriff Optimalität beruht hier auf dem der Dominanz. Das bedeutet, ein Verfahren ist optimal gegenüber einer Klasse von Verfahren, wenn es alle Elemente aus dieser Klasse dominiert. Für den ganz allgemeinen Fall lässt sich leicht erkennen, dass A* nicht das optimale Verfahren überhaupt ist. Es gibt in bestimmten Fällen offensichtlich Verfahren, welche nicht alle von A* untersuchten Knoten ebenfalls untersuchen. Gerade bei speziell auf eine Klasse von Graphen abgestimmten Verfahren ist dies der Fall. Jedoch ist unter bestimmten Einschränkungen eine konkretere Aussage bezüglich der Optimalität von A* möglich. Dafür schränkt man die Bedingungen auf eine Domäne ein, welche eine konsistente Heuristik h besitzt, und setzt voraus, dass es mindestens einen optimalen Lösungspfad gibt, bei welchem $h < h^*$ für alle Knoten außer dem Zielknoten gilt. Dann gilt, dass A* dominant für alle Problem-Instanzen ist, und zwar für alle möglichen Entscheidungsregeln bei Nicht-Eindeutigkeit des Knotens mit minimalem f-Wert. Da dem Algorithmus A* bei konsistentem h zusätzlich eine sehr gute Verwaltung des Suchbaumes möglich ist, lässt sich sagen, dass A* unter diesen Voraussetzungen sehr effektiv ist.

7.7.4 Beispiel des 8-Puzzles

Beim 8-Puzzle handelt es sich um ein Spiel, bestehend aus einem Spielbrett der Größe 3-mal-3 und acht Kacheln, nummeriert von 1 bis 8, welche auf dem Spielbrett verteilt sind. Ein Feld des Spielbrettes bleibt also frei. Grenzt eine Kachel an dieses freie Feld, so kann sie an dessen Stelle verschoben werden. Das Ziel des Spieles ist, von einer ungeordneten Stellung der Kacheln durch eine Folge von Verschiebungen zu einer Zielstellung zu gelangen. Die folgende Abbildung zeigt ein konkretes Problem mittels Start- und Zielkonfiguration.



Die Problemstellung ist, zu einer gegebenen Kachelkonstellation die kürzeste Abfolge von Verschiebungen zu finden, die nötig ist, um zur Zielkonfiguration zu gelangen.

Wird das Problem abstrakt betrachtet, ist es möglich einen Graphen zu konstruieren, auf den eine informierte Suche angewendet werden kann. Die Knoten des Graphen repräsentieren die Konstellationen die das Kachelfelde annehmen kann und zwei Knoten sind genau dann über eine Kante miteinander verbunden, wenn sich die Kachelkonstellationen der Knoten um genau 1 Verschiebung unterscheidet. Als Kantenbewertung $f : E \to \Re$ wird die Anzahl der Verschiebungen gewählt die nötig ist, um die Kachelkonstellation des Zielknotens zu erhalten. In dem konstruierten Graphen ist die länge jeder Kante 1.

Auf den so erhaltenen Graphen lässt sich nun eine informierte Suche mit einem A* Algorithmus durchführen. Um die Heuristik des A* leichter zu bestimmen, wird die Problemstellung in eine Konjunktion aus den folgenden Bedingungen zerlegt:

1. Die Anzahl der Verschiebungen soll minimal sein.
2. Es dürfen nur Kacheln auf benachbarte Felder verschoben werden.
3. Das benachbarte Feld muss leer sein.

Für Bedingung 1 bietet sich die Hamming-Distanz an. Diese gibt für jede Kachelkonstellation die Anzahl der falsch positionierten Felder an. Wird nun die 2 Bedingung hinzugezogen, muss berücksichtigt werden, dass keine

diagonalen Züge möglich sind. Durch diese Verschärfung ist die Manhattan-Distanz eine passendere Heuristik. Die Manhattan-Distanz berechnet für eine gegebene Kachelkonstellation die Anzahl der horizontalen und vertikalen Verschiebungen, die nötig sind, um die Zielkonstellation zu erhalten. Wird zuletzt noch die 3 Bedingung hinzugezogen kann bei der Manhattan Distanz verblieben werden. Nichts desto trotz können beide Heuristiken für das Problem verwendet werden und dies wird auch getan. Im folgenden bezeichnet die Heuristikfunktion...

- $h_1(n)$, die Anzahl falsch positionierter Kacheln in der durch Knoten n dargestellten Konfiguration (Hamming-Distanz)
- $h_2(n)$, die Summe der Distanzen der falsch positionierten Kacheln zu ihrer Zielposition (Manhattan-Distanz)

Der Ablauf des Algorithmus unter Verwendung von $h_1(n)$ bzw. $h_2(n)$ wird durch die Suchbäume G_{S1} bzw. G_{S2} dargestellt. g bezeichnet die zurückgelegte Wegstrecke, h den Wert von $h_1(n)$ bzw. $h_2(n)$ für die gegebene Kachelkonstellation und f die Summe von h und g.

Wie an den folgenden beiden Suchbäumen zu sehen, benötigt der A* Algorithmus unter Verwendung von $h_2(n)$ eine geringere Anzahl von Expansionen als bei der Verwendung von $h_1(n)$). Der Aufwand ist also geringer. Die Wahl des zu expandierenden Knotens ist stets eindeutig. Heuristik $h_2(n)$ scheint also ein besserer Schätzer zu sein als $h_1(n)$.

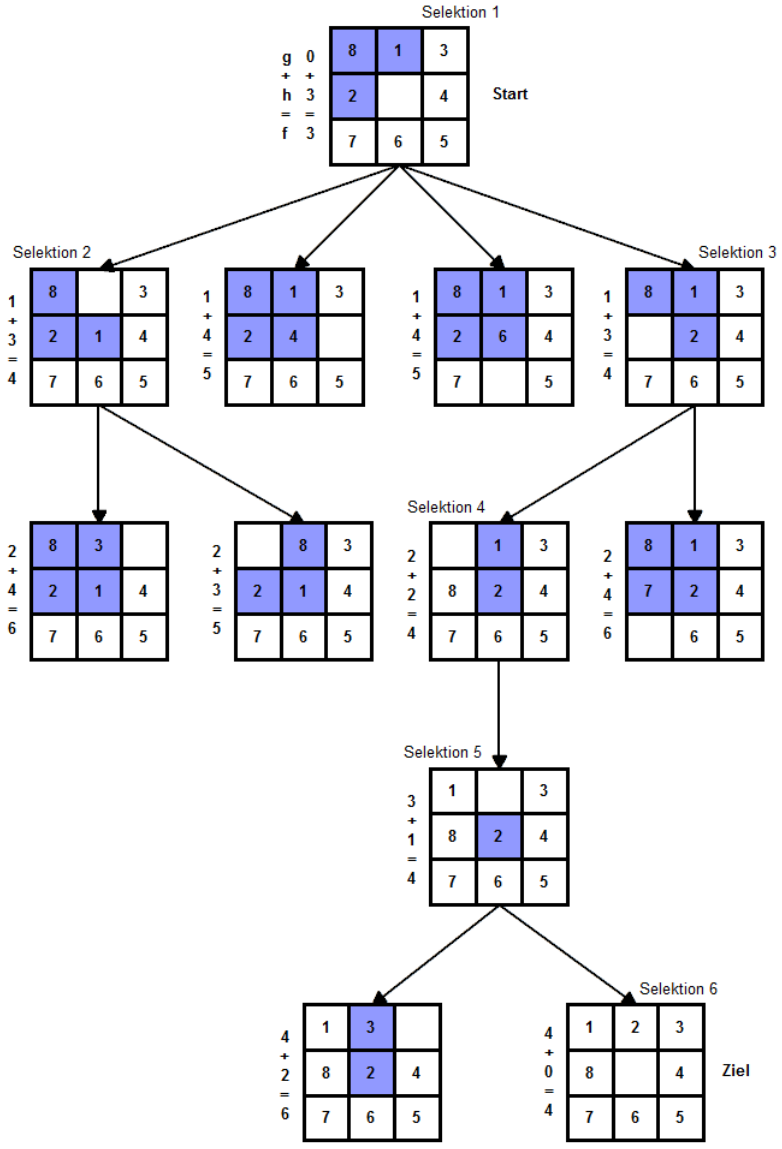


Abbildung 7.6: Suchbaum G_{S1} zu h_1

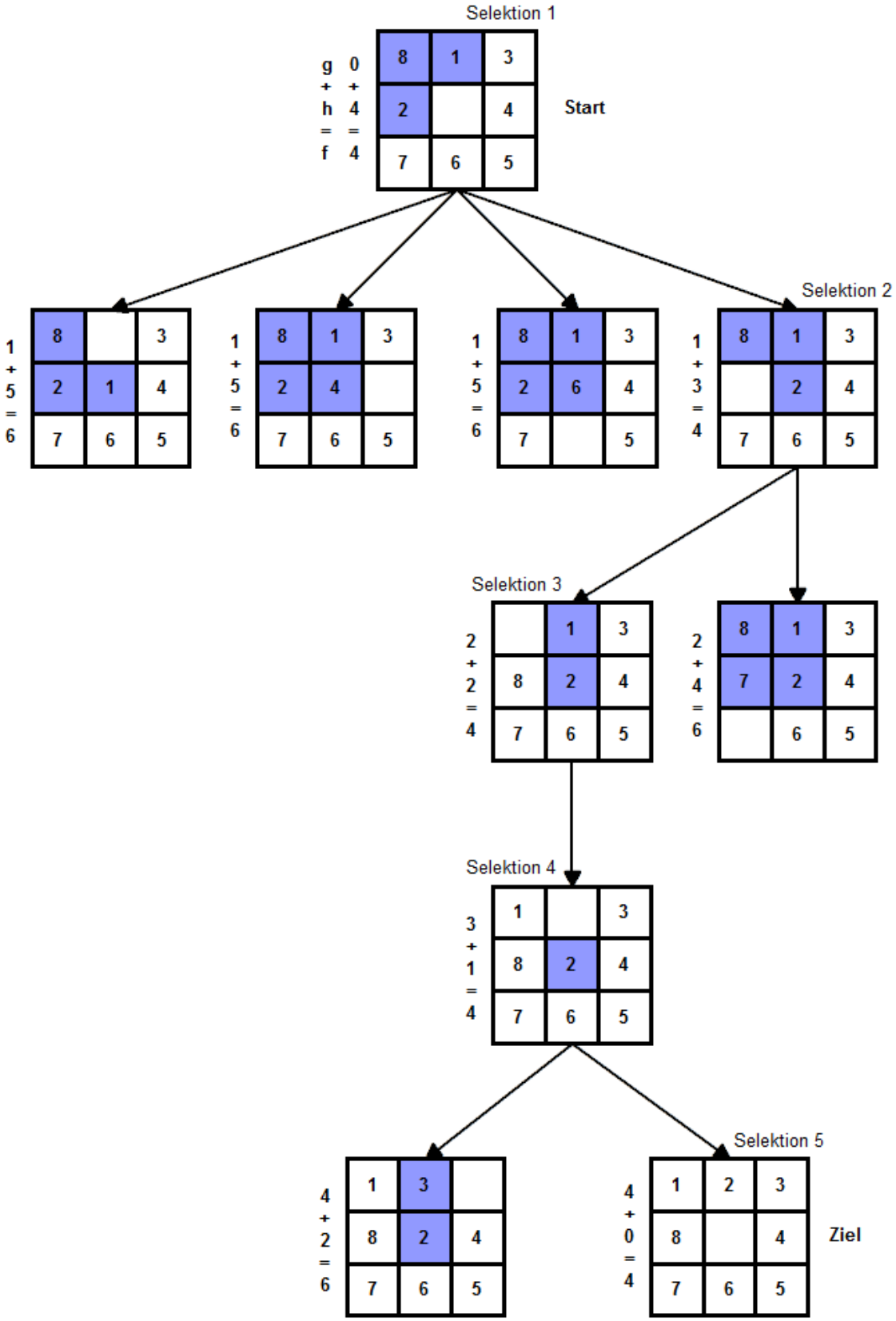


Abbildung 7.7: Suchbaum G_{S2} zu h_2

7.7.5 A* am Beispiel Graphen

Abschließend soll ein A* Algorithmus für das in Abschnitt 6.1 präsentierte Problem entworfen werden. Branch and Bound + Extended List wird wie beschrieben verwendet, die Heuristikfunktion

$$h(n) = h_e(n,x) = sqrt((n^1 - x^1)^2 + \cdots + (n^k - x^k)^2), \text{ mit } x \text{ als Ziel-Knoten der Suche,}$$

wird für die Problemstellung übernommen. Da es sich um die vereinfachte representation einer Karte handelt, wird $k = 2$ gewählt. Der resultierende Graph mit den in rot durch die Heuristik bestimmten Werten:

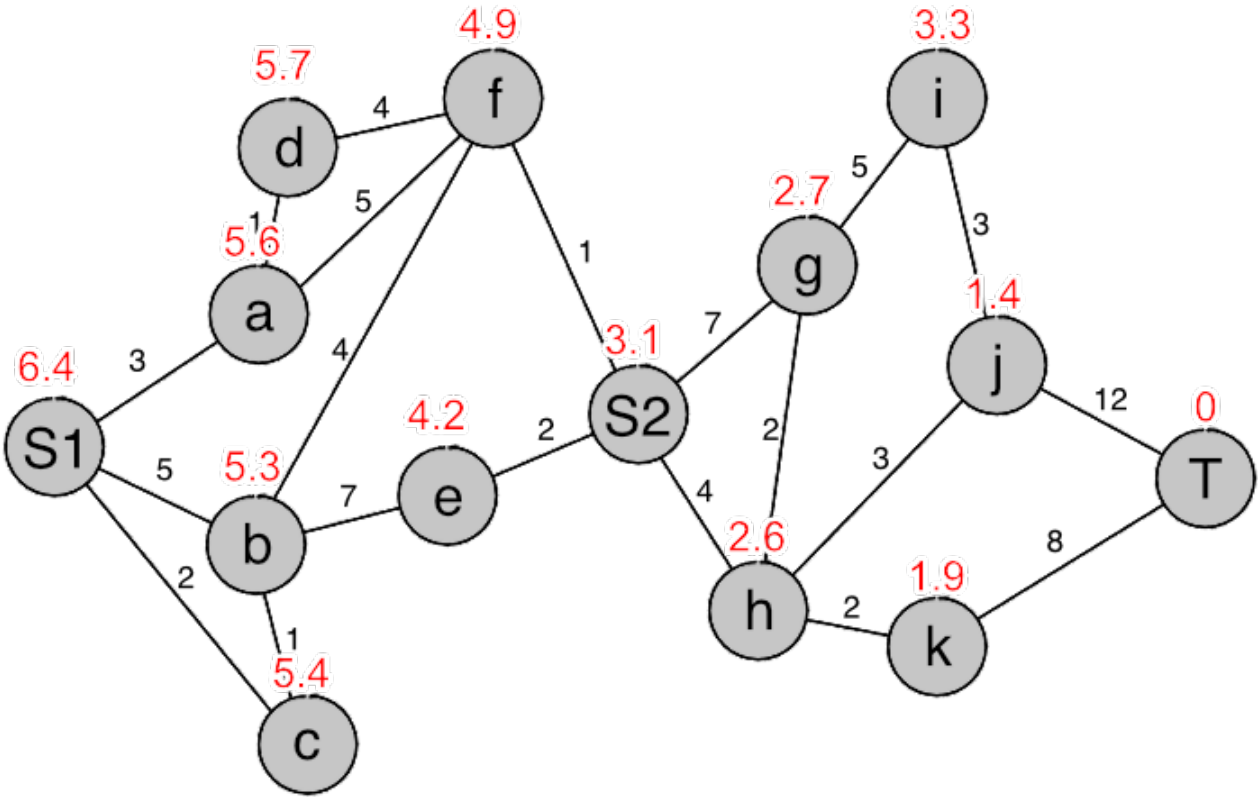


Abbildung 7.8: Beispiel Graph mit Heuristik

Für die Suche des kürzesten Weges von S1 bzw. S2 nach T ergeben sich die beiden Suchbäume B1 (Abbildung6.9) und B2 (Abbildung6.10). Die in blau gekennzeichneten Zahlen geben die Reihenfolge an in der die Pfade begangen werden. Rote Zahlen geben den von der Heuristik geschätzten Wert für den entsprechenden Knoten an. Die Schwarzen Zahlen geben die tatsächliche Weglänge vom Startknoten zum betrachteten Knoten an. In lila werden die geschätzten Gesamtkosten eines Knotens angegeben. Als Grau markierte Pfade wurden aufgrund der Extended List ausgeschlossen.

An beiden Suchbäumen ist gut zu erkennen, dass viele Abzweigungen aufgrund der Extended List nahezu sofort abgeschnitten werden (gekennzeichnet durch graue Pfade). Am Baum mit Wurzel S2 lässt sich der nutzen einer Heuristik nachvollziehen. Die Kosten der Abzweigungen e, f in den linken Teil des Graphen nehmen durch die Heuristik schneller zu und dies sorgt dafür, dass eine Suche dort obsolet (auch durch graue Pfade gekennzeichnet) wird, sobald die geschätzten Kosten größer als 14 sind (Der kürzeste Weg von S2 nach T hat die länge 14). Die unterschiedliche Struktur der Bäume lässt sich durch den Startpunkt der Suche erklären. Aufgrund dessen, dass die Knoten S1 und T auf entgegengesetzten Seiten des Graphen liegen, ergibt sich ein tiefer Baum und da der Knoten S2 den Graphen in zwei Partitionen auftrennt, bildet sich bei der Suche des kürzesten Weges von S2 nach T ein breiter Suchbaum.

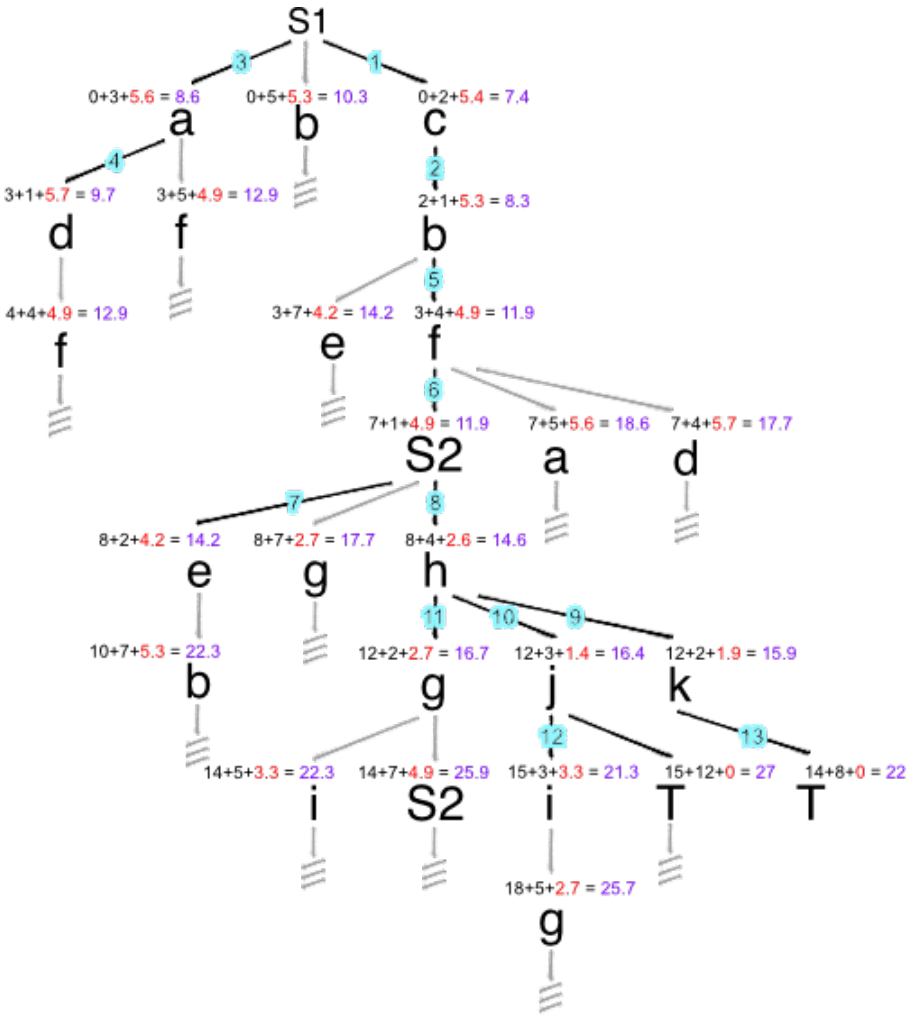


Abbildung 7.9: Such Baum B1 von T nach S1

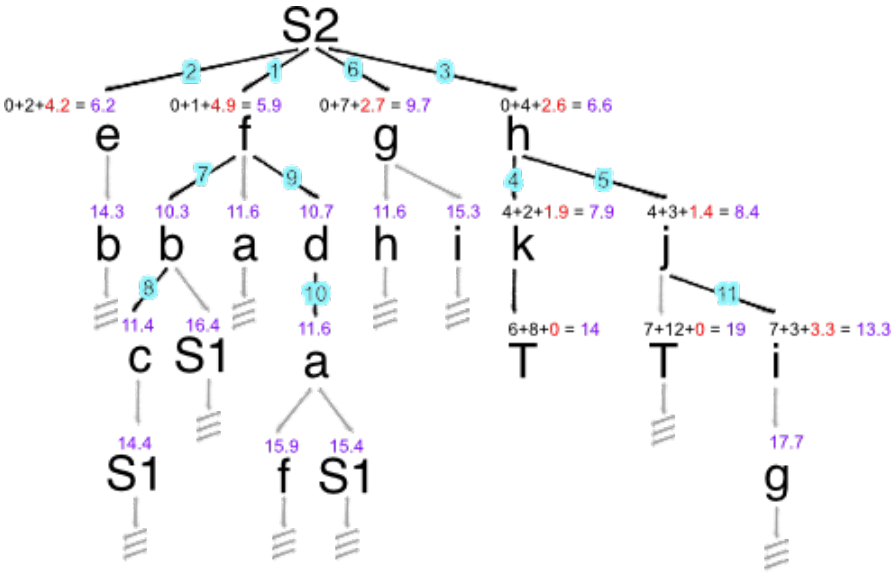


Abbildung 7.10: Such Baum B2 von T nach S2

8. Suchalgorithmen für Spiele

8.1 Intro & Motivation

In den vorangegangenen Kapiteln haben wir schon einige Suchverfahren kennengelernt, welche uns nun helfen werden Algorithmen zu verstehen, die verwendet werden um Computergegner für Nullsummenspiele zu entwickeln. Hierbei werden wir uns den Minimax-Algorithmus, sowie den Alpha-Beta-Algorithmus genauer anschauen, welche beide versuchen sich in die Lage des Gegners zu versetzen, um so herauszufinden, welche Reaktion dieser auf einen Zug von uns wählen würde. Ziel der Algorithmen ist es, ein für uns bestmögliches Ergebnis und somit, wenn möglich, einen Sieg zu erlangen. Dieses Vorgehen kann eine Betrachtung von sehr vielen möglichen Zügen zur Folge haben, was sehr viel Rechenaufwand bedeutet. Im Folgenden werden wir jedoch mit Hilfe des Alpha-Beta-Algorithmus eine Methode kennenlernen, die es uns ermöglicht die Anzahl der zu betrachtenden Züge zu reduzieren.

8.2 Inhaltliche Ausarbeitung des Themas

Wir betrachten ein Nullsummenspiel, was bedeutet, dass der Gewinn des einen Spielers (Nutzwert positiv) mit dem Verlust des Gegners (Nutzwert negativ) äquivalent ist und ein Unentschieden keine Auswirkungen auf die Nullsummeneigenschaft hat (Nutzwert 0). Gegeben sei für dieses Spiel ein Spielbaum $T = (V;E)$. Bei einem gleichförmigen Spielbaum besitzt jeder innere Knoten den gleichen Verzweigungsgrad b und alle Blätter befinden sich in der gleichen Tiefe d, sodass die Zahl der Knoten in einem gleichförmigen Baum exponentiell mit der Tiefe des Baumes wächst (b^d). Jedoch gibt es auch Spielbäume bei denen der Verzweigungsgrad der Ebenen variiert. Der Spielbaum beinhaltet alle möglichen Züge und Spielausgänge. Die Tiefe und der Verzweigungsgrad hängt von dem Spiel ab, welches wir betrachten. Beispielsweise hat der Spielbaum zu einem Tic-Tac-Toe Spiel eine Tiefe von 9 (da es 9 Felder zu besetzen gibt) und zu Beginn des Spiels einen Verzweigungsgrad von 9 (weil noch alle 9 Felder zur Auswahl stehen), der mit wachsender Tiefe allerdings abnimmt, da nach jedem Zug immer ein leeres Feld weniger vorhanden ist.

8.2.1 Problemstellung

Zu dem gegebenen Nullsummenspiel wollen wir einen Computergegner programmieren, der in der Lage ist die optimale Antwort auf jede Spielposition, bei optimalem Spiel beider Spieler, zu finden. Dafür müssen wir jedoch vorher festlegen, was für uns optimal in diesem Zusammenhang bedeutet. Wir gehen davon aus, dass der Gegner immer den für ihn optimalen Zug wählt, d. h. den Zug, der für ihn zum besten Nutzwert und somit für uns zum schlechtesten Nutzwert führt. Analog wollen wir agieren. Wie dies genau funktioniert werden wir im folgenden Abschnitt sehen.

8.2.2 Methoden

Minimax Algorithmus

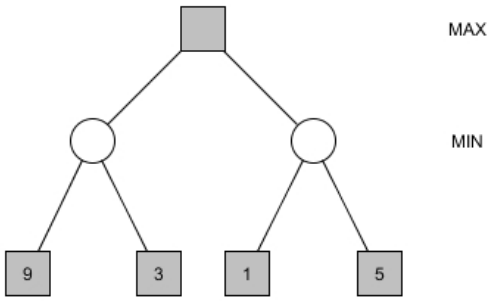
Im folgenden bezeichnen wir unseren Spieler mit MAX und den gegnerischen Spieler mit MIN. Wie zuvor schon angedeutet ist das Ziel des Minimax-Algorithmus den Minimax-Wert für die aktuelle Stellung zu bestimmen. Der Minimax-Wert der Endstellungen (Blätter) entspricht dem Nutzwert für MAX. Die Berechnung der anderen Knoten erfolgt rekursiv mit Hilfe der folgenden Bewertungsfunktion:

$$Minimax(k) = \begin{cases} Nutzwert(k) & \text{falls } k \text{ Endzustand} \\ max\{minimax(t) \mid t \in N(t)\} & \text{falls } k \text{ MAX-Knoten} \\ min\{minimax(t) \mid t \in N(t)\} & \text{falls } k \text{ MIN-Knoten} \end{cases}$$

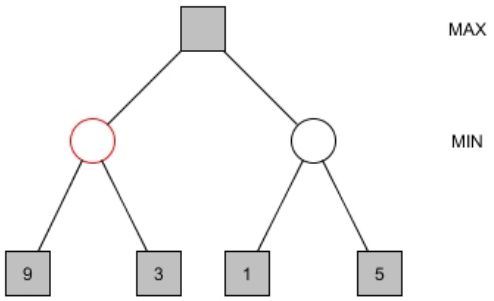
wobei $N(t)$ die Funktion ist, die die Kinder im Baum liefert. Die Auswertung der Knoten geschieht Bottom up, das bedeutet wir beginnen bei den Blättern und bewerten von dort ausgehend die Elternknoten und anschließend deren Eltern bis wir irgendwann bei der Wurzel angelangt sind. Der Minimax Algorithmus liefert nicht unbedingt den höchsten Nutzwert der Blätter, sondern den größten Nutzwert den man erhalten kann, wenn der Gegner selbst versucht zu gewinnen und somit den Nutzwert für uns möglichst klein zu halten.

Anschaulicher wird das ganze, wenn man sich den Algorithmus an einem Beispiel anschaut.
Beispiel

Gegeben sei der folgende Spielbaum

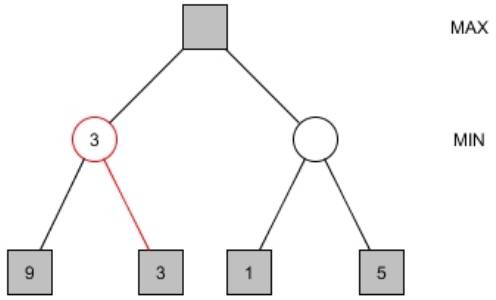


Angenommen MIN wäre an der markierten Position:

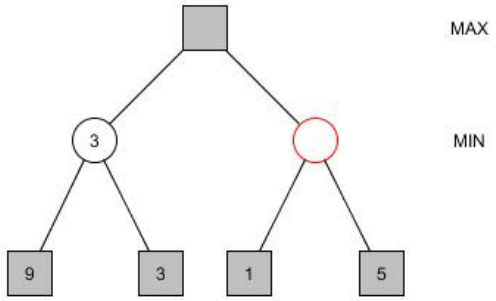


Dann würde MIN den Zug auswählen, der für ihn den größten Nutzwert und somit für uns den kleinsten Nutzwert hat. Das heißt in dem Fall:

$$min\{minimax(t)|t \in N(t)\} = min\{9,3\} = 3$$

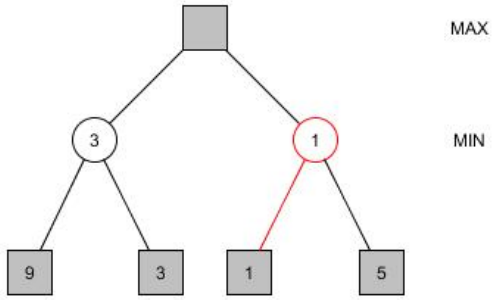


Angenommen MIN wäre an der markierten Position:

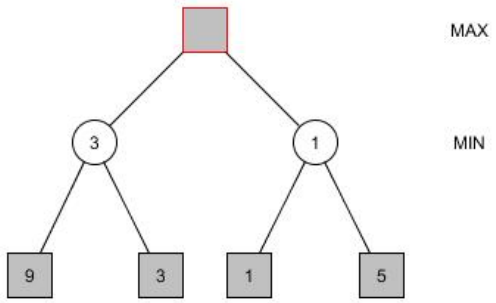


Dann würde MIN den Zug auswählen, der für ihn den größten Nutzwert und somit für uns den kleinsten Nutzwert hat. Das heißt in dem Fall:

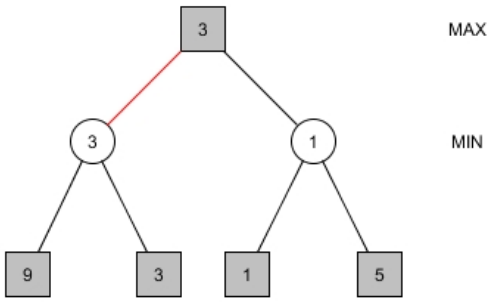
$$min\{minimax(t)|t \in N(t)\} = min\{1,5\} = 1$$



Nun betrachten wir den Zug von MAX. Der Suchbaum sieht wie folgt aus

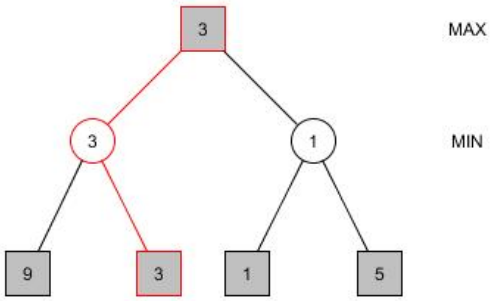


MAX würde den Zug wählen, der für ihn den höchsten Nutzwert hat.



$max\{minimax(t)|t \in N(t)\} = max\{3,1\} = 3$

Somit sieht der Suchbaum nach Abschluss des Minimax-Algorithmus wie folgt aus, wobei der rote Pfad den optimalen Nutzwert liefert, wenn beide Spieler optimal spielen. Somit sollte MAX den linken Knoten wählen.



Problem:
Der Minimax-Algorithmus durchsucht den Suchbaum $T = (V,E)$ mit Tiefensuche, in der Zeit $O(max\{|V|,|E|\}) = O(|V|)$. Somit benötigt die Suche zwar lineare Zeit, jedoch wächst der Baum exponentiell mit zunehmender Tiefe ($O(|V|) = O(b^d)$). Somit ist der Algorithmus für Bäume mit großer Tiefe ineffizient.

Alpha-Beta-Algorithmus
Der Alpha-Beta-Algorithmus ist eine Verbesserung vom Minimax-Algorithmus, da die Zugriffsanzahl reduziert wird, indem Teile des Suchbaums nicht durchsucht werden, ohne dabei das Ergebnis zu verfälschen. Für die Umsetzung werden die Variablen α und β eingeführt, wobei α der Nutzwert ist, welchen MAX mindestens erreicht und β der Nutzwert, den MIN höchstens erreicht. Die Auswertung der Knoten geschieht on-the-fly, d. h. nur dann, wenn wir sie wirklich benötigen.

MAX-Knoten:
Betrachten wir einen MAX Knoten und stellen dort einen Wert fest, der größer ist als β, brauchen wir den Teilbaum nicht weiter betrachten, da MIN diesen nicht wählen würde, weil der andere Teilbaum einen kleineren Nutzwert liefert. Das nicht weiter Betrachten des Teilbaums bezeichnet man als Beta-Cutoff. Ist der Wert des MAX Knoten größer als α, erhöht sich der Wert den MAX mindestens erreicht und wir aktualisieren α auf den Wert des Knotens.

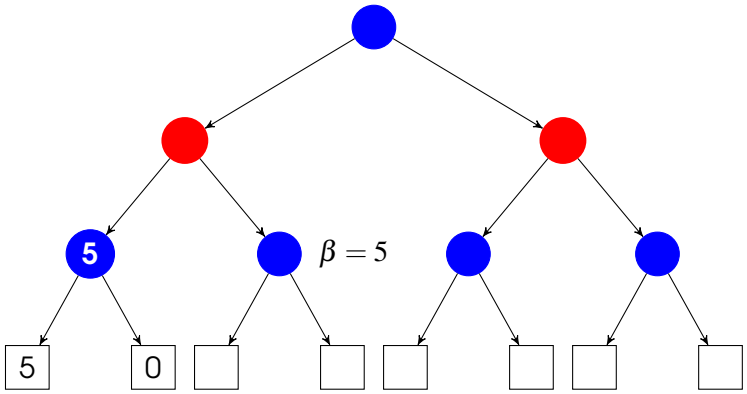
MIN-Knoten:
Ist der Wert eines MIN Knotens kleiner als der Wert von α müssen wir diesen Teilbaum nicht weiter analysieren, da MAX immer den Teilbaum mit dem größten Nutzwert auswählt und da α größer ist als der Wert des betrachteten Knotens existiert ein Teilbaum mit einem größeren Nutzwert. Also würde hier ein Cutoff stattfinden, welchen man als Alpha-Cutoff bezeichnet. Sollte jedoch der Wert des MIN Knotens kleiner als β sein, sinkt der Wert den MIN höchstens erreicht und wir müssen β auf diesen Wert anpassen.

Betrachten wir ein Beispiel, um den Algorithmus besser zu verstehen.

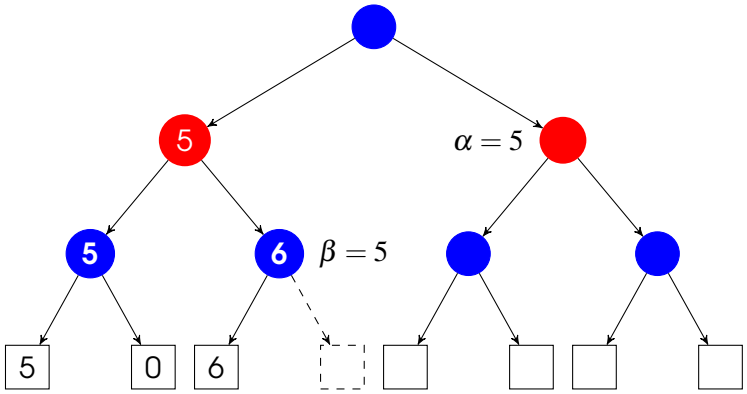
```
Data: Graph G = (V,E), Blätter mit Nutzwert, s = Knoten
Result: Nutzwert für jeden knoten
alpha-beta-suche(s)
return maximalerWert(s, -∞, ∞)
maximalerWert(Knoten, α, β)
if s = Blatt then
    return wert(s)
end
else
    wert = α
    foreach k= Kind(s) do
        wert = max{wert, minimalerWert(k, wert, β)}
        if wert >= β then
            return wert
        end
        α = max{α, wert}
    end
    return wert
end
minimalerWert(Knoten, α, β)
if s = Blatt then
    return wert(s)
end
else
    wert = β
    foreach k= Kind(Knoten) do
        wert = min{wert, maximalerWert(k, α, wert)}
        if wert <= α then
            return wert
        end
        β = min{β, wert}
    end
    return wert
end
```

Algorithm 6: Alpha-Beta-Algorithmus

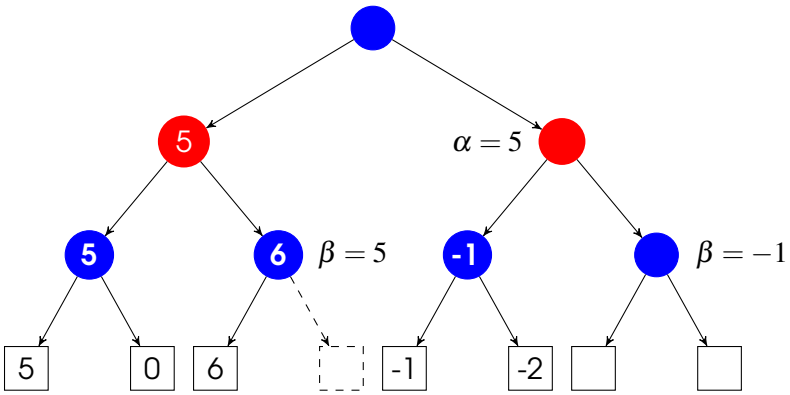
Beispiel:
Gegeben sei der folgender Spielbaum, blaue Knoten sind MAX- und rote MIN-Entscheidungen. Wir folgen immer den linken Kindern. Zunächst treffen wir unsere erste Entscheidung, dafür müssen beide Kinder des linkesten Pfades evaluiert werden. Wir entscheiden uns für die 5. Dieser Wert ist der neue β -Wert für seinen Schwester-Knoten. Heißt unser Gegenspieler MIN wird sich höchstens für den Wert 5 entscheiden.



Wir evaluieren den nächsten Knoten im Nachbar-Graphen. Er ist 6. Da dieser Wert größer-gleich des β-Wert ist, müssen wir den Rest des Teil-Graphen nicht zu evaluieren. Dies ist überflüßig, da MIN den tiefsten Wert wählen wird, welcher mindestens 5 ist. Wir brauchen also nicht nach einem noch größeren Wert suchen.
Dies nennt man $\beta - Cut$.
Der neue α-Wert des roten Schwester-Konten ist nun 5. Heißt: Wir können danach mindestens eine 5 wählen.

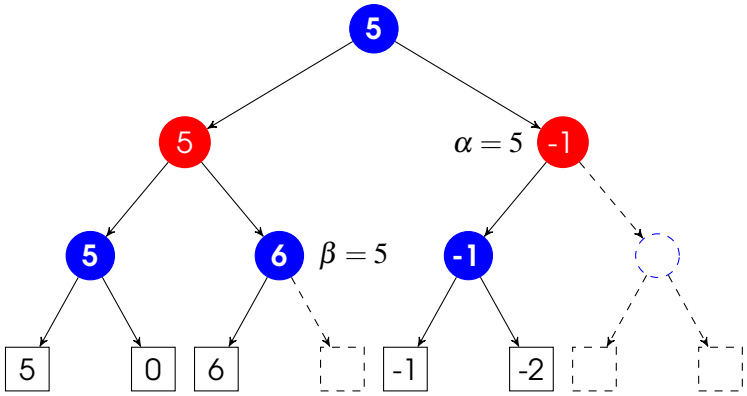


Da der ganze linke Teilbaum evaluiert ist, kann der rechte Baum evaluiert werden. Dort wird wieder dem linkesten Pfad gefolgt. Wir evaluieren und setzen den β-Wert für den Schwesterknoten genauso wie im linken Teilbaum.



Der Wert unserer letzten Entscheidung ist nun kleiner-gleich des nächsten α-Werts. Dies bedeutet, dass unsere Seite des Graphens nicht mehr relevant ist. Wir (MAX) können mindestens eine 5 wählen. Der Gegner (MIN) wird allerdings in diesem Teil des Graphens höchstens eine -1 wählen. Wir brauchen den Rest des rechten Graphen also

nicht weiter betrachten.
Dies ist ein $\alpha - Cut$.



Die gestrichelten Teile sind nun (mögliche) Teilgraphen, welche mit dem normalen MiniMax Algorithmus hätten evaluiert werden müssen. In diesem Beispiel haben wir 3/8 Blättern ignorieren können. Bei steigender Tiefe fallen so große Unterbäume weg.

8.3 Abgrenzung / Vergleich zu den vorherigen Kapitel

In den vorherigen Kapiteln haben wir verschiedene Suchalgorithmen (Bestensuche, Branch-and-Bound, A*) kennengelernt. Diese liefern den Pfad, der zu dem Blatt mit dem höchsten Nutzwert führt. Jedoch beachten sie nicht, ob der Gegner auch diesen Pfad wählen würde. Der Gegner würde versuchen den geringsten Nutzwert für uns zu erreichen und somit versuchen von diesem Pfad abzuweichen. Folglich würden sie keine realistische Einschätzung darüber geben, welche Züge das beste Spielergebnis liefern. Im Gegensatz dazu betrachten sowohl der Minimax, als auch der Alpha-Beta Algorithmus das Verhalten des Gegners, welches zu einer guten Einschätzung der Spielsituation führt.

8.4 Fazit & Bewertung

Zusammenfassend lässt sich sagen, dass der Minimax- und der Alpha-Beta-Algorithmus zum selben besten Ergebnis für den MAX-Spieler führen und sich lediglich in ihrer Effizienz unterscheiden können, da der Alpha-Beta-Algorithmus weniger Berechnungen und somit eine kürzere Rechnungszeit benötigen kann. Je nach Suchbaum kann die Anzahl der Cutoffs und die Einsparung der Rechenzeit sehr groß ausfallen. Aufgrund dessen ist der Alpa-Beta-Algorithmus eine Grundlage für viele Algorithmen im Bereich der Nullsummenspiele für zwei Personen. Für die Berechnungen der Algorithmen sind die Werte der Blätter von größerer Bedeutung, welche durch die Heuristik gegeben sind. Alles in allem kann man sagen, dass man sowohl mit Minimax als auch mit Alpha Beta einen Computergegner programmieren kann, der es einem Menschen sehr schwer, wenn nicht sogar unmöglich, macht zu gewinnen und somit eine künstliche Intelligenz bei Nullsummenspielen für die Zukunft denkbar ist.

8.5 Quellen und Literatur

- MIT - Lecture 6: Search: Games, Minimax, and Alpha-Beta
- http://home.in.tum.de/ adorf/pub/alphabeta-seminar-paper.pdf
- https://de.wikipedia.org/wiki/Minimax-Algorithmus
- https://de.wikipedia.org/wiki/Alpha-Beta-Suche

9. Konsistenz und Suche

9.1 Einführung

Wir haben bereits Probleme kennengelernt, wie beispielsweise das Einfärbeproblem von Landkarten, Kreuzworträtsel oder Sudokus. Das n-Damenproblem stellt die Frage, wie man n Damen auf einem $n \times n$-Schachbrett stellen kann, sodass keine Dame von einer anderen geschlagen werden kann. Wir wollen in diesem Kapitel einfache Strategien kennenlernen, solche Probleme effizient mithilfe von *Konsistenz und Suche* zu lösen.

9.2 Constraint Satisfaction Problem

Alle die genannten Probleme können wir wie folgt Formalisieren:

Definition 9.2.1 — Constraint Satisfaction Problem. Ein *Constraint Satisfaction Problem* (CSP) ist ein Tripel $P = (X, D, C)$, wobei

- $X = \{X_1, \ldots, X_N\}$ eine endliche Menge von Variablen,
- $D = \{\text{domain}(X_1), \ldots, \text{domain}(X_N)\}$ eine Menge der Domänen der Variablen in X, sowie
- $C = \{C_1, \ldots, C_M\}$ eine endliche Menge von Contraints $C_i = \langle (\text{ Vars}, \text{Relation}) \rangle$ wobei `Vars` ein Tupel von Variablen ist, sowie `Relation` eine Relation zwischen den Variablen aus `Vars` ist.

Eine Belegung einer Variablen $X_i \in X$ heisst *konsistent*, falls sie keine der gegebenen Constraints verletzt. Eine Belegung aller Variablen heisst *vollständig*, ansonsten *partiell*. Eine Lösung eines Constraint Satisfaction Problems ist eine vollständige, konsistente Belegung.

Wir können hierbei noch unterscheiden, zwischen

- Unären Constraints, welche sich auf einzelne Variablen beschränken, wie beispielsweise $C_1 = \langle (X_1), X_1 \text{ist gerade} \rangle$,
- Binären Constraints, welche sich auf genau zwei Variablen bieziehen, wie $C_2 = \langle (X_1, X_2), X_1 > X_2 \rangle$, oder
- allgemein als n-äre Constraints, welche sich auf n Variablen beziehen, wie $C_3 = \langle (X_1, \ldots, X_n),$ Alle paarweise verschiede

Wir können jedoch alle n-ären Contraints ($n > 2$) auch als unäre oder binäre Constraints auffassen:

Beispiel 1. Betrachte $C = \langle (X_1, X_2, X_3), \text{Alle paarweise verschieden} \rangle$. Wir führen eine Hilfsvariable X_h ein, mit der Domäne

$$\text{domain}(X_h) = \text{domain}(X_1) \times \text{domain}(X_2) \times \text{domain}(X_3)$$

Nun können wir äquivalent zu C die folgenden Constraints konstruieren:

$$
\begin{aligned}
C_h &= \langle (X_h), (a,b,c) \in \text{domain}(X_h) | a \neq b \wedge b \neq c \wedge a \neq c \rangle \\
C_1 &= \langle (X_1, X_h), X_1 = X_h[0] \rangle \\
C_2 &= \langle (X_2, X_h), X_2 = X_h[1] \rangle \\
C_3 &= \langle (X_3, X_h), X_3 = X_h[2] \rangle
\end{aligned}
$$

wobei $X_h[k]$ für den k-ten Eintrag in der vektorwertigen Variable X_h steht. In diesem konkreten Beispiel könnte man jedoch auch folgende Konstruktion wählen

$$
\begin{aligned}
\hat{C}_1 &= \langle (X_1, X_2), X_1 \neq X_2 \rangle \\
\hat{C}_2 &= \langle (X_2, X_3), X_2 \neq X_3 \rangle \\
\hat{C}_3 &= \langle (X_1, X_3), X_1 \neq X_3 \rangle
\end{aligned}
$$

Jedoch demonstriert erstere Konstruktion, dass wir theoretisch von unären und binären Constraints ausgehen können. Eine kleine Einschränkung jedoch ist, dass unsere Konsistenzalgorithmen sich eventuell anders verhalten können.

Ein weiteres Beispiel für CSP sind Krypto-arithmetische Puzzle

Beispiel 2 (Krypto-arthimetisches Problem)**.** Gesucht ist eine paarweise unterschiedliche Belegung der Variablen **A**, **B**, **C**, so dass die Formel in Tabelle 9.1 gilt. Mithilfe der Variable **U** können wir das Problem formalisieren. Dabei

Tabelle 9.1: Krypto-arithmetisches Puzzle

	A	
+	**B**	
B	**C**	

beachte, dass wir keine führenden Nullen zulassen wollen. Wir definieren also unsere Variablen

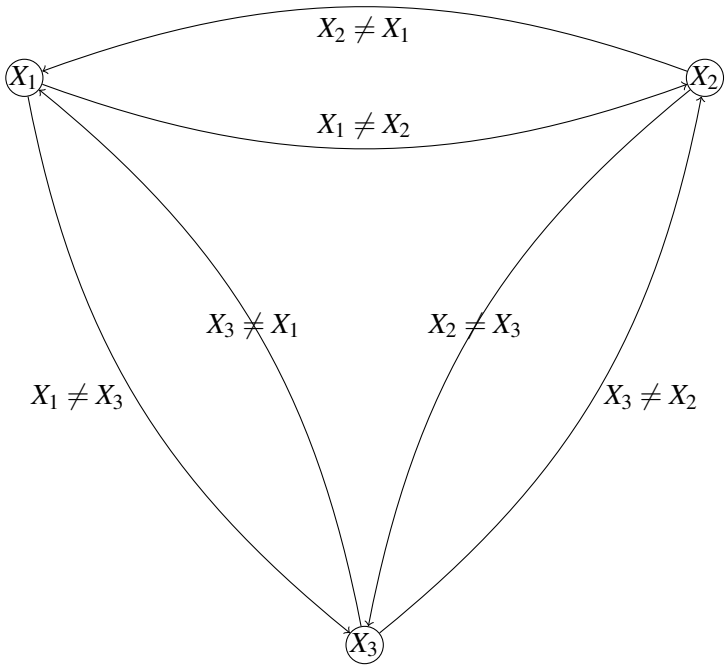
$$
\begin{aligned}
\text{domain}(A) = \text{domain}(C) &= \{0,1,2,3,4,5,6,7,8,9\} \\
\text{domain}(B) &= \{1,2,3,4,5,6,7,8,9\} \\
\text{domain}(U) &= \{0,1\}
\end{aligned}
$$

mit den Constraints

$$
\begin{aligned}
A + B &= C + 10 \cdot U \\
B &= U \\
A &\neq B \neq C
\end{aligned}
$$

9.2.1 Konsistenz

Um ein CSP zu lösen, können in der Regel bekannte Suchalgorithmen verwendet werden, jedoch bieten sich Algorithmen an, wie AC-3 oder Backtracking. Fassen wir ein CSP als einen Graphen auf, bei dem die Knoten für die Belegungsmöglichkeiten der jeweiligen Variablen steht, die Kanten hingegen für die Constraint-Relationen, so



können wir den AC-3 Algorithmus anwenden.

Dazu definieren wir zuerst:

Definition 9.2.2 — node- und arc-consistency.

- Eine Variable X_i heisst *node-consistent*, wenn für alle möglichen Werte aus domain(X_i) alle unären Constraints auf X_i erfüllen.
- Eine Variable X_i heisst *arc-consistent bezüglich der Variable* X_j, wenn für alle $v \in$ domain(X_i) ein $w \in$ domain(X_j) existiert, sodass alle Constraints auf (X_i, X_j) erfüllt sind.
- Ein CSP $P = (X, D, C)$ heisst *arc-consistent*, wenn alle Variablen in X paarweise arc-consistent sind.

Der Arc-consistency Algorithmus (AC-3), welcher solche Graphen verwendet, findet Konflikte, die beim Belegen der Variablen per Backtracking auftreten und entfernt diese aus dem Suchraum. Falls die Domänen verkleinert wurden, werden alle Kanten zu den benachbarten Knoten wieder überprüft. Der Algorithmus terminiert, da wir nur endlich viele Knoten und Kanten haben.

```
function AC3
 // Reduziert Domänen
     queue = Alle Kanten des CSP
     while (!empty(queue))
         Entferne eine Kante (x, y) aus queue;
         if(EntferneInkonsistenteWerte(x, y))
             foreach (Nachbar z von x)
                 queue.ADD(Kante(z, x))
end
function EntferneInkonsistenteWerte(x, y)
 // Liefert true, wenn Domäne D(x) reduziert wird
     removed = false
     foreach (Value v1 in D(x))
         if(Kein v2 in D(Y), so dass (x=v1, y=v2) alle Constraints
                                         zwischen (x,y) erfüllt)
             D(x).LÖSCHE(v1)
             removed = true
     return removed
end
```

Wenn mindestens ein Wertebereich einer Variable leer ist, ist offensichtlich keine vollständige Zuweisung und damit keine Lösung mehr möglich. Wenn alle Wertebereiche nur noch einen Wert enthalten, dann können wir diese Werte als Zuweisung wählen und haben eine Lösung gefunden. In den meisten Fällen werden wir aber um eine Suche

nicht herum kommen, denn häufig bleiben mehrere Variablen in den Domänen übrig.
Wir wollen uns weiter mit Beispiel 2 befassen:

Beispiel 3. Mit *Backtracking-Search* sieht man

```
C ausgewählt
  C = 0 konsistent
    B ausgewählt
      B = 1 konsistent
        A ausgewählt
          A = 0 nicht konsistent (verletzt A != C constraint)
          A = 1 nicht konsistent (verletzt A != B constraint)
          A = 2 konsistent
            U ausgewählt
              U = 0 nicht konsistent (B = U)
              U = 1 nicht konsistent (A+B=C+10*U)
            U entfernt
          A = 3 konsistent
          ...
          A = 9 konsistent
            U ausgewählt
              U = 0 nicht konsistent
              U = 1 konsistent
              ergebnis <- C=0,B=1, A=9, U=1
```

Mit Forward-Checking verkleinern wir in jedem Schritt die Domäne der relevanten Variablen

```
C ausgewählt
  C = 0 konsistent
    FC: domain(A) = {1,2,3,4,5,6,7,8,9}
    B ausgewählt
      B = 1 konsistent
        FC: domain(A) = {2,3,4,5,6,7,8,9}, domain(U) = {1}
        A ausgewählt
          A = 2 konsistent (hier kein A=0,1)
            FC: domain(U) = {}, backtrack
          A = 3 konsistent
          ...
          A = 9 konsistent
            FC: keine Änderung
            U ausgewählt
              U = 1 konsistent (U=0 kommt nicht vor)
              ergebnis <- C=0, B=1, A=9,U=1
```

9.3 Mit Prolog CSPs lösen

(SWI-)Prolog ist eine logische Programmiersprache, welche per Backtracking aus einer Wissensdatenbank und Termen CSPs bedingt lösen kann.

9.3.1 SWI-Prolog

Mit dem Standardrepertoire von SWI-Prolog wollen wir beispielsweise das Crypto-Arithmetische Rätsel in Tabelle 9.2 lösen, dann beschreiben wir das Problem beispielsweise durch

```
gen([S,E,N,D,M,O,R,Y]) :-
  permutation([S,E,N,D,M,O,R,Y,_,_],[0,1,2,3,4,5,6,7,8,9]).
```

Tabelle 9.2: Send More Money

	S	**E**	**N**	**D**
+	**M**	**O**	**R**	**E**
M	**O**	**N**	**E**	**Y**

```
gl1([S,E,N,D,M,O,R,Y]) :-
        1000*S+100*E+10*N+D
       +1000*M+100*O+10*R+E =:=
10000*M+1000*O+100*N+10*E+Y.

lsg :- gen([S,E,N,D,M,O,R,Y]),gl1([S,E,N,D,M,O,R,Y]).
```

In dieser Version liefert uns Prolog keine explizite Belegung der Variablen, jedoch erhalten wir die Bestätigung

```
?- lsg.
true
```

Leider akzeptiert der `=:=` Operator auf der rechten Seite keine freien Variablen, weshalb hier das Prädikat `gen` unabdingbar ist. Wir können in Prolog jedoch das Modul `clpfd` verwenden.

9.3.2 CLP(FD)

Constraint Logic Programming over Finite Domains verwendet, anders als Prolog, nun Konsistenzalgorithmen, um die Domänen der Variablen sukzessive zu verkleinern, und somit den Suchraum zu verkleinern. In Prolog mit CLP(FD) sieht unser Puzzle nun wie folgt aus:

```
:- use_module(library(clpfd)).

sendmore([S,E,N,D,M,O,R,Y]) :-
        Vars=[E,N,D,O,R,Y],
        Vars ins 0..9,
        S in 1..9,
        M in 1..9,
        all_different([S,E,N,D,M,O,R,Y]),
        1000*S+100*E+10*N+D
       +1000*M+100*O+10*R+E #=
10000*M+1000*O+100*N+10*E+Y.
```

Tatsächlich erhalten wir damit das Ergebnis

```
?- sendmore([S,E,N,D,M,O,R,Y]).
S = 9,
M = 1,
O = 0,
E in 4..7,
all_different([9, E, N, D, 1, 0, R, Y]),
91*E+D+10*R#=90*N+Y,
N in 5..8,
D in 2..8,
R in 2..8,
Y in 2..8.
```

Wir haben also die Domänen eingeschränkt, aber existiert tatsächlich eine Lösung unseres Puzzles? Wollen wir eine explizite Lösung erhalten, so verwenden wir `labeling`.

```
26 ?- sendmore([S,E,N,D,M,O,R,Y]),label([ff],[S,M,O,E,N,D,R,Y]).
S = 9,
E = 5,
N = 6,
D = 7,
M = 1,
O = 0,
R = 8,
Y = 2 ;
```

`labeling` zwingt Prolog nun die Variablen zu benennen. Dazu stehen einige verschiedene Suchstrategien zur Verfügung.

Variablen		
	leftmost	Die Variablen werden von links nach rechts durchgegangen (default)
	ff	First Fail. Belegt Variablen mit der kleinsten Domäne zuerst.
	ffc	Belegt Variablen mit der kleinsten Domäne und mit den meisten Constraints zuerst.
	min	Die Variable mit der kleinsten unteren Schranke wird zuerst belegt.
	max	Die Variable mit der kleinsten unteren Schranke wird zuerst belegt.
Werte		
	up	Aufsteigend (default)
	down	Absteigend
Verzweigung		
	step	Für jede Variable X entscheide zwischen X=V und X!=V (default)
	enum	Multiple Entscheidung für ein X entsprechend seiner Domäne
	bisect	Entscheidet zwischen X<=M und X>M, mit Mittelpunkt M der Domäne

Verschiedene CSPs benötigen verschiedene Strategien, um möglichst effizient eine Lösung zu finden.

Literaturverzeichnis

[JS] Joshua Schmidt, Constraint Logic Programming: Konsistenz und Suche, 31.05.2016

[TR] Tobias Rosenkranz, Konsistenz und Suche, 06.06.2017

IV Lernende Systeme

10. Support-Vector-Machines

10.1 Intro & Motivation

Obwohl keine allgemein anerkannte Definition von Intelligenz in Menschen (oder Tieren) existiert, ist durchaus Konsens über wichtige Merkmale intelligenten Verhaltens gegeben. So gut wie allen Definitionsversuchen ist gemein, dass Intelligenz mit der Fähigkeit einhergeht, sich in einer sich ändernden Umwelt zurechtzufinden und der Fähigkeit schnell zu lernen. Im Allgemeinen findet Lernen und die Adaptation an Umweltgegebenheiten zum Erreichen bestimmter Ziele statt. Legg und Hutter (2007) schlussfolgern in ihrer Sammlung verschiedener Definitionsversuche:

> "Intelligence measures an agent's ability to achieve goals in a wide range of environments. Features such as the ability to learn and adapt, or to understand, are implicit in the above definition as these capacities enable an agent to succeed in a wide range of environments."

Da bestimmte physikalische Prozesse immer chaotisch, das heißt nicht deterministisch, ablaufen, können nicht alle Veränderungen von Umweltbedingungen vorausgesagt werden. Ein intelligentes System muss demnach notwendigerweise mit Unsicherheit umgehen können, um sinnvoll zu handeln (Lawrence, 2016). Es ist nicht möglich einem intelligenten System die Voraussetzungen für alle möglichen zukünftigen Entwicklungen mit auf den Weg zu geben. Wenn es nicht lernen kann, ist es kein intelligentes System. Die Fähigkeit sich auf neuartige Umgebungen einzustellen ist also essentiell für intelligentes Verhalten. Die Voraussetzung für eine solche Anpassung ist eine sehr allgemeine Fähigkeit aus Erfahrung und Beobachtung zu lernen – Lernen ist der Mechanismus, der eine individuelle Adaptation ermöglicht. Eine weitere Voraussetzung für intelligentes Lernen ist das Vorhandensein eines oder mehrerer Ziele, die bestimme,n aus welchen Informationen überhaupt gelernt werden soll.

Ein intelligentes, anpassungsfähiges System muss also einen Lernmechanismus besitzen, mit dem das System sich selber verbessern, zeitüberdauernd Ziele verfolgen und den Erreichungsfortschritt überwachen kann. Dies ist gerade dann relevant, wenn sich gegebene Umweltbedingungen verändern und Handeln angepasst werden muss. Wenn beispielsweise ein Computersystem überwachen soll, was für Situationen mit hoher Wahrscheinlichkeit zu einer Gefährdung des Straßenverkehrs führen, muss das System etwaige Änderungen in den Rahmenbedingungen – zum Beispiel technologische Erneuerungen, Änderungen der gesetzlichen Bestimmungen oder Änderungen des Verkehrsaufkommens – in seiner zukünftigen Vorhersage berücksichtigen. Das System muss durch Lernen aus Beobachtung einschätzen, welchen Gegebenheiten mit dem Auftreten von Unfällen zusammenhängen, um für die Zukunft Vorhersagen zu erstellen und gegebenenfalls Interventionsmaßnahmen einzuleiten.

10.2 Arten des maschinellen Lernens

Es gibt verschiedenen Klassifikatoren, die dazu genutzt werden um Proben in einem mehrdimensionalen Raum Klassen zuzuordnen. Dabei kann man Klassifikatoren in drei Arten unterscheiden.

1. Unüberwachtes Verfahren: Klassen müssen vom Klassifikator erkannt werden.
2. Überwachtes Vefahren: Klassen werden vorher in einer Trainingsphase erlernt.
3. Bestärkendes Verfahren: Klasseneinteilung ist nicht fest und wird durch hinzufügen von Informationen korrigiert.

Dabei gehört die Support Vector Machine zu Art zwei. Zuerst wird eine Trainingsphase durchlaufen anhand der Klassifikator lernt, die Klassen zu unterscheiden. Nach der Trainingsphase können neue Daten mit Hilfe eines erstellten Modells zu einer der beiden Klassen zugewiesen werden.

Die folgende Ausarbeitung beschäftigt sich mit den Grundlagen statistischen Lernens, welches maschinellen Systemen das Lernen aus Beobachtung ermöglicht. Hierbei gehe ich vor allem auf das sogenannte *überwachte Lernen* ein und beziehe mich in erster Linie auf die Quellen *Artificial Intelligence – A Modern Approach* (3. Ed., Kap. 18; Russel & Norvig, 2010) und *An introduction to Statistical Learning* (James, Witten, Hastie & Tibshirani, 2013).

10.3 Lernen aus Daten

Damit ein System intelligent lernen kann, müssen Beobachtungen in ein Format gebracht werden, mit dem das System umgehen kann. Menschen und Tiere besitzen primäre Sinnesorgane, die Informationen über die Umwelt über Nervenlaufbahnen in ein zentrales Nervensystem transferieren. Dort werden Informationen aus verschiedenen Sinnesmodalitäten integriert und Handlungen geplant; vereinfacht gesagt liegen Informationen in Form von Nervenimpulsen vor. Ein Computersystem muss Informationen über die Umwelt in einem quantifizierten, numerischen Datenformat erhalten. Auch Daten, die distinkte, qualitative Merkmale repräsentieren – beispielsweise die Marke eines Autos – werden dem Computersystem in numerischer Form übergeben.

Beim sogenannten überwachten Lernen (*supervised learning*) werden Daten allgemein im folgenden Format dargestellt: Betrachtet werden n Datenpaare

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n).$$

Hierbei bezeichnen x_i, die *Input*-Variablen, y_i die *Output*-Variablen. X_i ist dabei entweder ein einzelner Wert oder ein Vektor beliebig vieler Werte. Eine Input-Variable könnte beispielsweise die Geschwindigkeit eines Autos, die Automarke oder das Temperament und Geschlecht des Autofahrers darstellen. Ziel des überwachten Lernens ist mithilfe der vorhandenen Input-Variablen (auch genannt: Features, Prädiktoren, unabhängige Variablen) zuverlässige Informationen über den Output (auch genannt: abhängige Variablen, Labels) zu erlangen.

Die Output-Werte y könnten beispielsweise repräsentieren, ob ein Auto am Tag der Datenaufzeichnung einen Unfall hatte. Dies stellt ein qualitatives Label dar: die Variable nimmt zwei distinkte Zustände an, entweder *Unfall – Ja* oder *Unfall – Nein* (im Computer dargestellt als 1 / 0). Probleme, in denen solch ein qualitativer Output vorhergesagt werden soll, werden **Klassifikationsprobleme** genannt; existieren genau zwei qualitative Kategorien, handelt es sich um ein *binäres* Klassifikationsproblem. Angenommen, das System soll nicht vorhersagen, ob Autos genau am Tag der Aufzeichnung einen Unfall haben. Schließlich ist für ein einzelnes Auto das Auftreten eines Unfalls sehr selten und die Variable ist somit wahrscheinlich nicht sehr informativ. Trotzdem soll das System eine Prognose für die Fahrsicherheit eines jeden beobachteten Autos ausstellen, damit in Zukunft riskante Verkehrssituationen mit hoher Präzision eingeschätzt werden können. In diesem Fall könnte als Output die Fahrstrecke, die ein Auto nach der Aufzeichnung noch unfallfrei fährt, dienen. Die Fahrstrecke stellt eine kontinuierliche Variable dar – es werden nicht qualitativ verschiedene Kategorien betrachtet. Mit der Vorhersage kontinuierlicher Variablenwerte beschäftigen sich die sogenannten **Regressionsprobleme**. Tabelle 2.1 zeigt einige exemplarische Input- und Outputwerte für das Regressionsproblem unseres Fahrsicherheitssystems.

Beim unüberwachten Lernen (*unsupervised learning*) liegen keine Informationen bezüglich möglicher Output-Variablen vor – hier wird alleine in den Input-Variablen nach Gesetzmäßigkeiten gesucht, um die Daten auf bestimmte Art und Weise zu strukturieren oder klassifizieren. Dies wäre in unserem Beispiel der Fall, wenn gar keine Information bezüglich des Auftretens von Unfällen vorliegen würden. Die vorhandenen Daten könnten trotzdem nach systematischen Regelmäßigkeiten untersucht werden, etwa mittels *Clustering*. Beispielsweise könnte

Tabelle 10.1: Beispielhafte Input- und Outputwerte für das Fahrsicherheitssystem.

	Automarke	Geschwindigkeit	Laune	Unfallfreie Fahrstrecke	
x_1	Volvo	73 km/h	fröhlich	14,302 km	y_1
x_2	VW	144 km/h	neutral	10,219 km	y_2
x_3	Audi	129 km/h	traurig	21,219 km	y_3
	...	...	...	...	
x_n	Porsche	267 km/h	wütend	22 km	y_n

sich herausstellen, dass männliche Fahrer in Sportwagen generell jähzorniger sind als weibliche Fahrerinnen in Kleinwagen – ohne, dass man dies zuvor explizit erwartet hätte.

Im semi-überwachten Lernen existieren zu manchen Datenpunkten (x_i, y_i) Outputwerte y, aber für manche nicht. Dies wäre beispielsweise der Fall, wenn die Datensammlung nicht lange genug gegangen ist, um für alle beobachteten Autos die Fahrdistanz bis zum nächsten Unfall zu beobachten.

10.3.1 Überwachtes Lernen

Beim überwachten Lernen wird generell angenommen, dass y durch die Prädiktoren X = (x_1, x_2, ..., x_n) beschrieben werden kann. Dabei soll es eine wahre Funktion f geben, die y in Abhängigkeit von X darstellt:

$$y = f(X)$$

Ziel des überwachten Lernens ist es eine Funktion h zu finden, welche die wahre Funktion f möglichst gut approximiert – das intelligente System soll f lernen, indem verschiedene Kandidaten-Funktionen untersucht werden. Diese Form des Lernens wird auch *induktives Lernen* genannt: aus beobachteten Phänomenen sollen allgemeine Regeln abgeleitet werden, durch die die Beobachtungen erklärt werden können. Da die wahre Funktion f im Allgemeinen unbekannt ist und nicht bestimmt werden kann, ist man in der Praxis häufig daran interessiert eine solche Funktion h zu bestimmen, die y möglichst fehlerfrei vorhersagt. Zu diesem Zweck werden die untersuchten Funktionen an einem Trainingsdatensatz trainiert und die Vorhersagegüte wird dann an einem unabhängigen Testdatensatz überprüft. Ziel ist es, den Testdatensatz anhand der Passung, die am Trainingsdatensatz erstellt wurde, möglichst gut vorherzusagen. Ob die Funktion h der wahren Funktion f ähnelt, ist zumeist gar nicht so wichtig solange sie akkurate Vorhersagen bezüglich der Zustände der Outputvariablen macht. In Klassifikationsproblemen kann die Vorhersagegüte einfach durch den prozentualen Anteil aller falschen Zuordnungen angegeben werden. Im Fall binärer Entscheidungen kann die Korrektheit von Entscheidungen mit einer 2x2 Tabelle dargestellt werden:

Tabelle 10.2: Entscheidungsmöglichkeiten in einer binären Klassifikationsaufgabe.

	Status: Unfall	Status: kein Unfall
Vorhersage: Unfall	richtige Klassifikation	falsche Klassifikation
Vorhersage: kein Unfall	falsche Klassifikation	richtige Klassifikation

In Regressionsproblemen ist die einfache Vorhersage-Trefferrate kein sinnvolles Gütekriterium für die Passung einer Funktion. Angenommen, h bestimmt für einen Fahrer eine unfallfreie Fahrt von 1,237.47 Kilometer, das Auto hat den nächsten Unfall aber nach 1,237.48 km – die Vorhersage liegt sehr nah am tatsächlichen Wert, ist aber nicht korrekt. Ist diese Abweichung als problematisch zu betrachten? Wäre eine Vorhersage von 20,000 km „gleich falsch“? Es macht Sinn in diesem Fall den Schweregrad der Abweichung der Vorhersage zu berücksichtigen. Bei

kontinuierlichem Output wird deswegen der Vorhersagefehler durch die Abweichungen der vorhergesagten von den tatsächlichen y-Werten bestimmt. Dabei wird häufig die mittlere quadratische Abweichung (*mean squared error*; MSE) als Fehlermaß verwendet.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - h(x_i))^2$$

Hierbei sind y_i die beobachteten Output-Werte und $h(x_i)$ die vorhergesagten Werte auf Basis der zugehörigen Input-Werte.

Die Funktion h sollte so gewählt werden, dass die Vorhersagefehler gering gehalten werden. Tatsächlich wird aber bei der Wahl von h nicht nur die Fehlerrate minimiert, sondern die **Kosten** verschiedener Fehlentscheidungen können ebenfalls berücksichtigt werden. Nicht alle Fehleinschätzungen sind gleich problematisch: eine starke Unterschätzung der verbleibenden unfallfreien Zeit wäre beispielsweise fataler als eine Überschätzung, da im ersteren Fall mögliche Interventionsmaßnahmen zu spät ergriffen würden. Im binären Fall wäre analog die falsche Vorhersage schlimmer, dass kein Unfall stattfindet, obwohl ein Unfall stattfinden wird (siehe die zwei Fehlertypen in Tabelle 2.2). Demnach muss ein solches h gesucht werden, welches einerseits gute Vorhersagen erlaubt, und andererseits die Kosten möglicher Fehler minimiert.

Wie aber wird bestimmt, welche Funktionen h untersucht werden als Schätzer für f? Der Hypothesenraum H definiert, welche Funktionen als Kandidaten für f untersucht werden. Ein Lernproblem wird realisierbar genannt, wenn H die wahre Funktion f enthält. Der Hypothesenraum H wird nach Funktionen durchsucht, die gut mit den Daten übereinstimmen. Eine Hypothese (d.h. eine hypothetisierte Funktion) heißt konsistent, wenn sie alle Trainingsdaten korrekt beschreibt. Sie heißt generalisierbar, wenn sie auch für Testdaten eine gute Vorhersage erlaubt. Man kann sinnvollerweise annehmen, dass ernsthaft falsche Hypothesen bereits nach der Beobachtung von nur wenigen Datenpunkten stark von der Realität abweichende Vorhersagen machen, sodass diese schnell ausgeschlossen werden können.

10.3.2 Lineare Regression

Im Falle einer Regressionsentscheidung könnte der Hypothesenraum aus der Klasse der linearen Polynome bestehen, das heißt Funktionen der Form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$$

Hier wird der Output y durch eine Linearkombination der Inputvariablen x_i dargestellt. β_i sind Koeffizienten, die so gewählt werden, dass die quadratische Abweichung der vorhergesagten Daten y von den tatsächlichen Daten möglichst wenig abweicht. Dieses Verfahren wird **Methode der kleinsten Quadrate** genannt. Man kann zeigen, dass die Minimierung der quadratischen Abweichungen zu einer eindeutigen Lösung der Koeffizienten β_i führt. Für die Datenmatrix $X = [1, x_1, x_2, .., x_n]$, wobei x_i ein Vektor ist, der die Werte der i'ten Prädiktorvariablen darstellt, und $\beta = [\beta_0, \beta_1, \beta_2, ..., \beta_n]$, ist diese Lösung gegeben durch

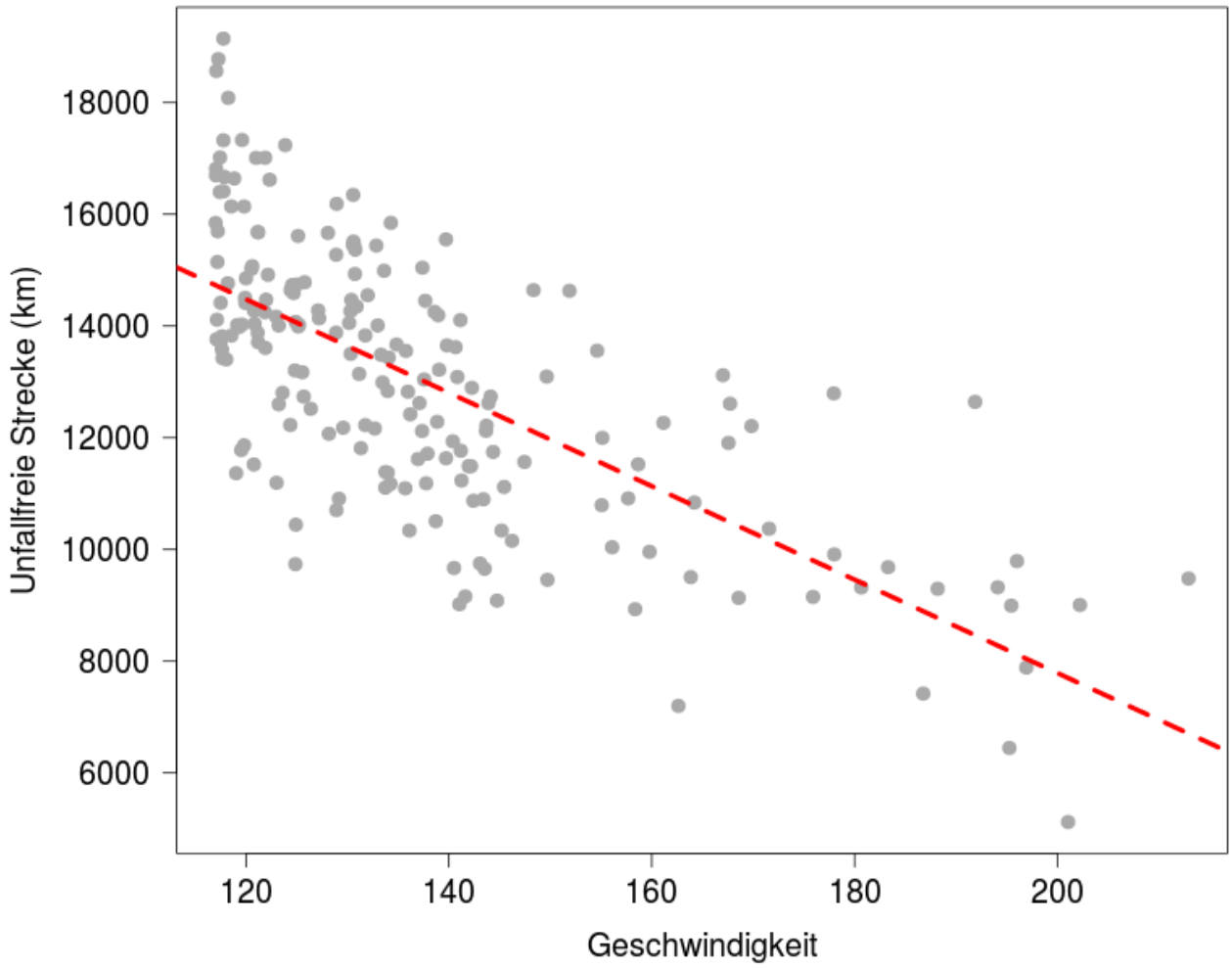
$$\beta = (X^T X)^{-1} X^T y$$

Für Menschen sind lineare Zusammenhänge gut zu verstehen und mathematisch sowie von der Rechenintesität einfach durchzuführen. Lineare Zusammenhänge können in der Weise „Wenn Variable x steigt, steigt/sinkt Variable y" interpretiert werden. Komplexere, nicht-lineare Verfahren wie Support-Vektor-Maschinen und Neuronale Netze schneiden in ihrer Vorhersagegüte häufig besser ab als einfache lineare Regression, sind aber rechenintensiver und für Menschen weniger leicht interpretierbar. Dies ist für ein weit entwickeltes intelligentes Computersystem aber gegebenenfalls kein Problem mehr – sofern Menschen dem Urteilvermögen des Systems vertrauen.

Wir könnten die Hypothese aufstellen, dass zwischen der beobachteten Geschwindigkeit eines Autos und der zu erwartenden unfallfreien Fahrstrecke ein negativer linearer Zusammenhang besteht – schnellere Fahrer könnten nach weniger gefahrenen Kilometern einen Unfall bauen. Dies ist ein Fall von **univariater Regression**: der Output wird vorhergesagt durch nur eine Prädiktor-Variable. Grafik 2.1 stellt einen möglichen Zusammenhang der beiden Variablen dar.

Es zeigt sich, dass schnelle Autofahrer tendenziell bereits nach einer kürzeren Strecke einen Unfall haben. Die rote Gerade zeigt hierbei die vorhergesagten Distanzwerte an, die allein durch Kenntnis der Variable Geschwindigkeit zu erwarten sind. Die Gerade wird beschrieben durch die Gleichung:

Abbildung 10.1: Ein linearer Zusammenhang zwischen Geschwindigkeit und der Länge der Strecke einer weiteren unfallfreien Fahrt.



$$y = 24,516.56 - 83.65 \cdot \textit{Geschwindigkeit}$$

Das lineare Modell sagt somit pro 1 km/h schnelleres Fahren etwa 84 weniger unfallfrei gefahrene Kilometer vorher. Es ist jedoch deutlich zu erkennen, dass allein das Wissen über die Geschwindigkeit keine allzu präzise Aussage über die unfallfreie Strecke ermöglicht – im Mittel liegt im vorliegenden Fall die absolute Abweichung zwischen vorhergesagten und tatsächlichen Werten bei 1401 km. Ein intelligentes System sollte bessere Prognosen erstellen, wenn seine Aufgabe die Überwachung der Verkehrssicherheit ist! Um die Vorhersagegüte zu erhöhen, bieten sich zwei Möglichkeiten an: Erstens können wir mehr aussagekräftige Prädiktoren in unser Regressionsmodell aufnehmen. In diesem Fall würden wir das Modell zu einem **multivariaten Regressionsmodell** erweitern. Zweitens könnten wir die Komplexität der Modellgleichung erweitern, um auch nicht-lineare Schwankungen in den Daten zu beschreiben.

10.3.3 Nicht-lineare Regression

Lineare Regressionsverfahren können erweitert werden, um auch nicht-lineare Zusammenhänge zu beschreiben. Somit ergibt sich eine höhere Flexibilität, Schwankungen in Daten abzubilden. Um nicht-lineare Zusammenhänge mithilfe der *linearen* Regression zu beschreiben, wird die Modellgleichung als Polynom höherer Ordnung aufgestellt. Dabei bleibt die Gleichung linear in den Koeffizienten, jedoch die Prädiktorvariablen gehen mit bis zu d-wertigen Polynomgrad in das Regressionsmodell ein. Für ein Polynom d-wertiger Ordnung mit einer Prädiktorvariablen x sieht in die Regressionsgleichung somit wie folgt aus:

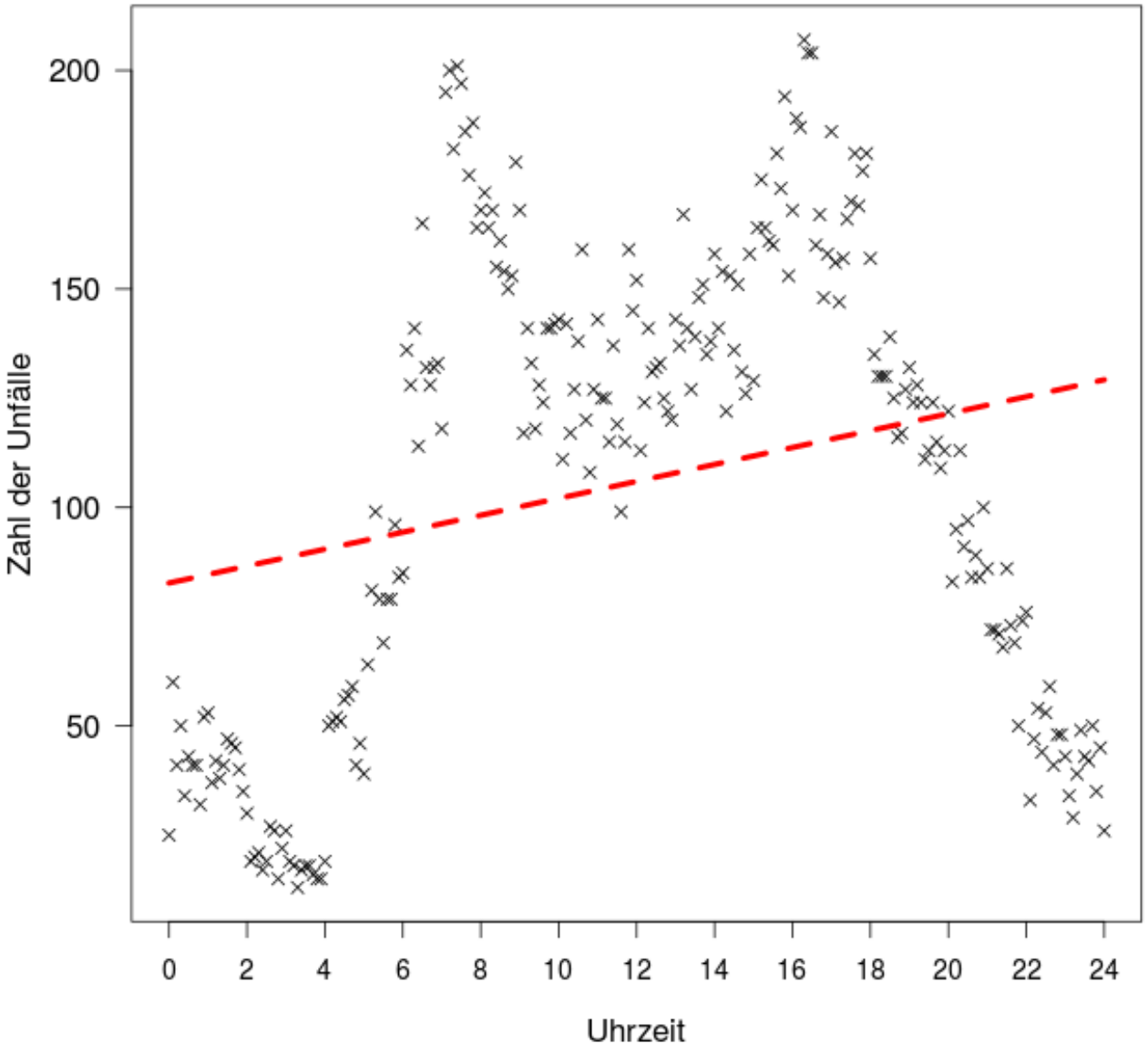
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_d x^d$$

Die allgemeine Koeffizientenlösung gilt auch für höherwertige Polynome, wobei in diesem Fall für die Datenmatrix gilt X = $[1, x, x^2, ..., x^d]$, mit $\beta = [\beta_0, \beta_1, \beta_2, ..., \beta_d]$:

$$\beta = (X^T X)^{-1} X^T y$$

Angenommen unser System möchte seine Vorhersagegenauigkeit verbessern und betrachtet weitere Variablen, die mit dem Auftreten von Verkehrsunfällen auftreten können. Es stellt fest, dass die Häufigkeit von Unfällen mit der Tageszeit zusammenhängt (siehe Grafik 2.2).

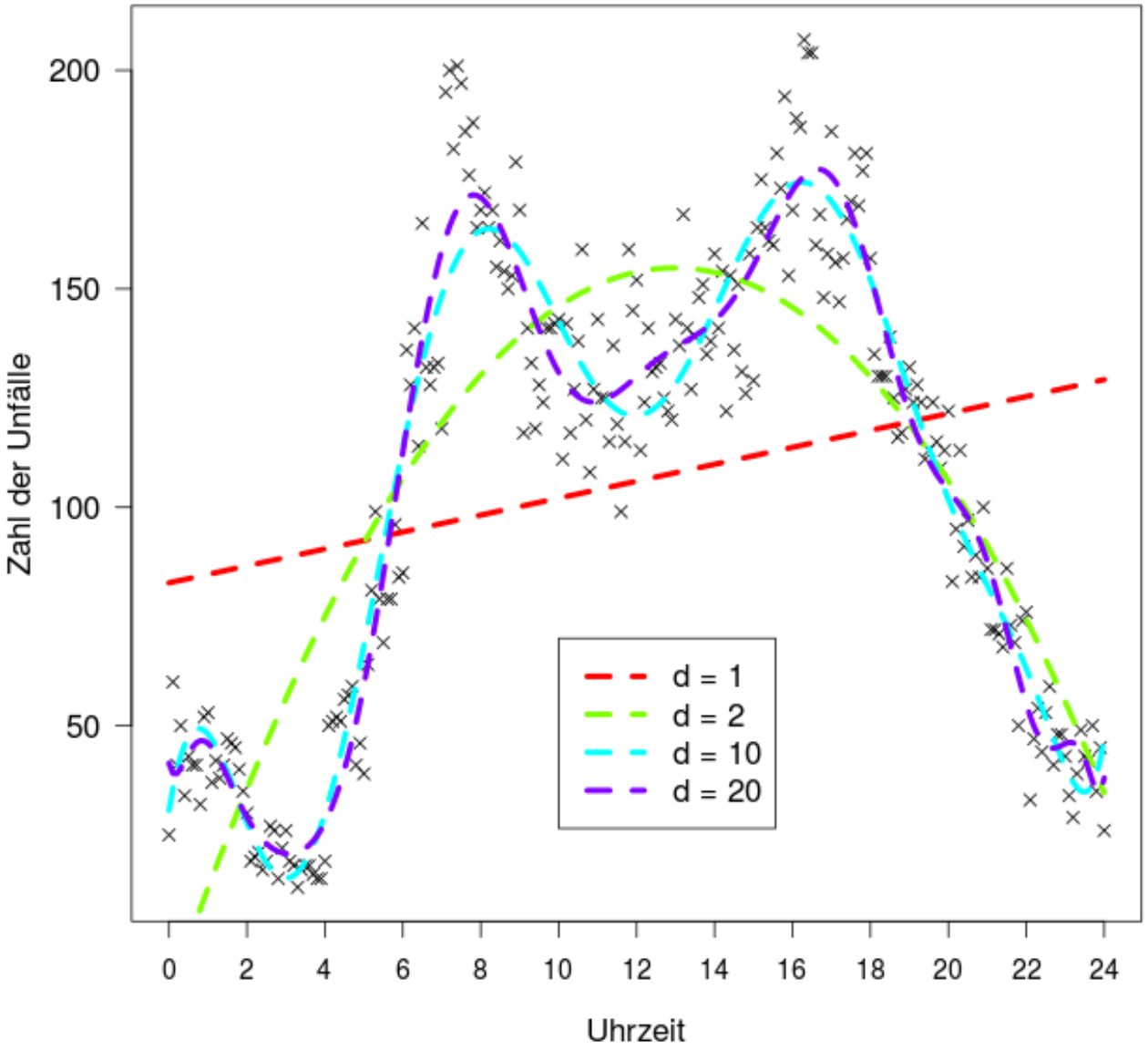
Abbildung 10.2: Der Zusammenhang zwischen Uhrzeit und der Häufigkeit von Autounfällen pro Jahr. Die rote Linie stellt den Schätzer der Unfallhäufigkeit durch eine einfache lineare Regression dar.



Ein einfaches lineares Regressionsmodell ergibt einen geringen positiven Zusammenhang zwischen Tageszeit und Zahl der Unfälle. Das Modell sagt somit die größte Unfallwahrscheinlichkeit für 23:59 Uhr vorher und die geringste für 00:00 Uhr, was eine sehr unplausible Vorhersage ist. Eine visuelle Inspektion der Daten ergibt schnell,

dass dieser einfache lineare Zusammenhang die Daten nicht adäquat beschreibt. Eine Systematik in den Daten geht augenscheinlich darauf zurück, dass sich nachts weniger Unfälle ereignen als tagsüber. Außerdem scheinen sich tagsüber während der Zeit des Berufsverkehrs noch einmal etwas mehr Unfälle zu ereignen. Ein Polynom zweiter Ordnung ist bereits in der Lage den U-förmigen Zusammenhang zwischen Tageszeit und Unfällen darzustellen, der durch die Reduktion der Unfälle nachts bedingt wird. Ein Polynom des 10. oder 20. Grades kann die Schwankungen abzubilden, die auf den Berufsverkehr zurückgehen (siehe Grafik 2.3).

Abbildung 10.3: Der Zusammenhang zwischen Uhrzeit und der Häufigkeit von Autounfällen pro Jahr. d ist der Grad der Polynomgleichung, die zur Vorhersage der Unfallhäufigkeit verwendet ist.



Ist also anzunehmen, dass komplexere Modelle besser sind, da sie Daten genauer beschreiben können und bessere Vorhersagen treffen? Sollte man in einer polynomialen Regression immer anstreben den höchst-möglichen Polynomgrad zu modellieren, um die Daten möglichst präzise abzubilden? Es gibt mehrere Gründe, die dagegen sprechen: Erstens ist nicht zu erwarten, dass jegliche Erweiterung in der Komplexität auch wirklich zu einer merklich verbesserten Vorhersageleistung führt. In dem Fall besagt das philosophische Prinzip *Ockhams Razor*, dass, wenn verschiedene Hypothesen Daten gleich gut beschreiben, diejenige gewählt werden sollte mit, welche die

wenigsten Annahmen macht. Ockhams Razor ist ein allgemeines Prinzip, welches nicht nur in Regressionsproblemen Anwendung findet. In Grafik 2.3 ist zu erkennen, dass ein Polynom 2. Grades die Daten schon wesentlich besser abbildet als der einfach lineare Zusammenhang. Das Polynom 10. Grad ist wiederum ebenfalls eine Verbesserung, da es Schwankungen, die auf den Berufsverkehr zurückgehen, abbildet. Es ist jedoch fraglich, ob die Erhöhung der Komplexität von 10 auf 20 Polynomgrade wirklich notwendig ist, um die Daten adäquat abzubilden. Nach Ockhams Razor wäre in dem Fall das weniger komplexe Modell zu bevorzugen.

Des Weiteren besteht bei der Annahme eines zu komplexen Modells die Gefahr des sogenannten **Overfittings**. Dieses bezeichnet den Umstand, wenn ein Modell zwar vorgegebene Daten sehr gut beschreibt, aber nicht auf neue Testdaten generalisiert, d.h. für solche Daten schlechte Vorhersagen macht, die nicht verwendet wurden, um das Modell zu trainieren. Es ist wichtig, dass ein Modell auch für neue Datenpunkte Aussagen treffen kann. Für unser Fahrsicherheitssystem wäre es fatal, wenn es zwar alte Unfalldaten gut rekonstruieren kann, aber nicht zukünftige Unfallgefahren voraussagen kann – in dem Fall wäre es sogar vollends unnötig dieses System überhaupt zu unterhalten.

Um bei der Erstellung eines Modells nicht auf Daten warten zu müssen, die erst in der Zukunft (oder vielleicht niemals) gesammelt werden können, kann man eine sogenannte **Kreuzvalidierung** durchführen: das vorhandene Datenset wird aufgeteilt in einen **Trainings-** und einen **Testdatensatz**. Dies geschieht meistens, indem ein bestimmter Prozentsatz der Daten zufällig dem Trainigsset und die anderem dem Testset zugeteilt werden; eine Aufteilung von 80% zu 20% ist zum Beispiel üblich. Anhand der Trainingsdaten werden die Regressionskoeffizieten geschätzt, entscheidend für die Beurteilung eines Modells ist aber seine Fähigkeit die Testdaten abzubilden. Dies ist ein allgemeines Prinzip, welches nicht nur bei linearer Regression Anwendung findet. Im Falle einer Regression werden die Koeffizienten so gewählt, dass sie den Trainingsdatensatz maximal präzise beschreiben. Durch Annahme höherer Polynomgrade wird die Passung an den Trainingsdatensatz verbessert, im Testdatensatz verschlechtert sich die Passung jedoch ab einem bestimmten Komplexitätsgrad: ab hier geschieht Overfitting. Grafik 2.4 zeigt für den Uhrzeit–Unfallhäufigkeit Datensatz die Modellpassung verschiedener Polynomgrade anhand einer Kreuzvalidierung. Es zeigt sich, dass Trainingsdaten mit steigendem Polynomgrad besser vorhergesagt werden. Im Testdatensatz ist bei sehr hoher Komplexität jedoch wieder eine Verschlechterung der Passung zu sehen. Beachtlich ist hierbei (sowohl für Trainings- als auch Testdatensatz) die Verbesserung, die erzielt wird, wenn der Polynomgrad von 1 auf 2 erhöht wird.

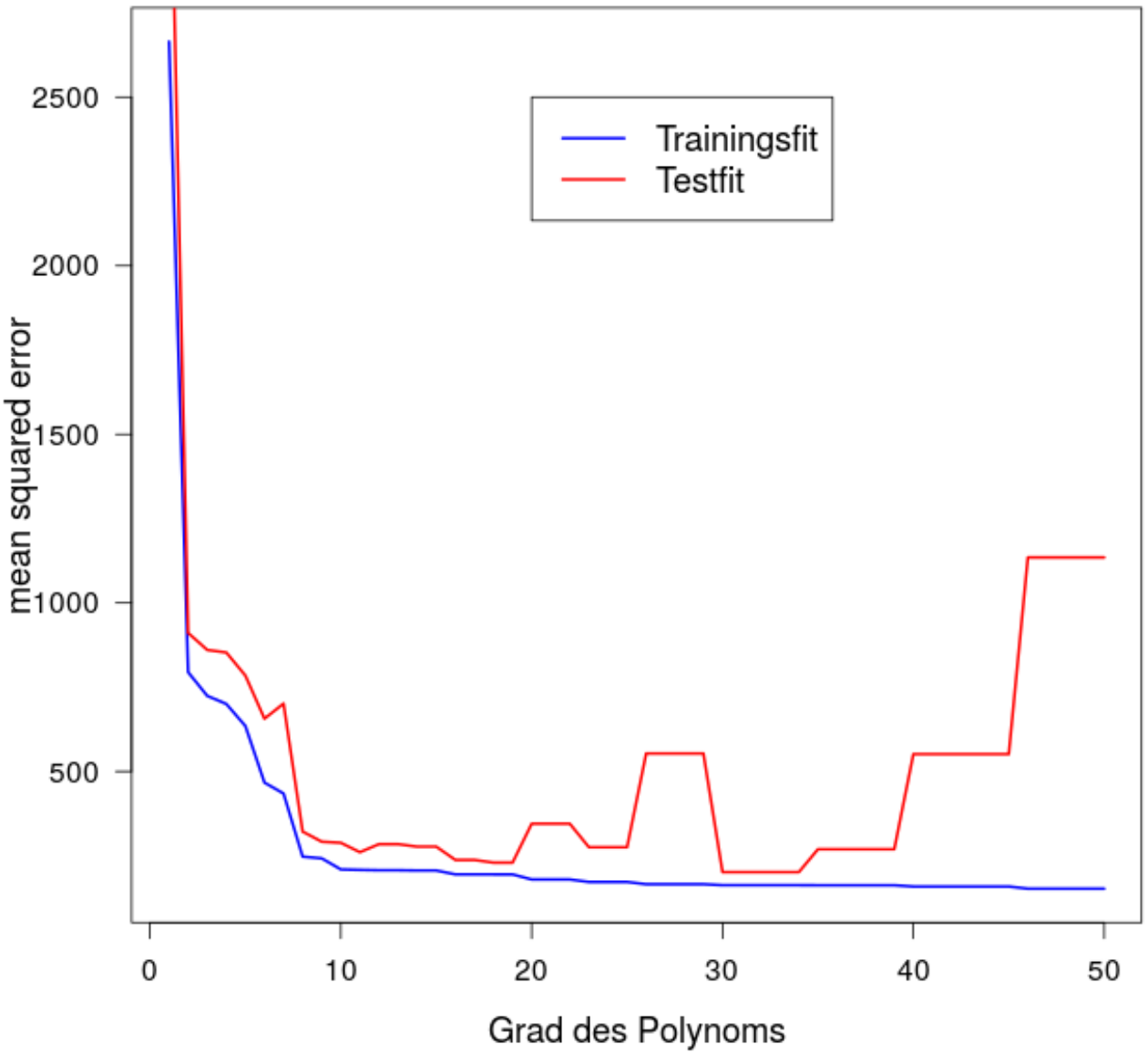
10.3.4 Klassifikation

Sehr häufig müssen in der Praxis qualitative Outcome-Variablen vorhergesagt werden. Unser Fahrsicherheitssystem könnte beispielsweise der sehr relevanten Frage nachgehen, unter welchen Umständen Verkehrsunfälle tödlich ausgehen. Beispielsweise könnte es betrachten, ob durch Kenntnis des Gewichts und der Geschwindigkeit eines Autos vorausgesagt werden kann, ob ein Unfall tödlich ausgeht. Grafik 2.5 zeigt für 100 verschiedene in Unfälle involvierte Autos das Gewicht und die Geschwindigkeit, und ob der Unfall tödlich verlaufen ist. Es zeigt sich, dass eine höhere Geschwindigkeit tendenziell mit tödlichem Ausgang einhergeht. Zu einem geringeren Maße hängt auch das Gewicht mit der Tödlichkeit des Ausgangs zusammen. Ziel einer Klassifikation ist eine Funktion zu finden, die auf Basis der Input-Werte das Label schätzen kann. Hierbei wird eine Entscheidungsgrenze („decision boundary“) angewendet; wenn der vorhergesagte Funktionswert über dieser Grenze liegt, wird eine Klasse vorhergesagt, andernfalls wird die andere Klasse vorhergesagt. Wenn es möglich ist die Klassen durch eine lineare Gleichung eindeutig voneinander zu trennen – wie in Grafik 2.5 der Fall – nennt man das Problem **linear separabel**. Verfahren, die zur Lösung von Klassifikationsproblemen verwendet werden, sind unter anderem k nearest neighbor, lineare Diskriminanz Analyse, logistische Regression Support Vector Machines oder neuronale Netze. Dabei können Support Vector Machines und neuronale Netze sogar nicht-lineare Entscheidungsgrenzen abbilden.

10.4 Fazit & Bewertung

Methoden des statistischen Lernens können verwendet werden, um Computersystemen das Lernen aus Beobachtungen zu ermöglichen. Die Fähigkeit zu lernen ist eine notwendige Voraussetzung für maschinelle Systeme intelligentes Verhalten zu zeigen. Systeme können aus großen Datenmengen Informationen integrieren, um zukünftige Ereignisse oder Verhalten vorherzusagen. Maschinelles Lernen handelt sich um ein stark wachsendes Forschungsfeld, das auch in der Öffentlichkeit mit viel Interesse verfolgt wird – nicht zuletzt aufgrund kürzlicher, medienwirksamer Errungenschaften intelligenter Systeme, beispielsweise dem Erfolg des Systems AlphaGo, welches

Abbildung 10.4: Das Ergebnis einer Kreuzvalidierung des Uhrzeit – Unfallhäufigkeit Datensatzes. Gezeigt ist der Verlauf des mittleren Voraussagefehlers nach Komplexität des modellierten Polynoms für Trainings- und Testdatensatz.

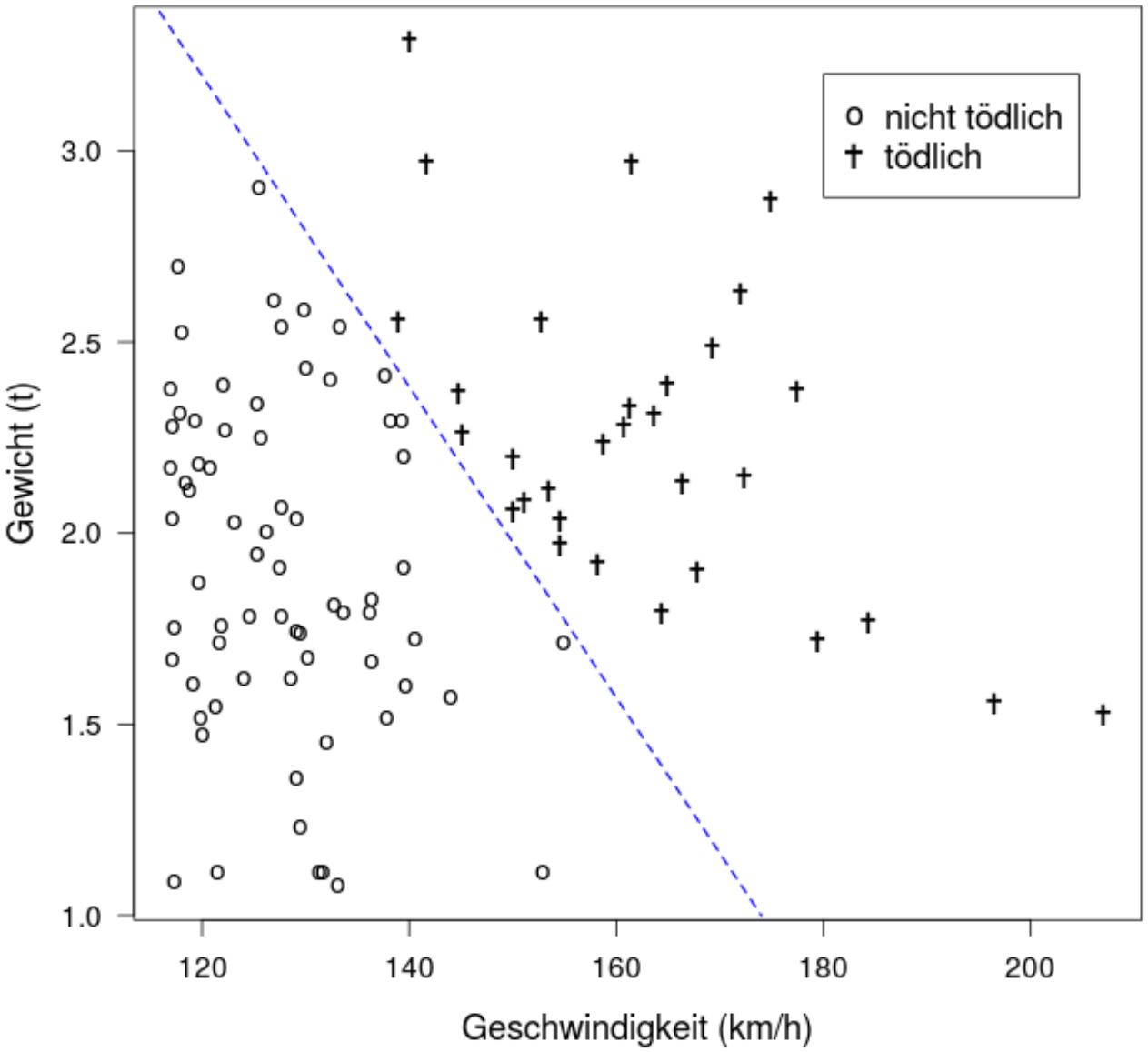


den Weltmeister Lee Sedol im Go-Spiel überraschend bezwang. Lineare Regression wurde als wichtiger Prototyp des statistischen Lernens ausführlicher vorgestellt. Dabei handelt es sich um ein einfaches Verfahren, welches bereits gute Voraussagen liefert und auch nicht-lineare Zusammenhänge darstellen kann. Auf fortgeschrittenere Klassifikationsverfahren wie Support Vector Machines und neuronale Netze wurde ein Ausblick gegeben. Einige grundsätzliche Prinzipien wie Ockams Razor und Kreuzvalidierung gelten für jegliche Verfahren des statistischen Lernens, egal wie komplex oder einfach sie erlernt werden können.

10.5 Quellen und Literatur

- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*. New York: Springer.
- Legg, S., & Hutter, M. (2007). A collection of definitions of intelligence. *Frontiers in Artificial Intelligence*

Abbildung 10.5: Die Klassifikation eines Autounfalls als tödlich oder nicht tödlich in Abhängigkeit des Gewichts und der Geschwindigkeit des Autos. Die blaue Gerade ist eine sogenannte Entscheidungsgrenze („decision boundary“).



and applications, 157, 17.

- Lawrence N. (2016, May 09). Future of AI 6. Discussion of “Superintelligence: Paths, Dangers, Strategies” [Web log post]. Retrieved from http://inverseprobability.com/2016/05/09/machine-learning-futures-6
- Russell, S., & Norvig, P. (2010) *Artificial Intelligence: A modern approach (3rd. ed.)*. Upper Saddle River: Pearson.

11. Neuronale Netze

11.1 Intro & Motivation

Heutzutage sind die Rechner so schnell und die Algorithmen so effizient, dass unsere Computer in der Lage sind, die kompliziertesten Berechnungen innerhalb von ein paar Millisekunden durchzuführen. Und jedoch scheitern sie, wenn es um so triviale Aufgaben wie zum Beispiel die Gesichts- oder Stimmerkennung geht. Für einen Menschen ist es aber gar kein Problem die Gesichter, sogar unter erschwerten Bedingungen (zu dunkel, zu nebelig, usw.), zu erkennen.

Woran kann das liegen? Warum ist es für einen Computer so einfach komplexe Mehrteilchensysteme zu berechnen und wiederum so schwer, auf einem unscharfen Bild ein paar Gesichter zu erkennen, obwohl diese Aufgabe sogar für ein kleines Kind keine Herausforderung ist?

Die Antwort auf diese Fragen ist ganz einfach: Solche Aufgaben (sowie viele andere) hängen von einer Menge Faktoren ab, wie zum Beispiel die Bild- oder Aufnahmequalität, und es ist leider unmöglich einen Algorithmus zu entwickeln, der alle Details berücksichtigt. Zusätzlich ist das menschliche Gehirn im Vergleich zu den Rechnern nicht nur in der Lage die Informationen zu empfangen und sie zu speichern, wie das die Rechner tun, sondern auch diese Daten gleichzeitig zu verarbeiten, zu analysieren und aus dieser Analyse zu lernen. Deswegen, bevor man versucht dieses Problem zu lösen, muss man sich zuerst fragen, wie es unserem Gehirn gelingt, solche Aufgaben zu bewältigen.

Dabei sollte man auch erwähnen, dass unser Gehirn fast sein komplettes Potenzial ausnutzt, indem es ständig und parallel arbeitet. Es ist bekannt, dass es fast unmöglich ist, die echte Parallelität zwischen den Prozessen in einem Computer zu erreichen. Und das ist noch ein Grund, zu fragen, nach welchen Prinzipien unser Gehirn funktioniert und wie es die Informationen verarbeitet.

11.2 Inhaltliche Ausarbeitung des Themas

11.2.1 Zurück in die Vergangenheit

Die ersten Schritte in diesem Gebiet haben Warren McCulloch und Walter Pitts gemacht, indem sie im Jahr 1943 „A logical calculus of the ideas immanent in nervous activity“ veröffentlicht haben. In diesem Werk untersuchen die Wissenschaftler die Fähigkeiten Neuronaler Netze und präsentieren einen einfachen durch Neuronen nachgebauten Schwellwertschalter, der in der Lage ist, fast jede logische und arithmetische Funktion zu berechnen.

Im Jahr 1947 wurde von Walter Pitts und Warren McCulloch vorgeschlagen, dass Neuronale Netze zur räumlichen Mustererkennung eingesetzt werden könnten.

Ein wichtiger Schritt wurde im Jahr 1949 von Donald Olding Hebb gemacht. In seinem Werk „The Organization of Behaviour“ formulierte er die klassische Hebb'sche Lernregel, welche in ihrer allgemeinen Form die Basis fast

aller neuronalen Lernvervahren darstellt.

Weitere neuropsychologische Arbeiten erschufen den Weg für spätere erfolgreiche Forschungen in diesem Gebiet.

11.2.2 Problemstellung

Neuronale Netze basieren auf der Ähnlichkeit zu den menschlichen Neuronennetzen. Damit ein Verständnis über den Aufbau und die Funktion der neuronalen Netze erreicht werden kann, muss die Funktionsweise der echten Neuronen verstanden sein.

Biologische Grundlagen

Unser Gehirn besteht größtenteils aus Nervenzellen, die auch Neuronen genannt werden. Eine typische Nervenzelle eines Säugetiers besteht hauptsächlich aus drei wichtigen Teilen (siehe Abb. 2.1): dem Zellkörper (Soma) mit Zellkern, dem Axon (Nervenfaser) mit Synapsen und den Dendriten (Zelleingängen).

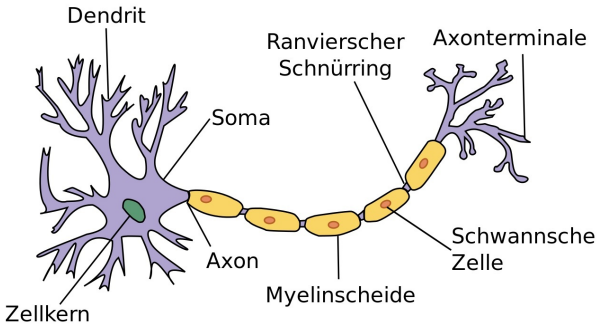


Abbildung 11.1: Schematischer Aufbau eines Neurons (Quelle: Wikipedia)

Dendriten sind baumartige Eingänge der Nervenzelle und dienen der Aufnahme elektrischer Reize von anderen Nervenzellen und ihrer Übertragung in den Zellkern. Aufgrund ihrer Form werden mehrere Dendriten auch als Dendritenbaum bezeichnet. Die Länge der Dendriten beeinflusst die Stärke des ausgelösten Signals: je kürze ist ein Dendrit, desto stärker ist das Signal. Diese Signale werden an den Zellkörper geleitet, wo sie sich aufsummieren.

Das Axon einer Nervenzelle kann man als „Ausgang“ des Neurons bezeichnen, weil es elektrische Signale weiterleitet, die durch Ladungsunterschiede (elektrischen Potential) zwischen der Umgebung und Zellinnerem entstehen. Wenn das aufsummierte Erregungspotenzial eine bestimmte Schwelle übersteigt, wird ein Aktionspotential fester Stärke ausgelöst, dass über das Axon und den Dendriten weitergeleitet wird.

Am Ende des Axons befinden sich die Verdickungen, die auch Synapsen genannt werden (siehe Abb. 2.2). Synapsen sind mit den Dendriten anderer Neuronen verbunden. Wobei es keine direkte physikalische Verbindung, sondern einen mit Flüssigkeit gefüllten Zwischenraum gibt. Erreicht ein über dem Schwellenwert liegendes Aktionspotential die Synapse, werden sogenannte Neurotransmitter ausgeschüttet. Diese Neurotransmitter wandern durch den synaptischen Spalt und erreichen die Rezeptoren des Dendriten. Die Neurotransmitter können an den Rezeptoren binden. Entweder führt das Anbinden der Neurotransmitter zu einer Reizweiterleitung bzw. Reizverstärkung in den Dendriten oder zu einer Hemmung des Aktionspotentials, sodass die Reizweiterleitung zum Erliegen kommt. Die durchschnittliche Anzahl an solchen Verknüpfungen beträgt ca. 14 000 pro Neuron.

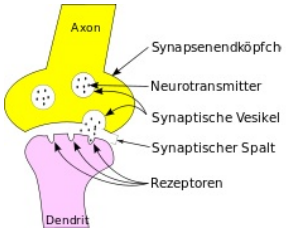


Abbildung 11.2: Synapsenkommunikation (Quelle: Wikipedia)

Der Zellkörper ist der Hauptbestandteil eines Neurons. Er ist nicht nur für die Energieproduktion verantwortlich, sondern auch für die Verarbeitung und Aufsummierung der eingehenden Signale sowie der Erzeugung bzw.

Weiterleiitung der Ausgangssignale.

Die neurologischen Kenntnisse über den Aufbau menschliches Gehirns bilden die Basis für die Entwicklung mathematischer Modelle, die auch „Neuronale Netze“ genannt werden.

11.3 Aufbau und Funktionsweise künstlicher Neuronaler Netze (KNN)

Jedes künstliche neuronale Netz hat drei Grundelemente: Neuronen, die Verbindungen zwischen den Neuronen und Lernregeln.

11.3.1 Neuronen

Ähnlich den Nervenzellen im menschlichen Gehirn sind die künstlichen „Neuronen“ die wichtigsten Bestandteile eines KNNs. Sie besitzen eine ähnliche Struktur mit vielen Ein- und Ausgängen für die Weiterleitung von ein- und ausgehenden Signalen sowie einen Körper zur Verarbeitung dieser Signale. Schematisch könnte ein künstliches Neuron folgendermaßen aussehen (siehe Abb. 3.1):

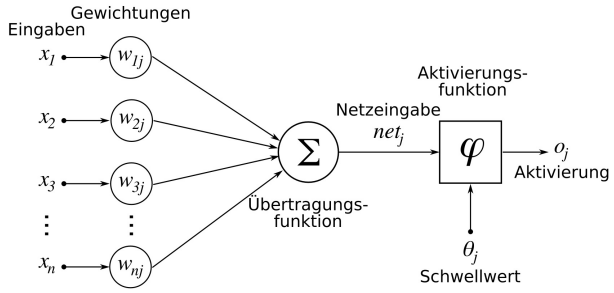


Abbildung 11.3: Schematische Darstellung der wichtigsten Bestandteile eines künstlichen Neurons (Quelle: Wikipedia)

Die Funktion der Dendriten erfüllen die Eingaben x_i, wobei i = 1, . . . , n. Beim Modellieren eines KNN sollte man nicht vergessen, dass die Länge eines Neurons eine große Rolle bei der Bestimmung der Signalstärke spielt. Dies wird durch die Gewichtungen w_{ij} dargestellt. Die Gewichtung wird mit dem Eingabewert multiplikativ verknüpft. Es gibt verschiedene Vorgehenstechniken bei der Belegung der Gewichtungen, aber häufig werden dafür zufällige Werte in einem bestimmten Wertebereich ausgesucht. Je nachdem, ob die Gewichtung einen positiven oder einen negativen Wert hat, wirkt der Transmitter entweder verstärkend oder hemmend genau so wie bei den Menschen. Falls die Gewichtung gleich 0 ist, hat ein Neuron auf ein anderes Neuron keinen Einfluss.

Die Übertragungsfunktion könnte man sich als Erregungspotenzial vorstellen. In dieser Funktion, genauso wie die Signale im Zellkern, werden die Eingaben entsprechend ihrer Gewichtungen aufsummiert und zu einem Wert umgerechnet, damit man danach eine reellwertige Funktion darauf anwenden kann. Der Schwellenwert θ_j dient als Schwellenpotenzial. Das Addieren eines Schwellwerts zur Netzeingabe verschiebt die gewichteten Eingaben. Die Aktivierungsfunktion ϕ berechnet aus dem aktuellen Aktivierungszustand und der Netzeingabe den neuen Aktivierungswert. Dieser Wert wird dann weitergeleitet. Die Aktivierungsfunktion kann wie die echten Neuronen dem Alles-oder-gar-nichts-Prinzip folgen. Dabei wird bei der Überschreitung des Schwellenpotenzials wie bei menschlichen Neuronen ein festes Aktionspotential generiert. So eine Aktivierungsfunktion heißt auch Sprungfunktion (siehe Abb. 3.2).

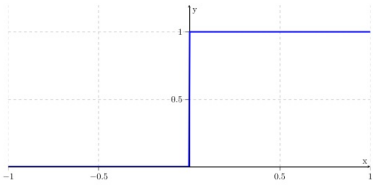


Abbildung 11.4: Eine Sprungfunktion (Quelle: Wikipedia)

Eine Aktivierungsfunktion kann aber auch nach den anderen Prinzipien funktionieren. Häufig wird eine sogenannte Sigmoidfunktion (siehe Abb. 3.3) mit einer ungleichmäßigen Abhängigkeit zwischen der Ein- und

Ausgabe verwendet. Weitere mögliche Aktivierungsfunktionen sind lineare oder stückweise lineare Funktionen (siehe Abb. 3.4). In solchen Funktionen hängt die Ausgabe linear von der Eingabe ab.

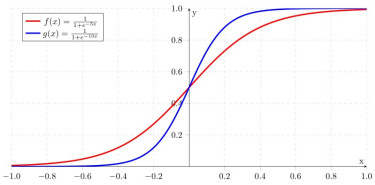


Abbildung 11.5: eine Sigmoidfunktion mit Steigerungsmaß (Quelle: Wikipedia)

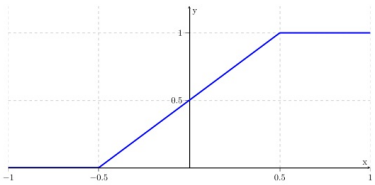


Abbildung 11.6: eine stückweise lineare Funktion (Quelle: Wikipedia)

Die Aktivierung o_j ist das Ergebnis einer Aktivierungsfunktion und dient als „Ausgang“ eines künstlichen Neurons.

11.3.2 Verbindungen und Netzwerktopologien

Durch den Verbund von mehreren geordneten Kanten zwischen den Verarbeitungseinheiten entsteht ein KNN. Dabei wird die räumliche Anordnung dieser Einheiten als Netzwerktopologie bezeichnet. Eine der wichtigsten Aufgaben der Verbindungen ist es die Architektur eines KNN festzulegen. In manchen Netzen entstehen durch solche Kanten sogenannte Schichten.

Die Verbindungen zwischen den Einheiten sind für die Kommunikation zwischen Neuronen verantwortlich: über diese Kanten werden die Daten ausgetauscht, wobei jede Verbindung durch einen Gewichtswert realisiert wird. Dabei unterscheidet man zwischen den unidirektionalen und bidirektionalen Verbindungen.

Durch unidirektionale Inter-Layer-Verbindungen werden die Daten der Eingabeschicht zur Ausgabeschicht weitergeleitet. Dadurch entstehen die sogenannten Feed-Forward-Netze, die man sich als einen azyklischen Graphen vorstellen kann.

Bidirektionale Verbindungen bilden die Feed-Backward-Netze. Bei den FB-Netzen unterscheidet man drei Typen:

a) later feedback: bidirektionale Verbindungen entstehen zwischen den Knoten einer Verarbeitungsschicht;

b) direct feedback/self-feedback: es entsteht eine unidirektionale Verbindung einer Einheit mit sich selbst;

c) indirect feedback: es entsteht eine bidirektionale Verbindung zwischen zwei Knoten aus zwei benachbarten Schichten.

Die Topologien lassen sich in drei Haupttypen unterscheiden:

a) Einschichtige Netze: solche Netze besitzen nur eine Ausgabeschicht und sind deswegen die einfachsten KNN Strukturen, die es gibt (siehe Abb. 3.5).

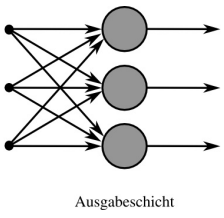


Abbildung 11.7: Ein einschichtiges KNN (Quelle: Wikipedia)

b) Mehrschichtige Netze: solche KNN verfügen neben der Ausgabeschicht auch über eine oder mehrere

versteckte Schichten, die sich zwischen den Ein- und Ausgabeschichten befinden. Sie sind deshalb von außen nicht sichtbar (siehe Abb. 3.6).

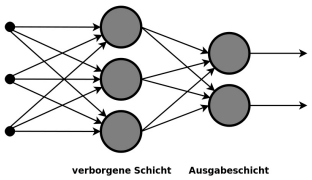


Abbildung 11.8: Ein mehrstufiges KNN mit einer verborgenen Zwischenschicht (Quelle: Wikipedia)

c) Rekurrente Netze: solche KNN verfügen über rückgekopellte Verbindungen, was zu Schleifen im Datenfluss führt (feedback loops). Dadurch wird der Ausgabewert eines Knotens als Eingabewert desselben Knotens verwendet (siehe Abb. 3.7).

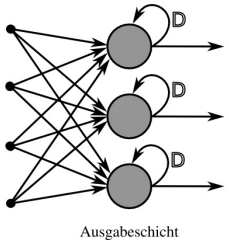


Abbildung 11.9: Ein rekurrentes KNN (Quelle: Wikipedia)

Aus diesen grundlegenden Netzwerktypen lassen sich gemischte Topologien bilden, wie z.B.: einschichtige/mehrschichtige vorwärtsbetriebene KNN (single-layer/multiple-layer feed-forward networks), Strukturen mit direkten und indirekten Rückkopplungen, Topologien mit lateralen Rückkopplungen (lateral feedback networks), usw.

11.3.3 Lernregeln

Lernregeln sind eines der wichtigsten Elemente in KNN. Ihre Aufgabe ist es dafür zu sorgen, dass das Netz aus Fehlern lernt. Beim Auftreten eines Fehlers werden die Gewichte und/oder die Topologie des KNN verändert.

Hebbsche Lernregel

Diese Regel wurde im 1949 von Donald Olding Hebb aufgestellt und gilt als universelles Lernprinzip eines KNN. Diese Regel besagt, dass, wenn zwei verbundene Neurone gleichzeitig aktiv sind, die Verbindung zwischen diesen Neuronen stärker ausgeprägt wird:

$\Delta w_{ij} = \eta a_i o_j = \eta \cdot g(a_i, t_i) \cdot h(o_j, w_{ij})$,

wobei: Δw_{ij} – die Veränderung des Gewichtes zweier Neuronen i und j, η – eine Lernrate (konstant), a_i – Aktivierung von Neuron i, o_j – Ausgabe von Neuron j.

Zusammen mit der Hebbschen Lernregel wird häufig eine binäre Aktivierungsfunktion benutzt. Dabei werden die Aktivierungen auf 1 oder -1 festgelegt. Das resultiert entweder in einer positiven Verstärkung oder in einer Verringerung der Gewichte, falls beide Neuronen nicht übereinstimmen.

Delta-Regel

Die Delta-Regel gilt als Weiterentwicklung der Regel von Hebb und berücksichtigt die Differenz zwischen der erwarteten Aktivierung t_i (teaching output) und der tatsächlichen Aktivierung a_i:

$\Delta w_{ij} = \eta \delta_i o_j = \eta (t_i - a_i) o_j$.

Diese Regel wird nur zusammen mit den linearen Aktivierungsfunktionen eingesetzt.

Backpropagation-Regel

Dieses Verfahren dient zum Einlernen von KNN und ist die Verallgemeinerung der Delta-Regel für die mehrstufigen Strukturen. KNN versuchen die Erkennungsfehler zu minimieren, indem sie jedes Mal die Abweichung von der Lösung durch das Netz rückwärts laufen lassen. Dadurch wird das KNN so angepasst, dass die Fehler besser erkannt werden.

Die standardisierte Delta-Regel führt bei zu kleinen Abweichungen zu keiner ausreichenden Informationstiefe und ist durch die Abhängigkeit von linearen Aktivierungsfunktionen in ihrer Anwendung beschränkt. Zur Lösung dieses Problems wird die Sigmoidfunktion, die selbst kleinste Abweichungen mit nichtlinearen Aktivierungsfunktionen warhnehmen kann, zur Anpassung der Delta-Regel zu Hilfe genommen.

$\Delta w_{ij} = \eta \delta_i o_j$,
wobei

$$\delta_i = \begin{cases} (1-o_i)(t_i-o_i), falls\ i\ Ausgabeneuron\ ist \\ (1-o_i)\sum_k \delta_k w_{jk}, sonst \end{cases}$$

11.4 Beispiele

11.4.1 Das Perzeptron

Im Jahr 1958 entwickelte Frank Rosenblatt das sogenannte Perzeptron: ein Modell, das praktisch fast alle logischen Funktionen berechnen kann. Man unterscheidet zwischen einlagigen und mehrlagigen Perzeptronen.

Einlagiges Perzeptron

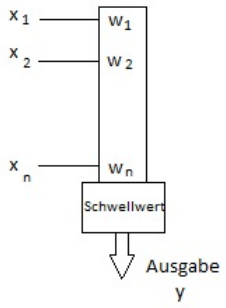


Abbildung 11.10: Ein einlagiges Perzeptron

Diese Art Perzeptronen besteht aus einer einzigen Schicht, die gleichzeitig den Ausgabevektor repräsentiert. Auf der Abbildung 11.10 ist ein einlagiges Perzeptron dargestellt. Als Eingabe bekommt es den Eingabevektor (x_1, ..., x_n).

Das Perzeptron besitzt auch einen Gewichtsvektor (w_1, ..., w_n) und einen Schwellwert θ.

Aus dem Eingabe- und Gewichtsvektor berechnet man das Skalarprodukt, und wenn das Ergebnis größer oder gleich dem Schwellwert ist, dann „feuert“ unser Perzeptron.

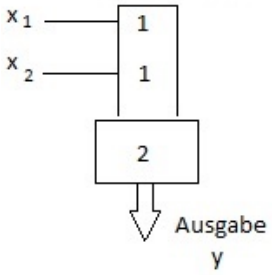
Als Beispiel betrachten wir nun ein binäres einlagiges Perzeptron, welches eine AND-Funktion modellieren kann (siehe Abb. 11.11). Bei einer binären Eingabe besteht unser Eingabe- bzw. Gewichtsvektor nur aus zwei Werten. Die erwartete Ausgabe wird als eine Tabelle in der Abbildung 11.11 dargestellt. Unser Neuron soll nur dann „feuern“, wenn das Ergebnis 1 ist, deswegen ist es sinnvoller die beiden Gewichte auf 1 zu setzen. Der Ausgabewert $y \in \{0, 1\}$, wobei y folgendermaßen definiert ist:

$$y = \begin{cases} 1, < x, w > \geq 0 \\ 0, \text{sonst} \end{cases}$$

Man sollte dabei erwähnen, dass einlagige Perzeptrone nur auf linear separierbaren Datensätzen gute Ergebnisse liefern. Linear separierbar bedeutet, dass man die Ergebnisse mit einer Linie in richtig oder falsch separieren kann. Die Abbildung 11.12 repräsentiert vier mögliche Ergebnisse aus unserem OR-Beispiel und zeigt, wie sie voneinander durch eine Hyperebene separiert werden können.

Mehrlagiges Perzeptron

Mehrlagige Perzeptrone bestehen aus mindestens zwei Schichten (einer Ausgabeschicht und einer/mehreren versteckten Schicht/-en). Dabei sind alle Neuronen einer Schicht mit den Neuronen einer anderen Schicht verbunden. So eine Topologie ermöglicht die Probleme zu lösen, bei den einlagige Perzeptrone gescheitert sind.



x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

Abbildung 11.11: Das Perzeptron-Modell einer AND-Funktion

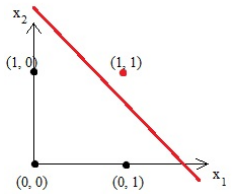


Abbildung 11.12: Linear separierbare Ergebnisse einer AND-Funktion

11.4.2 Perzeptron-Lernregel

Die Lernregel eines binären einlagigen Perzeptrons könnte man folgendermaßen definieren:
w_{ij}(neu) = w_{ij} (alt) $\pm \Delta w_{ij}$,
wobei Δw_{ij} die Änderung des Gewichts w_{ij} ist. Das bedeutet:

- Wenn das Ergebnis 1 und der Ausgabewert 1 sein soll, dann werden die Gewichte nicht geändert (das gleiche gilt für 0).
- Wenn das Ergebnis 0 und der Ausgabewert 1 sein soll, dann werden die Gewichtungen inkrementiert.
- Wenn das Ergebnis 1 und der Ausgabewert 0 sein soll, dann werden die Gewichtungen dekrementiert.

Als Beispiel betrachten wir die Funktion aus der Abbildung 11.13.

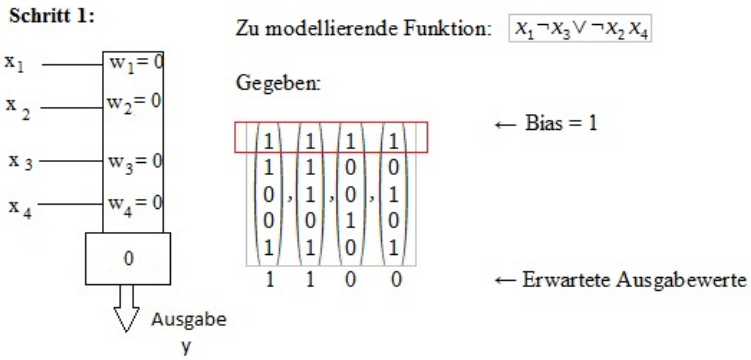


Abbildung 11.13: Eingabevektoren und Initialisierung des Gewichtsvektors

Als erstes initialisieren wir den Gewichtsvektor und den Schwellenwert mit Nullen und berechnen die erwarteten Ausgabewerte. Danach berechnen wir das Skalarprodukt von den Gewichtungen und dem 1. Eingabevektor (siehe Abb. 11.14, Schritt 2). Die erwartete Ausgabe ist 1, wir bekommen aber 0 raus, deswegen müssen wir die Werte unseres Gewichtsvektors inkrementieren. Im Schritt 3 berechnen wir das Skalarprodukt von unserem neuen Gewichtsvektor und dem 2. Eingabevektor. Der Ausgabewert stimmt mit dem erwarteten Wert überein, deswegen müssen wir nichts ändern. Die gleiche Berechnung machen wir für den 3. Eingabevektor (Schritt 4) und merken, dass der Ausgabewert 0 statt 1 sein soll. Deswegen werden die Werte des Gewichtsvektors dekrementiert. Das gleiche machen wir mit dem 4. Eingabevektor.

Wenn alle Eingabevektoren getestet wurden, fängt der neue Zyklus an. Im neuen Zyklus werden die gleichen

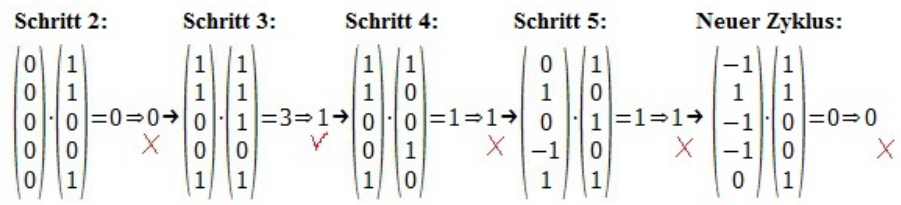


Abbildung 11.14: Schritte 2-5

Berechnungen durchgeführt, bis die passenden Gewichte gefunden sind.

11.5 Fazit

In dieser Ausarbeitung haben wir die Grundlagen und die wichtigsten Bausteine Künstlicher Neuronaler Netze kennengelernt. Zusammenfassend kann man sagen, dass KNN lernfähig sind und im Vergleich zu regelbasierten Expertensystemen keine Menschen benötigen, die explizit beschriebene Regeln erstellen müssen. Sie sind in der Lage die Gewichtswerte und ihre Struktur an jedes Problem anzupassen und besitzen daher hohe Performanz und Adaptivitätsfähigkeit. Dank ihrer Anpassungsfähigkeit sind KNN in der Lage auch für nicht gelernte Eingaben korrekte Lösungen zu finden, deswegen hat dieses Verfahren eine Allgemeingültigkeit. KNN können nicht nur lineare, sondern auch nichtlineare Datensätze bearbeiten.

11.6 Quellen und Literatur

- David Kriesel. Ein kleiner Überblick über Neuronale Netze. (Online unter: `http://www.dkriesel.com`)
- `http://www2.cs.uni-paderborn.de/cs/ag-klbue/de/courses/ss03/kimas/project/downloads/AUSARBEITUNGEN/ThomasHeinenNNInMAS.pdf`
- `https://de.wikipedia.org/wiki/KünstlichesneuronalesNetz`
- `https://de.wikipedia.org/wiki/Perzeptron`
- MIT - Lecture 12A: Neural Nets
- MIT - Lecture 12B: Deep Neural Nets

12. Genetische Algorithmen

12.1 Intro & Motivation

Was bedeutet eigentlich Intelligenz?

Das ist eine Frage, auf die keiner eine absolut klare Antwort geben kann. Was wir aber mit Sicherheit wissen ist, dass wir Menschen uns als intelligent bezeichnen. Um nun also das Konzept der künstlichen Intelligenz besser zu verstehen und voran zu treiben, hat man sich – ähnlich zu den neuronalen Netzen – einen Ansatz überlegt, der sich an der Natur orientiert: Genetische Algorithmen!

Dieser Typ Algorithmus verhält sich ähnlich zu den simplen Prinzipien der Genetik und wird häufig dafür genutzt, um Optimierungsprobleme zu lösen. Um euch dies näher zu bringen, werde ich im Folgenden das Prinzip, die Funktionsweise und Arbeitsbereiche von genetischen Algorithmen erläutern.

12.2 Begriffe

Definition 1. (Individuum)
Ein Individuum ist einzelnes Element des großen Ganzen. Es ist nicht teilbar und ist das „Ding“, was betrachtet wird. In der Biologie sind dies häufig die Zellen, aber hier werden Kandidaten für mögliche Lösungen eines Problems betrachtet.

Definition 2. (DNA)
Die DNA ist die Codierung sämtlicher Erbinformationen eines Individuums. Sie entspricht der Zusammenfassung aller Grundeigenschaften.

Definition 3. (Gen)
Ein Gen ist ein Abschnitt der DNA. Es entspricht meist einer einzelnen Eigenschaft des Individuums, kann aber auch manchmal mehrere Eigenschaften umfassen.

Definition 4. (Population)
Die Population ist die Menge aller betrachteten Individuen zu einem bestimmten Zeitpunkt.

Definition 5. (Genotyp/Phänotyp)
Der Genotyp ist die Beschreibung eines Individuums in codierter Form. Der Phänotyp dagegen beschreibt das Individuum so, wie es ist, also durch seine Eigenschaften.

Definition 6. (Fitness)
Die Fitness eines Individuums gibt dessen Güte/Stärke an. Von ihr hängen hauptsächlich die Überlebenschancen ab.

12.3 Biologischer Hintergrund

Im Folgenden werde ich kurz die Abläufe der Evolution auf biologischer Ebene erläutern. Dafür betrachte ich de Zellebene in Kombination mit der Vererbung, wie sie beim Menschen auftritt.

Jede Zelle hat eine eigene DNA (Desoxyribonucleinsäuren) im Kern, die sämtliche Informationen speichert. Diese ist aus verschiedenen Basenpaaren zusammengesetzt, die wie Puzzelteile ineinander passen. Die Gene sind Teilabschnitte der DNA und sind eine Codierung für je eine Eigenschaft. Bespiele sind helle Haut oder blaue Augen. Die Allele sind jeweils Paare von 2 Genen, die zu ein und derselben Eigenschaft gehören. Dabei stammt je ein Gen vom Vater und eines von der Mutter.

Es gibt verschiedene Arten von Erbgängen. Ich werde im Folgenden zunächst den dominant-rezessiven Erbgang erklären: Dabei ist ein Gen dominant und das andere rezessiv, was bedeutet, dass sich das dominante durchsetzt und letztendlich für die bestimmte Eigenschaft verantwortlich ist. Es sind aber beide Gene in der DNA enthalten. Bei der Reproduktion wird sowohl beim Vater, als auch der Mutter von jedem Allel eines der Gene ausgewählt und dann zu einem neuen Allel kombiniert. Dies kann dazu führen, dass beispielsweise zwei rezessive Gene zusammenkommen und sich dann im Kind eine Eigenschaft durchsetzt, die keines der beiden Elternteile besaß.

Nun folgt der intermediäre Erbgang: Hierbei haben die Gene verschiedene Werte und bei der Vererbung überlagern sich diese. Hat man zum Beispiel zwei Blumen, von denen eine weiß und die andere rot ist, so kann die Kreuzung der beiden eine rosa Blume hervorbringen.

Es können zufällige Veränderungen in der DNA einer Zelle auftreten. Dies kann zum Beispiel durch äußere Einflüsse, wie radioaktive oder ultraviolette Strahlung auftreten. Die Zellen versuchen meist diese Veränderung rückgängig zu machen, gelingt dies jedoch nicht, so ist eine Mutation in der Zelle aufgetreten. Dies kann unter anderem auch zu Krankheiten wie Krebs führen. Die Mutationen werden auch bei der Vererbung weitergegeben.

Die Selektion basiert auf dem Überleben des Stärkeren. Dies kann sich in vielen verschiedenen Dingen äußern. Einige Beispiele sind Anpassungsfähigkeit bei Veränderung der Umweltbedingungen, Stärke beim Kampf um Nahrung, Schnelligkeit und Wendigkeit, strategisches Handeln. Dabei kann es auch Vor- oder Nachteile durch gewisse Mutationen geben.

Ein weiterer Faktor ist hierbei auch die Nahrungskette. Die Lebewesen, die höher in der Nahrungskette stehen, setzen sich oft gegenüber den Schwächeren durch, da sie diese einfach fressen. Dies bedeutet aber nicht unbedingt, dass sie überleben, wenn sie nicht so anpassungsfähig sind, wie andere.

Beim Menschen tritt die natürliche Selektion jedoch kaum noch auf. Dies kommt daher, dass wir Menschen versuchen alle Leute aufzufangen, indem wir ein soziales Konstrukt aufgebaut haben. Dabei ist es das Ziel, dass jeder mit den lebensnotwendigen Dingen versorgt wird und eine Chance hat sich in die Gesellschaft einzubringen.

12.4 Ablauf

Das Ziel eines genetischen Algorithmus ist die Optimierung der Fitness aller Individuen in einer Population. Dies wird durch langfristige Weiterentwicklung in einem Zyklus erreicht. Es startet mit einer Anfangspopulation, die sich durch Vererbung immer weiter entwickelt. Man fängt damit zunächst mit einer Betrachtung des Genotyps an.

Der erste Schritt ist die Mutation einzelner Stellen in den Genen der Individuen. Darauf folgt dann eine Kreuzung der Individuen zu jeweils neuen Nachkommen. Dafür werden die Gene zweier Individuen zunächst aufgespalten und dann neu kombiniert. Dies kann auf verschiedene Arten geschehen. Man kann z. B. den Anfang des ersten mit dem Ende des zweiten Gens kombinieren. Dann wird der Phänotyp betrachtet. Dieser wird mit Hilfe einer Fitness-Funktion bewertet und anhand dieser, werden die Überlebenschancen für ein Individuum festgelegt. Zuletzt findet dann die Selektion statt. Dabei werden aus sämtlichen neu entstandenen Individuen die „besten“ ausgewählt, um die nächste Generation zu bilden. Mit dieser kann dann wieder von vorne begonnen werden.

Dies wird solange wiederholt, bis man sein Ziel erreicht hat oder es keinen Fortschritt mehr zu beobachten gibt. Dies kann aufgrund verschiedener Probleme, die später noch genauer erläutert werden, auftreten.

12.5 Codierung

Um die Individuen gut betrachten und später reproduzieren zu können muss man eine anschauliche Darstellung für die Gene wählen. Diese Wahl hängt hauptsächlich vom gegebenen Problem ab und kann deshalb nicht allgemein angegeben werden. Dennoch werde ich einige grundlegende Prinzipien zur Codierung darstellen.

12.5.1 Binärcodierung

Die Binärcodierung ist eines der ältesten Konzepte und beruht auf der Darstellung von Zahlen im Binärsystem. Dabei haben die Gene immer eine feste Länge und die Stellen darin sind entweder 0 oder 1. Die Bedeutung der Gene lässt sich im Genotyp zunächst nicht erkennen (dies ist im Allgemeinen unabhängig von der Codierung), kann aber nach der Umwandlung in den Phänotyp erkannt werden. So kann zum Beispiel eine 1 an einer Stelle dafür stehen, dass das Individuum eine bestimmte Eigenschaft besitzt, und eine 0, dass diese nicht vorhanden ist. Ein anderes Beispiel ist, dass die Gene einfach als Binärzahlen interpretiert werden und sich somit eine Zahl als Phänotyp ergibt.

Die Binärcodierung in Kombination mit der Interpretation als Zahlen kann aber auch zu Problemen führen. Mutiert man zum Beispiel bei einer 10-stelligen Binärzahl die erste Stelle erreicht man eine Änderung von 512 im Ergebnis, bei einer Mutation in der letzten Stelle jedoch nur um 1. Daher haben die Mutationen eine stark schwankende Bedeutung im Ergebnis zu Folge. Um dies auszugleichen bräuchte man einen großen algorithmischen Aufwand.

Ein weiterer Kritikpunkt ist, dass bei Kreuzungen das Elternteil, das für den vorderen Teil zuständig ist, einen deutlich höheren Einfluss auf die nächste Generation hat, als der hintere Teil. Dies sollte aber möglichst vermieden werden.

Ein Lösungsansatz für diese beiden Probleme ist die Verwendung des Gray Codes. Dieser hat im Gegensatz zum Binärcode den Vorteil, dass benachbarte Codewörter eine Hamming-Distanz von 1 haben, sich also zwei benachbarte Zahlen im Phänotyp, im Genotyp an nur einer einzigen Stelle unterscheiden.

12.5.2 Realzahl Codierung

Bei dieser Art der Codierung werden die Gene als reelle Zahlen aufgefasst. Im Gegensatz zur Binärdarstellung ist die Länge der Gene hier – zumindest theoretisch – nicht fest. In der Umsetzung ist es oft aber leichter eine feste Länge zu verwenden, was dann zu einer Darstellung as Fließkommazahlen führt.

Diese Darstellung wird oft verwendet, wenn kontinuierliche statt diskrete Probleme betrachtet werden.

12.6 Reproduktion

Im Zuge der Reproduktion werden einerseits die Gene der einzelnen Individuen mutiert und andererseits zwei Individuen zu einem neuen gekreuzt. Dabei gibt es mehrere Faktoren zu beachten, die im Folgenden erläutert werden:

12.6.1 Kreuzung

Da es oft mehrere Eigenschaften gibt, die zur Güte eines Individuums beitragen, aber nicht unbedingt ein einzelnes Individuum perfekt in allen Eigenschaften ist, ergibt es Sinn mehrere gute Individuen zu kombinieren. Da aber im Allgemeinen nicht klar erkennbar ist, welcher Teil des Gens für die Fitness positiv oder negativ verantwortlich ist, gibt es auch hier keine perfekte allgemeine Lösung als Schema zur Kreuzung. Ich werde aber dennoch mehrere Verfahren vorstellen:

Variante 1. (1-Punkt-Crossover)
Bei diesem Verfahren werden die Gene der beiden Eltern an jeweils einer Stelle geteilt. Die Gene der Nachfahren ergeben sich dann dadurch, dass der erste Teil des einen Elternteils mit dem zweiten Teil des anderen kombiniert wird und umgekehrt. Wenn zum Beispiel das erste Individuum einen sehr guten ersten Teil des Gens besitzt, aber der zweite Teil des zweiten ist überragend, so kann die Kombination der beiden ein hervorragendes Ergebnis sein und die beiden Vorgänger um vieles überragen.

Variante 2. (N-Punkt-Crossover)
Dies kann man als Verallgemeinerung des 1-Punkt-Crossovers verstehen. Dabei werden die Gene in mehr als 2 Teile aufgespalten und die Nachkommen erhalten dann alternierend die Teile der Eltern.

Variante 3. (Uniform Crossover)
Dieses Verfahren unterscheidet sich von den beiden anderen dadurch, dass die Eltern nicht in Teile aufgespalten werden. Hierfür wird zu jeder einzelnen Stelle eine Wahrscheinlichkeit berechnet, ob sie ausgetauscht wird. Ist diese größer als 0,5 werden die Stellen getauscht und sonst nicht. Dadurch ergeben sich die Nachkommen.

Variante 4. (Mittelung) Dieser Algorithmus lässt sich gut bei reellwertiger oder ganzzahliger Codierung anwenden. Um von den Eltern auf einen Nachkommen zu kommen wählt man hier – ähnlich zum intermediären Erbgang – jeweils das Mittel der beiden Stellen in den Genen um den Wert an der entsprechen Stelle im Gen des Nachkommens zu berechnen.

12.6.2 Mutation

Die Mutationen sind wichtig um eine höhere Varianz in die Population einzubringen. Hat zum Beispiel die Startpopulation nur Individuen mit demselben Genotyp und die Länge der Gene ist fest, so kann ohne Mutationen keine Veränderung im Genotyp, also auch im Phänotyp stattfinden, weshalb man zu keiner Lösung des Problems kommt.

Eine Möglichkeit zur Mutation ist die Veränderung einzelner Stellen in den Genen. Anhand einer vorher festgelegten Mutationsrate wird bestimmt, ob die betrachtete Stelle gleich bleibt, oder verändert werden soll. Tritt eine Veränderung auf, so wählt man ein anderes Codewort aus, welches nun dieser Stelle zugewiesen wird. Man kann auch die maximale/minimale Anzahl der Mutationen in einem Gen oder sogar der gesamten Population festlegen, um die Mutationen gezielt zu steuern.

Die Wahl des neuen Codeworts kann auf mehrere Arten realisiert werden. Eine Möglichkeit ist eine komplett zufällige Wahl des neuen Codeworts und eine andere eine Änderung um einen kleinen Wert.

Man sollte jedoch vorsichtig bei der Wahl der Mutationsrate sein, da sonst auch Probleme auftreten können. Wählt man die Mutationsrate zum Beispiel zu hoch, kann es sein, dass die Kreuzung der besten Individuen gar keinen Effekt mehr hat, weil sie vorher bereits zu sehr verändert wurden und sich somit deutlich verschlechtert haben.

Eine andere Variante zur Mutation der Gene ist die Mutation durch Permutation. Dabei gibt es mehrere Varianten und hier werden 3 davon kurz vorgestellt:

Variante 1. (Swap Mutation)
Hierbei werden zunächst zufällig 2 Positionen im Genotyp bestimmt und dann die Werte an den Stellen getauscht.

Variante 2. (Insert Mutation)
Man bestimmt wieder 2 zufällige Positionen. Dann wird der Wert an der 2. Position direkt hinter den an der 1. verschoben und die restlichen werden jeweils um eine Position nach hinten verschoben, bis man an der 2. Position angelangt ist. Eine Voraussetzung dafür ist, dass die erste Position vor der zweiten liegt.

Variante 3. (Scramble Mutation)
Der Algorithmus wählt hier wieder zwei Stellen im Gen aus und vertauscht dann zufällig alle Positionen zwischen diesen beiden. Die anderen Stellen im Gen bleiben dabei unverändert.

12.7 Fitness-Funktion

Die Fitness-Funktion ist einer der wesentlichsten Bestandteile eines genetischen Algorithmus. Sie bewertet die Brauchbarkeit eines Individuums bezüglich des zu lösenden Problems und ist notwendig um später die Überlebens-/Fortpflanzungschancen festzulegen. Eine der gängigsten Varianten ist, die proportionale Fitness $\frac{B_x}{B_{gesamt}}$, die die Brauchbarkeit B_x eines Individuums x in direkte Proportion zur Brauchbarkeit aller Individuen setzt. Da die genaue Form dieser Funktion stark vom zu lösenden Problem abhängt, kann man jedoch keine allgemeine Form angeben. Daher ist die Definition einer geeigneten Fitness-Funktion auch eines der größten Probleme beim Entwurf eines genetischen Algorithmus. Man kann jedoch im Allgemeinen sagen, dass es sinnvoll ist, keine Werte kleiner als 0 zuzulassen, da es sonst, wie wir später sehen werden, leicht zu Problemen kommen kann.

Man kann die Fitnessfunktion auch anpassen, um die Bedingungen zur Laufzeit des Algorithmus anzupassen:

So kann man die Fitnessfunktion f zum Beispiel zeitabhängig machen, indem man $f^* = f^{k(t)}$ als neue Fitnessfunktion wählt. Der zeitabhängige Exponent k(t) steuert dann den Selektionsdruck, bewirkt also eine Anpassung der Verteilung der Überlebenswahrscheinlichkeit.

Eine weitere Variante dafür ist die sogenannte Boltzmann-Selektion. Dabei wählt man als neue Fitnessfunktion $f^* = exp(\frac{f}{kT})$. Dabei ist T ein zeitabhängiger Temperaturparameter. Er fängt bei einem relativ hohen Wert an und fällt dann immer weiter ab. Dies hat zur Folge, dass mit steigender Zeit immer größere Unterschiede in der Fitnessfunktion auftreten, wodurch der Selektionsdruck immer weiter gesteigert wird. k ist dabei eine Normierungskonstante.

12.8 Selektion

Wir haben nun bereits die Individuen nach ihrer Fitness bewertet, aber nun stellt sich die Frage: Wie wirkt sich das eigentlich aus?

Es gibt verschiedene Ansätze von der Fitness zur Überlebenswahrscheinlichkeit zu kommen und ich möchte euch fünf davon vorstellen:

Im Folgenden bezeichne ich mit $f(x)$ die Fitness eines Individuums x und mit $P(x)$ dessen Überlebenswahrscheinlichkeit. Dabei ist die aktuelle Population die Menge $\{x_1, x_2, \ldots, x_n\}$. (Achtung! Es ist auch möglich, dass nicht nur die Kinder, sondern auch die Eltern in die nächste Generation übernommen werden. Dies nennt sich „steady state“.)

Variante 1. (Fitnessproprotionale Selektion)
Eine Möglichkeit ist ein simpler Stochastischer Ansatz:

$$P(x_j) = \frac{f(x_j)}{\sum_{i=1}^{n} f(x_i)}$$

Dabei wird die Fitness eines einzelnen Individuums ins Verhältnis zu der Summe der Fitness aller Individuen der aktuellen Population gesetzt. Dabei ergibt sich mit geeigneter Fitnessfunktion ein Wert zwischen 0 und 1, also eine gültige Wahrscheinlichkeit und somit eine Überlebenswahrscheinlichkeit.

Hierbei können jedoch mehrere Probleme auftreten:

1. Wenn die Fitnessfunktion negative Werte zulässt, dann könnte $P(x_j)$ undefiniert sein, weil $\sum_{i=1}^{n} f(x_i) = 0$ nicht ausgeschlossen ist.
2. Für $P(x_j)$ sind negative Werte nicht ausgeschlossen, was dann keine gültige Wahrscheinlichkeit mehr ergibt.
3. Bei einer sehr großen Population fallen einzelne Individuen mit relativ gesehen sehr großer Fitness gegebenenfalls gar nicht auf und deren Überlebenswahrscheinlichkeit ist trotzdem sehr gering.

Um diese Probleme zu lösen kann man nun einen etwas anderen Ansatz wählen:

Variante 2. (Ranked-Fitness-Selection)
Das Individuum mit der größten Fitness sollte die besten Überlebenschancen haben, damit ein optimaler Wert erreicht werden kann. Dafür ordnet man die Individuen zunächst in absteigender Reihenfolge nach ihrer Fitness. Dann legt man einen Startwert P_s als maximale Überlebenswahrscheinlichkeit fest. Die Überlebenswahrscheinlichkeit für x_i berechnet sich dann als:

$$P(x_i) = \begin{cases} P_s & \text{falls } i = 1 \\ (1 - P_s)P(x_{i-1}) & \text{sonst} \end{cases}$$

Damit erhält man in jedem Fall eine gültige Wahrscheinlichkeit und die besten Individuen haben eine deutlich höhere Überlebenswahrscheinlichkeit als der Rest.

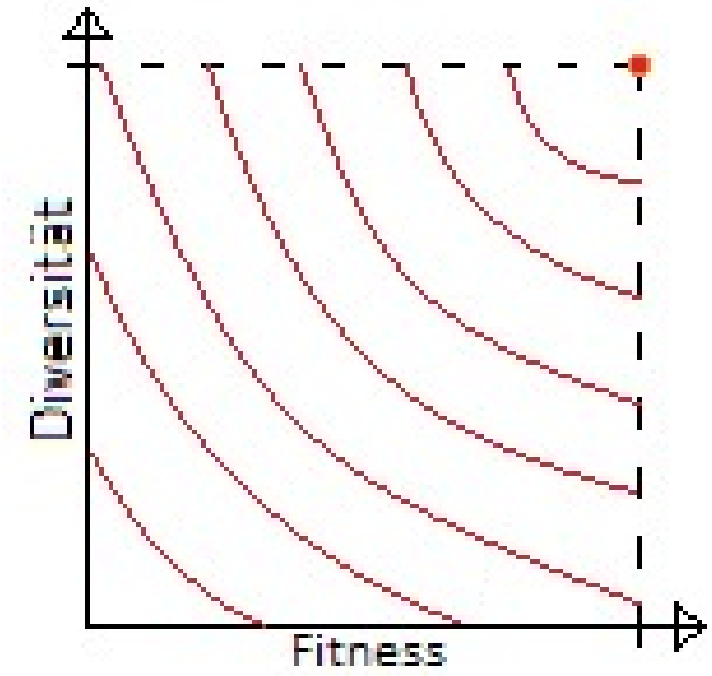
Ein Problem bleibt aber bestehen: Lokale Maxima in der Fitnessfunktion können schnell dazu führen, dass kein Fortschritt mehr auftritt und der Algorithmus somit nicht mehr funktioniert.

Um dieses Problem zu umgehen, muss eine hohe Varianz in der neuen Population gesichert werden. Dafür kann man folgenden Ansatz verwenden:

Variante 3. (Uniform-Fitness-Selection)
Man bewertet im Folgenden nicht mehr ausschließlich nach der höchsten Fitness eines Individuums, sondern zieht dabei auch die Vielfalt der nächsten Generation in Betracht. Das optimale Ergebnis wäre also eine Population mit sehr hoher Varianz und maximaler Fitness aller Individuen. Dafür bewertet man die Individuen nach folgendem Schema:

Der Startpunkt der Linien ist hierbei oben Rechts in der Ecke, also bei einem Maximum an Varianz und Fitness, also dem besten Individuum. Dabei symbolisieren die Linien einen immer weiteren Abstand zum Optimum und damit eine schlechtere Überlebenschance. Man sieht hier auch eine gewisse Abwägung zwischen Varianz und Fitness, da zum Beispiel bei der 5. Linie ein Individuum mit einer Fitness von nahezu 0 gleich bewertet wird wie eines mit maximaler Fitness, aber einer Varianz von nahezu 0.

Ein Problem, das hierbei bestehen bleibt, ist, dass durch die hohe Varianz der Algorithmus nicht wirklich konvergiert. Dies kann man lösen indem man zunächst Uniform-Fitness-Selection betreibt und danach zur Ranked-Fitness-Selection wechselt.



Eine weitere Methode, um anfangs eine hohe Varianz zu haben, aber später den Algorithmus zum Konvergieren zu bringen, ist die nächste Selektionsart:

Variante 4. (Tournament-Selection)
Das Prinzip ist einfach: Man wählt eine bestimmte Anzahl an Individuen aus und lässt sie sozusagen gegeneinander antreten, was bedeutet, das Individuum mit der höchsten Fitness wird in die nächste Generation übernommen. Ob es danach weiterhin an "Turnieren"teilnehmen kann oder nicht, bleibt dem Programmierer überlassen.

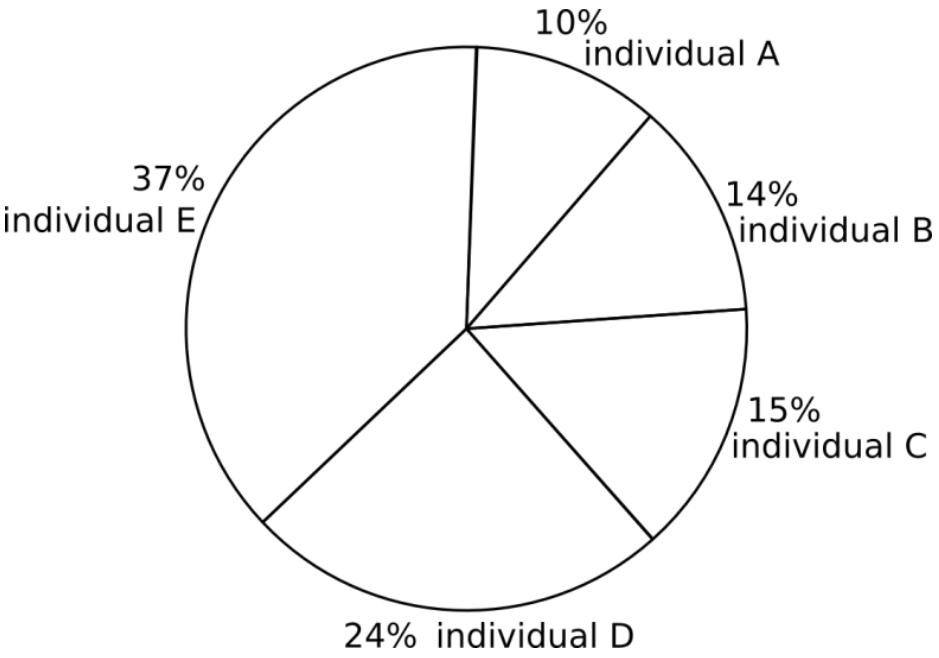
Der Vorteil, den man hierbei gewinnt, besteht darin, dass man die "Turniergröße", also die Anzahl der Individuen, die gegeneinander antreten, während der Algorithmus läuft, verändern kann. Wenn man also am Anfang eine hohe Varianz beibehalten möchte, wählt man eine kleine "Turniergröße", da so die Wahrscheinlichkeit, dass auch Individuen mit geringer Fitness in die nächste Generation übernommen werden, entsprechend hoch ist. Soll der Algorithmus später dann konvergieren, kann die "Turniergröße"größer gewählt werden, damit mehr Inidividuen mit einer wirklich hohen Fitness überleben.

Wenn man sich nun für eine dieser Methoden (oder gegebenenfalls einer anderen) entschieden hat, bleibt noch zu klären, wie man die erhaltenen Wahrscheinlichkeiten nutzt, um die neue Generation zu entscheiden (Tournament-Selection ausgenommen, da hier keine Wahrscheinlichkeiten berechnet werden, sondern die neue Generation sofort bestimmt wird). Häufig verwendet wird dazu folgende Methode:

Variante 5. (Roulette-Wheel-Selection)
Man simuliert ein Glücksrad, welches in n Abschnitte unterteilt ist, wobei jeder Abschnitt einem Individuum zugeordnet ist und die Größe des Abschnittes der Wahrscheinlichkeit entspricht, dass das zugehörige Individuum in die nächste Generation übernommen wird. Dieses Glücksrad wird dann n mal gedreht und somit die nächste Population bestimmt.

12.9 Anwendungsbereiche

Ein theoretisches Anwendungsbeispiel ist das sogenannte Traveling-Salesman-Problem. Es geht dabei darum, dass ein Kaufmann von seinem Warenlager aus mehrere Stationen in einer Stadt beliefern und am Ende wieder zum Lager zurückkehren möchte. Dabei soll jede Station genau einmal besucht werden. Nummeriert man die Stationen mit 1 bis n durch, so erhält man eine Codierung für die Gene als Permutationen von 1 bis n. Die



Fitnessfunktion ergibt sich dann als Kehrwert der Gesamtlänge des Weges. Bei der Reproduktion muss man jedoch vorsichtig sein, damit jedes mal gültige Permutationen entstehen. Dabei bietet es sich an eine Swap-Mutation durchzuführen. Die Kreuzung zweier Individuen erweist sich als komplizierter. Hier bietet es sich an ein Crossover durchzuführen, dann auf Gültigkeit zu testen und falls diese nicht vorhanden ist, das ganze rückgängig zu machen. Die entsprechenden Wahrscheinlichkeiten erhält man durch systematisches Probieren. Für die Selektion bietet sich dann die Ranked-Fitness-Selection an.

Genetische Algorithmen werden beispielsweise angewendet bei:

- Konstruktion von komplexen Bauteilen
- Erstellung von Phantombildern
- Flugzeugbau
- Logistik (Ablaufplanung)
- Routenplanung
- Anordnungsprobleme
- Packprobleme
- Strategieprobleme (Spieltheorie)
- Parameteroptimierung
- Allgemein Optimierungsprobleme

Ein Beispielprogramm für die Logistik findet sich auf www.dna-evolutions.com.

12.10 Abgrenzung / Vergleich zu Neuronalen Netzen

Ein Neuronales Netz ist ein Ansatz zur Lösung eines spezifischen Problems. Der Nachteil ist, dass man die Gewichte der einzelnen Knoten manuell durch Training anpassen muss. Genetische Algorithmen dagegen sind ein viel allgemeineres Prinzip, dass selbst optimale Lösungsansätze findet. So kann zum Beispiel auch ein genetischer Algorithmus verwendet werden, um ein neuronales Netz zu optimieren, oder sogar selbst zu entwerfen, indem er die Anzahl an Koten, Verbindungen und die Gewichte optimiert.

12.11 Fazit & Bewertung

Genetische Algorithmen sind eine gute Möglichkeit um Optimierungsprobleme und Suchen mit großem Lösungsraum zu berechnen. Dabei werden im Gegensatz zu Expertensystemen keine weiteren Datenbanken oder ähnliches benötigt, sondern der Algorithmus verbessert sich immer weiter selbst. Der Nachteil ist, dass man in jedem einzelnen Schritt stark vom Problem abhängige Wahlen treffen muss, die sich nicht unbedingt verallgemeinern lassen. Diese

wirken sich dann stark auf den Ablauf des Programms aus und bestimmen das Ergebnis. Außerdem muss das Programm gegebenenfalls zur Laufzeit angepasst werden, entscheidet man sich zum Beispiel das Selektionsverfahren zu ändern, damit eine Konvergenz des Algorithmus auftritt oder eine frühzeitige Konvergenz bei lokalen Maxima verhindert wird. Dennoch sind genetische Algorithmen eine sehr gute Alternative zum „Brute Force“ Ansatz, da zum Finden einer Lösung nur ein kleiner Bruchteil des gesamten Lösungsraums betrachtet werden muss und das Programm mit einer gewissen Intelligenz nach einer Lösung sucht, anstatt blind alles auszuprobieren. Ein Problem bleibt jedoch, welches ist, dass immer nur eine Approximation und keine exakte Lösung berechnet wird. Daher muss man vorher abwiegen, ob dies ein guter Ansatz ist.

12.12 Quellen und Literatur

(Skript Uni Paderporn)
(Skript TU Dortmund)
(Skript Uni Köln)
(Otto-von-Guericke-Universität)
(Skript Uni Magdeburg)
(Vorlesung des MIT)
(Vortrag von Kickstarter)
(Spektrum Artikel zu Zellen)
(Spektrum Artikel zu Genom)
(Semantic Web)
(Roulette-Wheel-Grafik)
(Slides von Jan Monschke, TSP)
(Tournament-Selection)
Uiform-Fitness-Grafik von Denis Krzysztala

V Sensorik & Interaktion

13. Objekterkennung und Computer Vision

13.1 Intro & Motivation

Wie können wir uns eigentlich in der Welt, in der wir sind, zurechtfinden? Die **Wahrnehmung** ermöglicht die Aufnahme von Informationen über die Umgebung. Der Mensch hat fünf Hauptwahrnehmungssinne: Sehen, Hören, Tasten, Riechen und Schmecken. Können wir einer künstlichen Intelligenz (KI) diese Wahrnehmung beibringen? Wenn ja, wie?

Sinne bei KIs nennen wir **Sensoren**. Solche, die wir in KIs einbauen können, sind Sehen, Hören und Tasten. **Sehen** ist dabei im Hinblick auf den Gebrauch von KIs in der physischen Welt für uns Menschen der hilfreichste von allen.

In dieser Ausarbeitung werden zunächst Grundlagen über die Funktionsweise des Sehens beschrieben. Der Fokus wird dann auf die Verarbeitung von gewonnenen Rohinformationen liegen und ein paar Anwendungen ansprechen.

13.2 Überblick Computer Vision

Die Frage, die Computer Vision stellt, ist eigentlich recht einfach: Wenn ein von der Umgebung ausgelöster Sensor-Impuls (S) so und so aussieht, wie sieht dann die Umgebung (W) dazu aus? Leider ist das Invertieren dieser Funktion

$$S = f(W) \Rightarrow W = f^{-1}(S)$$

nicht so einfach möglich. Man muss sich also überlegen, anhand welcher Merkmale Entscheidungen über den Inhalt eines Bildes getroffen werden können. Computer Vision lässt sich dafür grob in drei Stufen einteilen:

- *Low-Level Vision*: Grundlegende Untersuchung des Bildes auf herausstechende Features wie Unterschiede in Helligkeit, Farben oder Texturen und die damit verbundene Kantendetektion
- *Medium-Level Vision*: Objekterkennung und Bewegungsanalyse mithilfe der zuvor gesammelten Informationen
- *High-Level Vision*: Die Kombination und Analyse aller Daten: Was sagt mir das Bild, wie reagiere ich darauf?

Die Hauptanwendungsgebiete hierfür sind:

- Manipulation: Die Erlangung von benötigen Informationen und Rückmeldung über lokale Gebilde, um mit diesen zu interagieren (z. B. greifen, einfügen; Anwendung: z. B. Qualitätskontrolle).
- Navigation: Die Berechnung der eigenen Geschwindigkeit und Position (Ausrichtung), um Routen zu planen und Hindernissen auszuweichen.
- Objekterkennung: Die Charakterisierung von Objekten zur Unterscheidung von „gut“ und „schlecht“ (z. B. Beute- und Raubtier, (un)genießbare Früchte, Bekannte und Fremde).

Wir werden uns hier nur mit Low-Level- und Medium-Level Vision beschäftigen. Zunächst werden wir auf die Entstehung von Bildern, Bildverarbeitungsoperationen, nutzbare Hinweise in Bildern, optisch gesteuerte Manipulation und Navigation und schließlich verschiedene Ansätze zur Objekterkennung eingehen.

13.2.1 Grundlagen der Bildentstehung

Um Informationen über einen **Ort** zu erhalten, benötigen wir ein Bild. Um ein **Bild** (2D) zu erstellen, müssen wir **Sehen** (3D) können. Das Sehen, so wie es in unseren Augen passiert, ist sehr komplex. Vereinfacht können wir es jedoch anhand einer Lochkamera erklären - der Prozess ist ungefähr der Gleiche.

Lochkamera

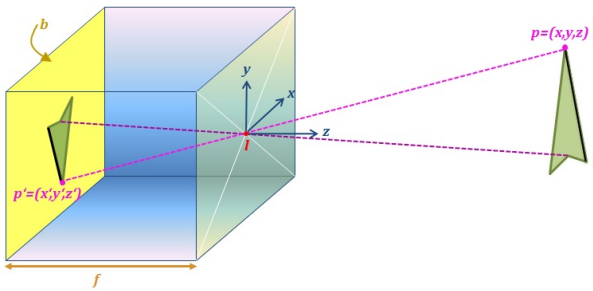


Abbildung 13.1: Geometrie der Bildentstehung in einer Lochkamera.

Eine Lochkamera ist eine schwarze Box, die an ihrer Front nur durch einen Punkt Licht eindringen lässt, welcher auf die Bildebene (Wand parallel zur Front) trifft. Je nachdem, wie groß die Lochblende ist, gelangen weniger oder mehr Lichtstrahlen hindurch. Je kleiner das Loch ist, desto schärfer ist das Bild, vor allem bei Objekten, die einen kleinen Abstand zur Lochkamera haben. Sobald das Loch vergrößert wird, gelangen mehr Lichtstrahlen hindurch, also wird ein Punkt vom Objekt durch mehrere Lichtstrahlen auf die Bildebene geschickt und das Bild erscheint uns unscharf.

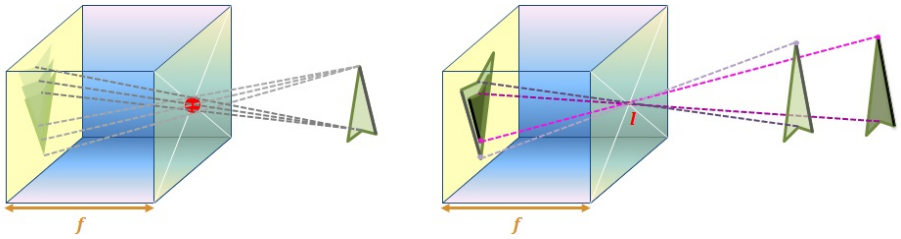


Abbildung 13.2: Größe der Lochblende beeinflusst die Schärfe (links), der Abstand vor der Lochblende beeinflusst die Projektionsgröße (rechts).

Wir betrachten einen Punkt p mit den Koordinaten (x,y,z) und seine „Kopie“ $p' = (x',y',z')$ auf der Bildebene. f beschreibt den Abstand zwischen der Lochblende l und der Bildebene b. Es entstehen zwei kongruente Dreiecke,

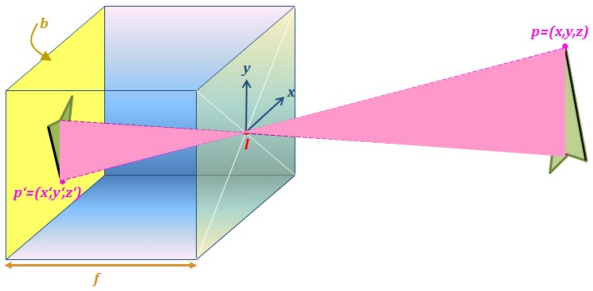


Abbildung 13.3: Invertierte, gleiche Dreiecke.

sodass wir die folgenden Umformungen vornehmen können:

$$\frac{-x'}{f} = \frac{x}{z}, \quad \frac{-y'}{f} = \frac{y}{z} \quad \Rightarrow \quad x' = \frac{-fx}{z}, \quad y' = \frac{-fy}{z}$$

Wie auf der Abbildung zu sehen ist, wird das Objekt auf der Bildebene invertiert, sodass das Objekt sowohl auf dem Kopf steht als auch gespiegelt ist (in der Gleichung ist dieser Vorgang durch das negative Vorzeichen markiert). Was hier passiert heißt **perspektivische Projektion**: parallele Linien treffen sich in einem **Fluchtpunkt** in der Ferne. Hierfür ist ausschließlich der Richtungsvektor der Linien ausschlaggebend. Für eine Linie mit der Richtung (u,v,w), die durch einen Punkt $p = (x,y,z)$ geht, gilt, dass die Projektion P_λ auf der Bildebene für jeden Punkt $p' = (x+\lambda u, y+\lambda v, z+\lambda w)$ mit $\lambda \in (-\infty, +\infty)$ auf ihr durch

$$P_\lambda = \left(f\frac{x+\lambda u}{z+\lambda w}, f\frac{v+\lambda v}{z+\lambda w} \right)$$

gegeben ist.

Linsensysteme

Linsen sind dem menschlichen Auge nachempfunden und im Vergleich zu Lochblenden viel größer, sodass mehr Licht hindurch gelangt. Dadurch können nicht alle Objekte der Welt gleichzeitig scharf auf der Bildebene abgebildet werden, sondern nur solche, die sich im Bereich der **Tiefenschärfe** – also in einem bestimmten Abstand zur Linse – befinden.

In der Linse werden alle von einem Objekt eingehenden Strahlen so gebrochen, dass sie sich in einem Punkt hinter der Linse schneiden. Die **Brennweite** f der Linse, bei der ein bestimmtes Objekt scharf dargestellt wird, ergibt sich aus den Punkten z (Abstand des Objekts zur Linse) und z' (Abstand von der Linse zur Bildebene). Die Beziehung zwischen z und z' wird mit

$$\frac{1}{z} + \frac{1}{z'} = \frac{1}{f}$$

beschrieben. Da z überwiegend sehr viel größer als z' ist, wird die Abschätzung

$$\frac{1}{z} + \frac{1}{z'} \approx \frac{1}{z'} \quad \Rightarrow \quad \frac{1}{z'} \approx \frac{1}{f} \quad \Rightarrow z' \approx f.$$

gemacht.

Licht und Schatten

Typischerweise wird die Bildebene in einer Kamera mit einem Pixel-Raster der Größe 512×512 bemessen. Die Bildhelligkeit über die Zeit wird durch $I(x,y,t)$ repräsentiert und kann modelliert werden. Sie ist von der vorhandenen Lichtmenge, der Objekt-Position und den Reflexionseigenschaften während der Bildaufnahme abhängig, wobei sowohl die Oberfläche des Objektes als auch die umliegenden Objekte Reflexionseigenschaften besitzen.

- Wir sprechen von diffus reflektiertem Licht, wenn das Licht von der Objektoberfläche absorbiert und wieder abgestrahlt wurde, wodurch die Oberfläche von allen Richtungen aus gleich hell erscheint. Eine *Lambert'sche* Ebene oder *perfekter Streukörper* ist ein Objekt, das Licht vollständig diffus reflektiert. Ein perfekter Streukörper erzeugt die reflektierte Lichtintensität $E = p \cdot E_0 \cdot \cos\theta$ mit Lichtquelle E_0, Rückstrahlvermögen $p \in [0,1]$ (schwarz bis weiße Ebene) und θ zwischen der Richtung der Lichtquelle und der Oberflächennormale.
- Das spiegelnd reflektierte Licht hingegen wird überwiegend in eine bestimmte Richtung abgestrahlt. Der Reflexionswinkel stimmt hierbei mit dem Einfallswinkel überein, wie bei einer *perfekten Spiegelreflexion*.
- Durch Unterschiede in der Lichtintensität entstehen **Schatten** auf Objekt-Oberflächen. Aus der gegebenen Helligkeit $I(x,y)$ unseres Bildes möchten wir auf die Geometrie und Reflexionseigenschaften der Umwelt schließen. Für Oberflächen von Objekten können wir eine Reflexionskarte berechnen, welche ihre Reflexionseigenschaften beschreibt.

Bei der Computergrafik wird genau diese Lichtsituation, also die Welt um das Objekt, wo Licht aus verschiedenen Richtungen kommt und (mehrfach) reflektiert wird, versucht zu simulieren. Normalerweise ist eine Mischung von direktem und indirektem Licht vorhanden. Wenn solche Interreflexion vorliegt, hilft uns die Reflexionskarte nicht weiter – ein Problem, welches noch nicht gelöst ist.

Spektralphotometrie

Neben dem Kontrastsehen gibt es noch das Farbsehen, welches sich in einem Wellenlängenbereich von Violett bis Rot, 400 nm – 700 nm, bewegt. Das menschliche Auge ist für drei verschiedene Spektralkurven empfindsam, sodass der unendlich-dimensionale Bereich der Wellenlänge auf einen drei-dimensionalen Farbbereich projiziert wird. So kommt es zu sogenannten **Metameren**: Verschiedene Lichtspektren, die von Menschen als gleiche Farben wahrgenommen werden.

13.2.2 Bildverarbeitung und Bildhinweise - „low-level“

Nun möchten wir ein aufgenommenes Bild analysieren und strukturieren.

Es gibt verschiedene Anhaltspunkte und Vorgehensweisen, mit denen wir 3D-Informationen verarbeiten können, um gewisse Aufgaben je nach Anwendungsgebiet (Manipulation, Navigation, Objekterkennung) zu lösen. In jedem Fall ziehen wir Informationen aus jedem **Pixel** (**pic**ture **el**ement = Bildpunkt). Dabei wird zwischen *low-level* Wissen, beschränkt auf lokale Information von Pixeln, und *high-level* Wissen, globale Information über Objekte, unterschieden.

Damit wir uns einen Überblick über unser Bild verschaffen können, ist es hilfreich, dieses zunächst in prägnante Bereiche zu unterteilen, also **Kanten** zu finden und **Regionen** zu identifizieren.

Kantenermittlung

Kanten entstehen durch Helligkeitsunterschiede. Wir unterscheiden verschiedene solcher **Diskontinutäten**: Tiefe, Oberfläche, Reflexion, Schatten.

Um Kanten zu finden, betrachten wir die Helligkeitsfunktion $I(x)$ und bilden ihre Ableitung $I'(x)$. Dort wo die Steigung von $I'(x)$ groß ist, lässt sich eine Diskontinuität vermuten. Jene Stellen, die nur kleinere Spitzen der Funktion sind, können durch **Rauschen** im Bild entstehen – also keine echten Kanten. Dieses Rauschen können wir durch **Glättung** der Funktion eliminieren. Eine häufig verwendete Methode ist die **Faltung** von I mit einer geeigneten Faltungsfunkton, etwa einer Gaußschen Glockenkurve G_σ oder ihrer Ableitung G'_σ. Dabei werden kleine Spitzen geglättet und große Spitzen betont.

In der entstehenden Funktion $I_G(x)$ lesen wir alle Spitzen ab, die oberhalb einer festgelegten Grenze (bspw. $0,5$) liegen, sodass Rauschen herausgefiltert wird.

Die Tatsache, dass unsere Bildfunktion nicht ein- sondern zweidimensional ist, macht den Vorgang technisch noch ein wenig komplexer, aber durch geeignete Rotation der Faltungsfunktion kann man im Prinzip Punkte auf Kanten beliebiger Ausrichtung erkennen. Kanten können wir nun als eine Reihe nebeneinanderliegender Kantenpunkte mit ähnlichen Ausrichtungen identifizieren: Dies ist die Idee des sogenannten **Canny**-Kantenfinders.

Segmentierung

Jeder Pixel ist mit optischen Eigenschaften ausgestattet: Helligkeit, Farbe, Textur. Eine Region zeichnet sich dadurch aus, dass sich in ihr diese Eigenschaften nur geringfügig ändern.

Um Regionen zu identifizieren eignet sich die Kantenfindung vorerst schon, lässt aber dann zu wünschen übrig, wenn Kantenteile nicht (durch fehlenden Kontrast) oder aber fälschlicherweise (durch Rauschen, Oberflächenmarkierung, Schatten) erkannt werden. Es gibt zwei weitere Ansätze, welche die optischen Eigenschaften von Pixeln nutzen:

- *Machine Learning* Ansatz [Malik]
  Wir können eine Abgrenzungslinie L finden, indem wir die Wahrscheinlichkeit $P_b(x,y,\theta)$ berechnen, dass L durch einen Pixel $b=(x,y)$ mit dessen Orientierung θ verläuft.
  Diese Methode funktioniert zwar besser, um Kanten im Bild zu finden, beschränkt sich aber auf den lokalen Kontext: Bestehende Regionen werden ggf. vernachlässigt, falls das Ende der Abgrenzungslinie L nicht genau im Anfang mündet. Wenn also die Pixel auf L, zu sehr in ihren Eigenschaften variieren, wird L, die eine Region einkreist, nicht als *eine* Linie erkannt und somit eben diese Region nicht als solche identifiziert.
- *Graph Partitioning* Ansatz (Shi und Malik, 2000)
  In diesem Ansatz wird jeder Pixel als Knoten in einem Graph betrachtet. Kanten bestehen zwischen nebeneinanderliegenden Pixeln. Jede Kante wird mit einem Gewicht ausgestattet, welches die Ähnlichkeit der beiden Pixel in ihren optischen Eigenschaften repräsentiert.
  Die Segmentierung besteht nun darin, den Graphen in Cluster zu zerteilen, sodass die Summe der Gewichte innerhalb einer Gruppe maximiert und zwischen den Gruppen minimiert wird.

13.2.3 Objekterkennung – „high-level“

Pose

Die Position und Orientierung, die **Pose**, von Objekten, ist besonders für die Gebiete Manipulation und Navigation wichtig. Die Position eines Punktes ist seine Koordinate $P(x,y,z)$. Wir verfügen über dessen perspektivische Projektion auf unserem Bild mit der Koordinate $P'(x',y')$, kennen aber nicht die Entfernung von P zur Kamera.

Wie können wir nun die Richtung des Objektes bestimmen? Entweder, wir betrachten das Objekt als ganzes (Koordinatensystem mit dem der Kamera verbinden) oder wir betrachten die Oberfläche des Objektes, welche abgeschrägt und geneigt sein kann.

Gestalt

Sobald sich die Kamera relativ zu einem Objekt bewegt, verändert sich Objektdistanz und -richtung, die Gestalt jedoch bleibt die gleiche. Die *lokale* Gestalt ist mit der Differentialgeometrie mittlerweile recht gut verständlich. Schwieriger ist es, *globale* Gestalten für Objekte zu definieren, die eine große Vielfalt von Objekten erfassen können.

Die geometrische Gestalt zusammen mit der Farbe und der Textur eines Objektes ist für die Objekterkennung, also die Identifizierung und Klassifizierung von Objekten, essentiell. Hierfür können wir verschiedene Hinweise nutzen, die uns mit Informationen über die physische Welt versorgen:

- Bewegungsfluss
  Das Prinzip ist, zwei Aufnahmen – zeitlich kurz aufeinanderfolgend – zu machen, und zu vergleichen, wie sich das Objekt verändert hat, also den Bewegungsfluss zu messen. Dazu müssen wir in beiden Aufnahmen die entsprechend zueinander passenden Punkte finden. Wir nehmen einen Pixel $p = (x_0, y_0)$ aus der ersten Aufnahme zum Zeitpunkt t_0. Da die Struktur in einem Block um p in beiden Aufnahmen erhalten bleibt, können wir geeignete Blöcke mit Pixel-Kandidaten $q_i = (x_0 + x_i, y_0 + y_i)$ in der zweiten Aufnahme zum Zeitpunkt $t_0 + t_i$ wählen, miteinander vergleichen und so unser mit p übereinstimmendes q finden.
  Dieses Verfahren können wir nutzen, um den Abstand von der Kamera zu Objekten einzuschätzen, denn: Entferntere Objekte verändern sich weniger stark als nahe Objekte.
- Zweifach-Tiefenwahrnehmung
  Ähnlich wie beim Bewegungsfluss werden hier auch zwei Aufnahmen genutzt, allerdings nicht unterschieden in der Zeit, sondern in der Kameraposition, sodass zwei leicht verschiedene 2D-Bilder der gleichen 3D-Umgebung entstehen.
  Wie können wir nun feststellen, dass diese beiden Bilder das gleiche Objekt darstellen? Dazu betrachten wir die Geometrie dahinter: In Abb. 13.4 ist die Idee der Stereopsis skizziert. c_1 stellt die linke Kamera, c_2 die

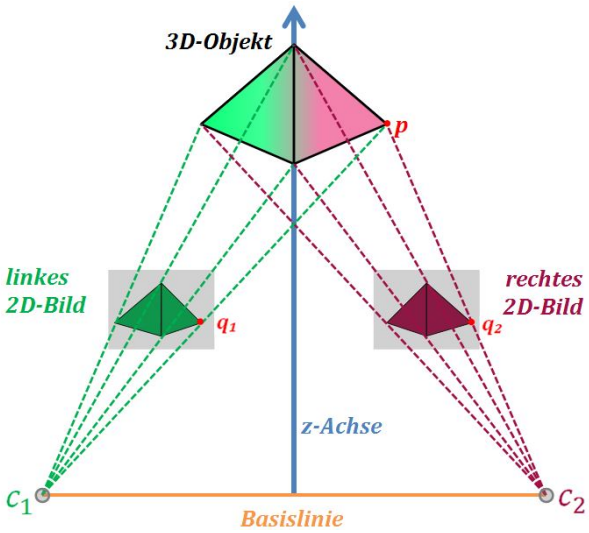


Abbildung 13.4: Stereopsis mit epipolaren Linien.

rechte dar. Die gestrichelten Linien sind die **epipolaren Linien**. Damit können wir feststellen, welche zwei Pixel q_1 und q_2 aus den beiden variierenden 2D-Bildern den gleichen Punkt p des Original-Objektes aus der 3D-Welt darstellen. Pixel, die den gleichen Punkt des Objektes darstellen, müssen immer auf epipolaren

Linien der 2D-Bilder liegen. Hierbei besteht

- **Eindeutigkeit** (es kann immer nur maximal – denn manchmal wird der Objekt-Punkt wegen Überdeckungen nur von einem Sichtfeld aufgenommen – einen Pixel q_1 des einen Bildes zugehörig zu einem Pixel q_2 des anderen geben) und
- **stückweise Stetigkeit** (abgesehen von Objektkanten und Überdeckungen, haben benachbarte Punkte immer ähnliche Werte).

Notiz: Algorithm von Belh

- Textur
  Die Textur von Oberflächen in der Welt ist ein sich wiederholendes invariables Muster. Im Bild wird dies durch *texel* (**tex**ture **el**ements) repräsentiert, welche in ihrer Größe und Gestalt aufgrund der texel-Richtung (Verkürzung falls nicht orthogonal) relativ zur Kamera sowie auch durch unterschiedlichen Abstand zur Kamera sehr stark variieren können.
  Mit Hilfe von **texture-Gradienten**, Funktionen der Oberflächengestalt unter Berücksichtigung der Textur-Abschrägung und -Neigung, kann die Veränderungsrate von Bereich, Verkürzung und Dichte der texel aufgeklärt werden.

Notiz: Algorithm von Mal Rosenho

- Kontur
  Konturen sind Linien, welche verschiedene Bedeutungen haben können, die wir ihnen zuordnen möchten. Hierfür betrachten wir einfache Strichzeichnungen und unterscheiden zunächst zwei grobe Klassen, welche wir anschließend verfeinern:

So können wir Konturen bspw. klassifizieren:

Abbildung 13.5: Konturklassifizierung.

Huffman und Clowes haben unabhängig voneinander ein Schema entwickelt, um systematisch vielgestaltige Bilder zu analysieren. Es werden hierbei Objekte mit bis zu drei aneinandergrenzenden Flächen (Spalten werden ausgeschlossen) pro Eckpunkt betrachtet. So ergeben sich vier Fälle, wie ein solcher dreiflächiger Eckpunkt entstehen kann:
Aus den verschiedenen Sichtweisen der Eckpunkte ergeben sich folgende Konturklassifizierungen:

Notiz: Waltz ha Algorithm entworfe der S Schatten getrennte Aushöhlu berücksi Mackwo

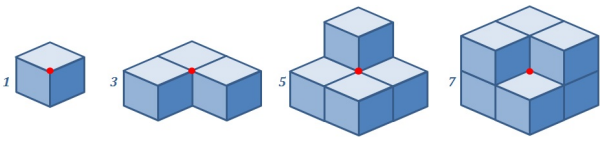


Abbildung 13.6: Anordnungen von Würfeln für die Entstehung eines neuen dreiflächigen Eckpunkts (roter Punkt). Die Zahl gibt die Würfelanzahl an.

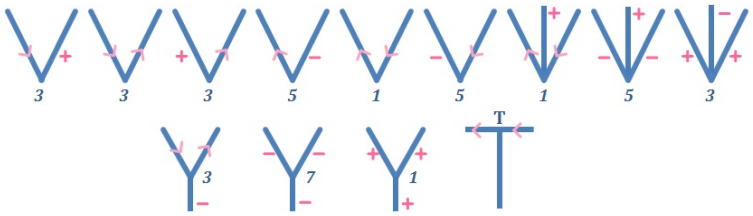


Abbildung 13.7: Das Huffman-Clowes Klassifizierungs-Set.

13.3 Objekterkennung: Probleme & Anwendung

13.3.1 Probleme

Menschen ist es möglich, unzählige Objekte zu erkennen, selbst wenn diese teilweise verdeckt oder nur karikiert sind. Für einen Computer ist das nicht so einfach. Er muss über ein großes Reportoire von Objektarten sowie ihre Variationen verfügen und diese dann auch noch im Bild wiederfinden. Dieses Problem der **Objektvielfalt** können wir für abgewandelte Grundformen wie z. B. Würfel und Zylinder ganz gut lösen: Ein **verallgemeinerter Zylinder** hat

- einen ebenen Querschnitt,
- eine Achse (gerade oder gewölbt) und
- eine Regel, die die Änderung des Querschnitts entlang der Achse beschreibt.

Viele Objekte können in Zylinder zerlegt und somit erkannt werden. Dies kann aber auch dazu führen, dass das Objekt in so viele Zylinder-Teile gebrochen wird, dass sich dies auf die Laufzeit auswirkt.

13.3.2 Anwendung

Betrachten wir eines der am Anfang genannten Hauptanwendungsgebiete der Computer Vision, die Navigation – speziell: automatisiertes Fahren. Die Aufgaben eines Fahrers sind

1. Beschleunigung des Fahrzeugs,
2. Kontrolle: seitlich (Fahrspur) und längs (Sicherheitsabstand),
3. Wachsamkeit: Verkehrsüberblick und ständige Handlungsbereitschaft.

Nehmen wir Punkt 2 heraus: Welchen Ansatz aus den bisherigen Kapiteln könnten wir hier verwenden?

Bei der Kontrolle eines Fahrzeugs gibt es zwei wichtige Bedingungen: Zum einen möchten wir eine Linie bzw. Kante – die Fahrspur – finden und zum anderen möchten wir vor uns liegende Objekte – Autos/Hindernisse – erkennen. Folgende Tabelle zeigt auf, was wir aus dem bisher Gelernten nun anwenden können:

Aufgabe	benötigte Infos	Ansatz CV
Kontrolle seitlich	Spur, Fahrzeugsposition	Kantenerkennung
Kontrolle längst	Distanz zu Objekten	Zweifach-Tiefenwahrnehmung, Bewegungsfluss

Was aber ist mit speziellen Objekten, wie zum Beispiel Verkehrsschilder oder Menschen? Bisher haben wir sozusagen nur die Basis der Objekterkennung kennengelernt – im Folgenden lernen wir noch zwei Methoden zur detailliertere Objekterkennung kennen.

Abgleichungsmethode

Wir möchten die Projektion eines 3D-Objektes der Welt auf unserem Bild identifizieren, ohne dass wir die Gestalt, Position und Richtung des Objektes kennen. Wir wissen nur, dass es eine **skalierte orthografische Projektion**

ist, sprich das Objekt wird auf dem Bild nach hinten hin nicht kleiner. Huttenlocher und Ullmann haben folgende Vorgehensweise (hier vereinfacht) verifiziert:

1. Wir verfügen über m **charakterisierende Originalpunkte** $\mu_1, \mu_2, ..., \mu_m$ des 3D-Objektes in seiner Grundhaltung aus der Welt.
2. Durch Rotation und Verschiebung des 3D-Objekts sowie anschließender Projektion, sodass ein Bild entsteht, verändern sich die Originalpunkte zu n **charakterisierenden Bildpunkten** $p_1, ..., p_n$.
   Hierbei ist normalerweise $m \neq n$, da Originalpunkte verdeckt werden und nicht/fälschlicherweise (wegen Rauschen) identifiziert werden. (Abb. 3.1)

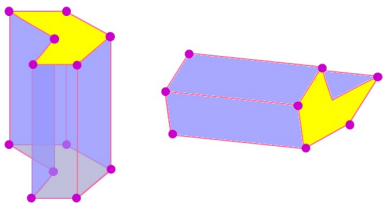


Abbildung 13.8: Links: Original 3D-Objekt mit 12 charakterisierenden Punkten. Rechts: 2D-Projektion, nur 8 der Charakterpunkte sind erhalten geblieben.

3. Q, die unbekannte Rotation und Verschiebung des Objektes, **transformiert** jedes μ_i und kreiert exakt zugehörige Bildpunkte $p_i = Q(\mu_i)$.
4. Wenn wir genug Punkte haben, können wir Q auflösen. Es gilt: Mit drei nonkollinear (nicht auf einer Linie liegenden) Originalpunkten und drei Bildpunkten ergeben sich genau zwei Transformationen des 3D-Objekts.

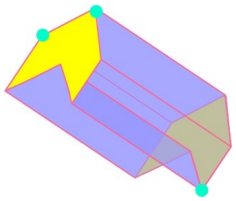


Abbildung 13.9: Bei drei Fixpunkten (türkis) kann das Objekt nicht rotieren.

Man sucht sozusagen eine Funktion, mit der ein Originalpunkt zu einem Bildpunkt transformiert wird: $Q(\mu_i) = p_i$.

Der Algorithmus ALIGN verwendet eine „Ausprobier"-Lösungsstrategie: ALIGN vermutet zu Originalpunkten zugehörige Bildpunkte und gibt entweder `false` zurück, falls die Annahme falsch war, oder Q, falls sie richtig war.

```
function align(Originalpunkte,Bildpunkte) returns Q or false
    loop do
        choose set_p
        if (set_p==tried) && (no set_p left) then
            return false
        while !(no set_p left) do
            choose set_μ!=tried
            Q <- findTransform(set_μ,set_p)
            if projection according to Q explains image then
                return Q
        end
    end

function findTransform(set_μ,set_p) returns Q such that
    Q(μ_1)=p_1
    Q(μ_2)=p_2
    Q(μ_3)=p_3
```

Abbildung 13.10: Pseudocode ALIGN.

Wir können uns das in etwa so vorstellen:

- Beispiel:
  Wir haben je fünf charakterisierende Punkte unseres 3D- und 2D-Objektes.

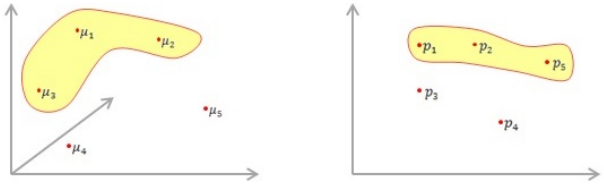


Abbildung 13.11: ALIGN rät ein Set S_μ von drei Originalpunkten und ein Set S_p von drei Bildpunkten.

Jetzt möchten wir eine Transformation Q berechnen. ALIGN „rät" hierfür Paare (μ_i, p_j) mit $\mu_i \in S_\mu$, $p_j \in S_p$ und $i, j \in \{1, ..., 5\}$. In unserem Beispiel legt ALIGN die Paare $(\mu_1, p_1), (\mu_2, p_2), (\mu_3, p_3)$ fest.
Wir können uns die Rechnung wie folgt vorstellen, die für jedes Paar angewandt wird:

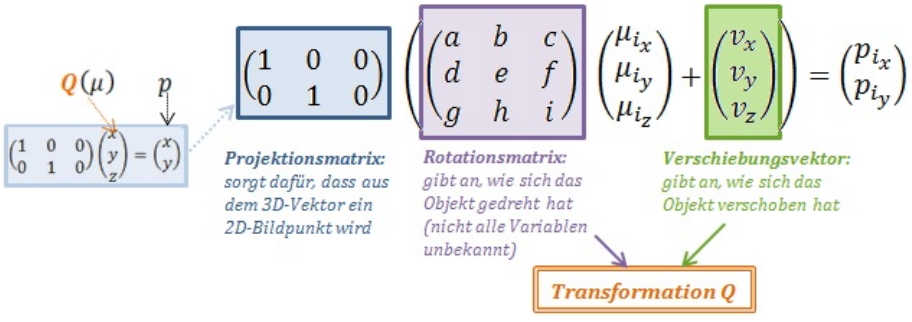


Abbildung 13.12: Berechnung der Transformation Q.

Die gefundene Transformation Q wird nun auf die restlichen, *nicht* im Set vorhandenen Paare – in unserem Beispiel bleiben die Kombinationen $(\mu_4, p_4), (\mu_5, p_5), (\mu_4, p_5), (\mu_5, p_4)$ – angewandt. Wenn für ein Paar, bei dem ALIGN Q auf μ anwendet, genau das Ergebnis p rauskommt, dann haben wir die richtige Transformation Q gefunden. Falls nicht, werden zwei neue, noch ungetestete Sets ausgewählt und es geht von vorne los.

- Laufzeit
  Im schlimmsten Fall liegt die Laufzeit in $O(m^4 n^3 log\ n)$, denn Q muss so oft berechnet werden, wie es Sets gibt, das ist $\binom{m}{3}\binom{n}{3}$ und die Überprüfung von Q kostet nochmal $m\ log\ n$.
  Olson hat die Laufzeit auf $O(mn^3)$ wie folgt verbessert:
  - „pose-clustering": Er **gruppiert** nur sich ähnelnde – somit nah an der korrekten Transformation liegende – Q-Transformationen, was in $O(mn)$ liegt.
  - Wenn wir im Vorhinein zwei Paare (μ_1, p_1) und (μ_2, p_2) gegeben hätten, könnten wir das Set komplettieren, indem wir das dritte Paar aus den Schnittpunkten der verlängerten Linien ablesen (Abb. 13.13). Olsen nutzt dieses Prinzip mit der **randomisierten** Auswahl zweier Paare, das kostet $O(n^2)$.

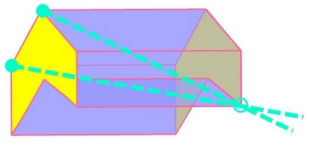


Abbildung 13.13: Geometrische Berechnung des dritten Punkts aus zwei gegebenen Punkten.

Projektive Invarianz

Hier haben wir eine Bibliothek mit verschiedenen Objekten. Jedes Objekt hat einen **invarianten Wert** („index function") I, welcher mit Hilfe der geometrischen Invarianz als Repräsentation für alle Posen, die das Objekt annehmen kann, erstellt wird (Abb. 13.14).
Ein Objekt wird als Modell mit seinen charakterisierenden Eigenschaften abgespeichert (diese sind u. a. Name, Kanten, Kegelquerschnitte , Wölbungen, I). Die Aufnahme von Modellen ist sehr einfach, sie kann

Notiz:
Wir er
uns an
Objekte k
in Zy

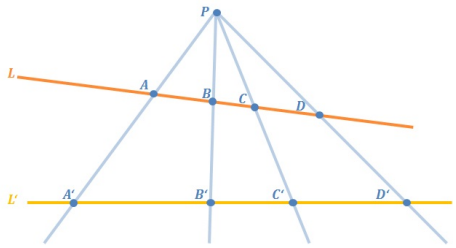


Abbildung 13.14: Ein einfaches Beispiel für die projektive Invarianz: Doppelverhältnis von vier Punkten auf einer Geraden. Solange L nicht durch P geht, gilt: $(A,B;C,D) = \frac{AC \cdot BD}{BC \cdot AD} = \frac{A'C' \cdot B'D'}{B'C' \cdot A'D'} = (A',B';C',D')$

zum Beispiel direkt von einem Bild – ohne spezielle Kenntnisse über die Kameraposition etc. – übernommen werden.
Zusätzlich generiert jedes I seine eigene Hypothese. Die Is sind idealerweise alle einzigartig – falls sehr viele Objekte in der Bibliothek sind, ist das manchmal nicht möglich, dann werden sie zu einer gemeinsamen Hypothese für diese Objekte kombiniert.
Die Objekterkennung geschieht dann wie folgt:

1. Kanten finden und bündeln
2. I messen
3. Nach Übereinstimmung in Bibliothek suchen:

entweder	I gefunden und einzigartig	$\Rightarrow$ Objekt erkannt
oder	I ist nicht einzigartig $\rightsquigarrow$ Hypothese des Modells mit größten Übereinstimmung (i. A. ab 50%) wird bestätigt	$\Rightarrow$ Objekt erkannt
oder	I nicht gefunden	$\Rightarrow$ Objekt *nicht* erkannt

Anwendung findet diese Methode beispielsweise bei Satellitenaufnahmen.

13.4 Fazit & Bewertung

Computervision ist die Königsdisziplin der Informationsverarbeitung. Was bei uns Menschen automatisch passiert, ist für eine KI nur mit raffinierten Berechnungen annähernd möglich.

Bspw. ist die effektive Repräsentation von Objektarten – vor allem für gewölbte Objekte – größtenteils noch ungelöst. Es ist nicht nur aus wissenschaftlicher Sicht interessant, sondern auch aus industrieller: Würden wir mehr darüber wissen, könnten wir z. B. den Robotern besser beibringen, wie sie Objekte erkennen, greifen, bearbeiten, und so weiter.

In den letzten Jahrzehnten hat sich zwar viel getan (z. B. ist jede Digitalkamera heutzutage standardmäßig mit einer Gesichtserkennung ausgestattet, sind automatische Fahr- und Bremssysteme von Automobilherstellern bereits auf dem Markt, ...), dennoch ist der Mensch unverzichtbar: Er wird dort gebraucht, wo die Maschine (noch) an ihre Grenzen stößt.

13.5 Weitere Anwendungsgebiete

Die Hauptanwendungsgebiete neben der **Objekterkennung** sind **Manipulation** der dreiminesionalen Welt und **Navigation** (Pfaderkennung, Ausweichmanöver, etc.). Diese und andere Einsatzgebiete neben den zur Veranschaulichung schon bereits besprochenen Nutzungsmöglichkeiten der Objekterkennung werden in den folgenden Abschnitten dargelegt.

13.5.1 Bilder und Schlagworte

Heutzutage bietet das Internet die Möglichkeit, anhand von Schlagwörtern nicht nur nach Text zu suchen, sondern auch Bilder zu finden. Das automatisierte Versehen von Bilder mit Schlagworten ist sehr wichtig: Wir haben eine Menge von Beispielbildern und möchten mit diesen einige Testbilder abgleichen (**Problem der Auto-Annotation**). Das beste Ergebnis erhält man mit der **Nächste-Nachbarn-Klassifikation**, die die Trainingsbilder raussucht, die

dem Testbild durch bestimmte Merkmalsausprägungen am ähnlichsten sind. Die Schwierigkeit liegt in der Erkennung von Aktionsmustern. Man kann zwar schon durch Hintergrundsubtraktion und anschließender Betrachtung von HOG-Merkmalen basierend auf dem optischen Fluss die Verhaltensstrukturen annähernd gut erkennen und so beispielsweise einen trinkenden Menschen im Bild ausfindig machen (Hand wird mit Objekt zum Mund geführt), allerdings ist diese Erkennung sehr eingeschränkt: Man könnte mit diesen Merkmalsausprägungen keine trinkende Katze in einem Bild erkennen.

13.5.2 Rekonstruktion

Besteht die Vermutung, was für ein Objekt sich auf einem Testbild befindet, so reicht manchmal allein schon dieses Bild aus, um mehrere Ansichten des Objekts zu erstellen. Wenn das Objektmodell, d.h. die Repräsentation des Objekts in dem Klassifizierer, aus mehreren Punkten oder sogar Bildern besteht, können wir Korrespondenzen zu Punkten aus dem Testbild dokumentieren. Anhand der Punkte können wir dann die Parameter der Kamera, falls diese nicht vorliegen, bestimmen und überprüfen, ob die Modellpunkte mit den Bildpunkten übereinstimmen.

Diese Technologie der Rekonstruktion ist heutzutage so hochentwickelt, dass sie in folgenden Bereichen zur Anwendung kommt:

- **Modellerstellung** (z.B. dreidimensionale Karten)
- **Bewegungsabgleich** (z.B. bei animierte Figuren in realen Filmsequenzen)
- **Pfadrekonstruktion** (z.B. bei mobilen Robotern)

13.5.3 Bewegungssteuerung

Wie wichtig die Manipulation von Objekten und die Navigation beispielsweise beim Ausweichen von Hindernissen ist, wird nicht nur in der Tierwelt deutlich. Wollen wir ein Visionssystem für ein **selbstfahrendes Fahrzeug** implementieren, so hat dies für folgende Aufgaben geeignete Aktionen zu finden:

- **Seitensteuerung**: innerhalb der Spur bleiben oder Spurwechsel ausführen.
- **Längssteuerung**: bremsen, beschleunigen, Abstand halten.
- **Ausweichen**: Objekte in der Nähe beobachten und Kollisionen vermeiden.

Für die Seitensteuerung werden Algorithmen zur Kantenerkennung: Sie dienen dazu, Mittelstreifen zu erkennen und die Position und Ausrichtung des Autos relativ zur Spur zu verwalten.
Für die Längssteuerung und das Wahrnehmen von Hindernissen müssen wir Abstandserkennung mit Strukturerkennung kombinieren: Durch binokulares Stereosehen oder durch optischen Fluss, unterstützt durch HOG-Merkmale, können sich Fahrzeuge an Verkehrsregeln halten.

Mobile Roboter stellen eine größere Herausforderung da. Eine Methode zur Lokalisierung dieser in ihrer Umgebung besteht aus stereoskopischen Kamerasystemen. Um bei schlechten Lichtverhältnissen mit genügend Informationen versorgt zu werden, ist von den zwei Systemen eins nach vorne und das andere nach hinten ausgerichtet. Sollte die Beleuchtung allerdings doch mal erheblich schlecht sein, sorgt ein Trägheitsmodul, das ähnlich wie unser Gleichgewichtsorgan funktioniert, für die weitere Informationsbereitstellung.

Ein weiteres Problem der (visuellen) **Odometrie** (Bestimmung der Positionsänderung) ist die „Drift“, bei der sich kleine Positionsabweichungen mit der Zeit aufaddieren. Zur Lösung dieses Problems werden feste Standpunkte in der Umgebung installiert, an denen der Roboter sich orientieren kann.

13.6 Quellen und Literatur

Die Ausarbeitung stützt sich hauptsächlich auf das Buch

Artificial Intelligence - A Modern Approach
von Stuart J. Russell and Peter Norvig (1995, Kap. 24.2.4 aus 2016)

Weiteres habe ich hier nachgelesen:

http://aima.cs.berkeley.edu/
http://faculty.washington.edu/cfolson/papers/pdf/ijcv97.pdf
Object Representation in Computer Vision II
von Jean Ponce, Andrew Zisserman, Martial Hebert
http://e-collection.library.ethz.ch/eserv/eth:25629/eth-25629-03.pdf
https://www-m10.ma.tum.de/foswiki/pub/Lehre/WS0809/
GeometrieKalkueleWS0809/GeoKalkSkript.pdf

www.wikipedia.de
https://www.bmvi.de/SharedDocs/DE/Anlage/Digitales/bericht-zum-forschungsbedarf-runder-tisch-automatisiertes-fahren.pdf?__blob=publicationFile (2015)
http://tex.stackexchange.com/questions/232550/visual-table-of-contents-using-tikz-mindmap-or-similar
http://www.texample.net/tikz/examples/servers/

14. Sprachverarbeitung

14.1 Einleitung

Sprache ist ein natürliches Kommunikationsmittel, das der Übermittlung von Informationen dient. Es liegt daher auf der Hand, Sprache auch zur Kommunikation mit Computern nutzen zu wollen. Zeichenketten sind als Eingaben gang und gäbe. Interessant für den Bereich der künstlichen Intelligenz werden solche Eingaben, wenn sie eine Interaktion mit einem Rechner ermöglichen, die nicht nur streng formal definierte und eng begrenzte Eingaben, sondern eine möglichst große Teilmenge einer natürlichen Sprache erlaubt. Dies setzt voraus, dass der Rechner in der Lage ist, die Eingabe zu analysieren und zu interpretieren. Insbesondere ist auch die gesprochene Kommunikation mit Computern ein aktueller Trend. Diese Eingabeschnittstelle gewinnt insbesondere durch die zunehmende Nutzung mobiler, tastaturloser Geräte in Form von Sprachassistenten wie Siri, Cortana und Google Now an immenser Bedeutung. Aber auch die Sprachsteuerung von Unterhaltungselektronik und Haushaltsgeräten (smart homes) sowie von Automobilen, die verstärkt computerisiert ausgestattet sind und sich während der Fahrt am besten per Sprache bedienen ließen[1], sind hier zu nennen. Hier befassen wir uns mit zwei Formen von Sprache: gesprochene Sprache und Texte.[2]

Texte sind Folgen von Buchstaben und Zeichen aus einem gegebenen Alphabet. Die Menge der in einer Sprache gültigen Zeichenfolgen wird syntaktisch durch eine Grammatik definiert. Hinzu kommt aber noch eine semantische Ebene, die einen logischen Zusammenhang dieser Folgen herstellt und somit die eigentliche Information repräsentiert. Es ist also leicht möglich grammatikalisch korrekte Zeichenfolgen zu konstruieren, die keine verwertbare Information liefern (z. B. „Der Fisch fliegt grün.“). Wie wir sehen werden, ist die Konstruktion einer Grammatik, die eine natürliche Sprache definiert, bereits komplex genug. Deutlich schwieriger gestaltet sich jedoch die logische Erfassung des Kommunikationsprozesses und somit eine tatsächliche Kommunikation mit einer künstlichen Intelligenz. Als Folge von Zeichen lassen sich Texte gut bearbeiten, da Zeichen leicht binär kodiert werden können (z.B. ASCII). Für gesprochene Sprache gilt dies nicht. Hier wird Sprache zunächst als Wellenformen digitalisiert, die ein breites Spektrum von Mustern einnehmen können und oftmals auch noch durch Störquellen verfälscht sind. Das gesprochene Wort muss also zunächst in einen Text übersetzt werden, der dann wiederum auf syntaktischer und semantischer Ebene interpretiert werden kann. Für diese Übersetzung sind akustische und phonetische Merkmale und Modelle zu berücksichtigen, die wir uns zunächst anschauen.

[1] Zumindest solange wir noch nicht von selbstfahrenden Autos reden.

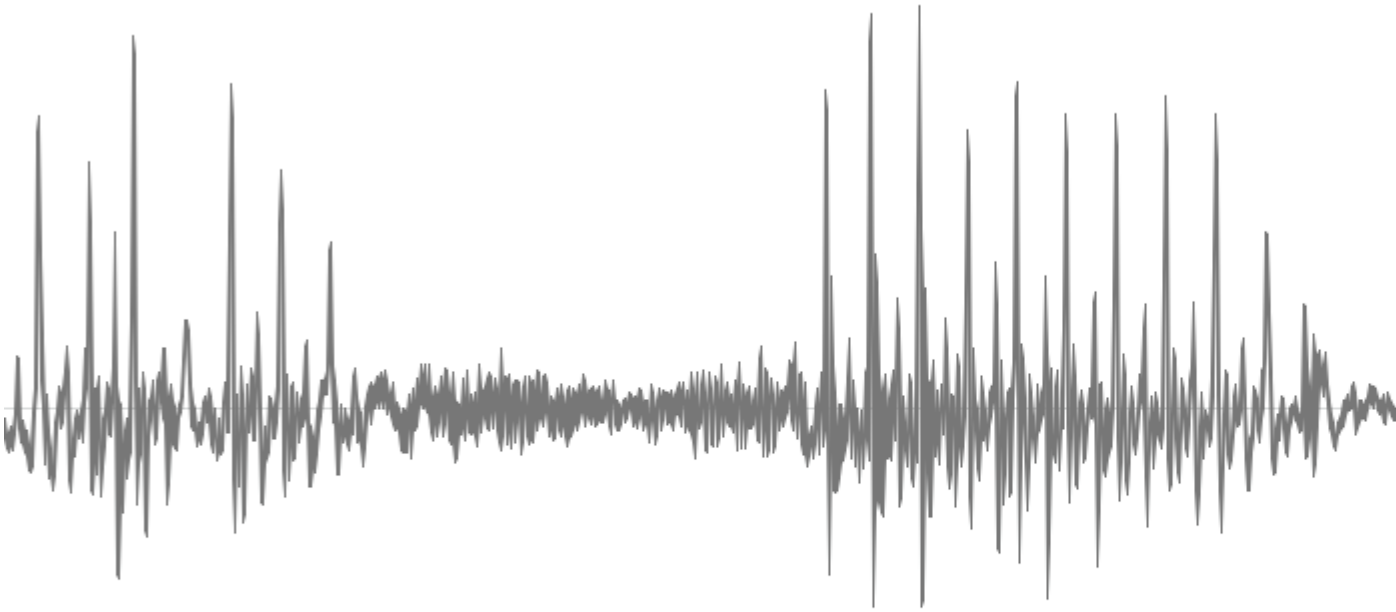
[2] Darüber hinaus gibt es natürlich noch weitere Möglichkeiten sprachlicher Kommunikation, etwa Körpersprache und Gebärdensprache, für die aber wiederum andere Methoden der digitalen Erfassung notwendig wären.

14.2 Spracherkennung

14.2.1 Problemstellung

Das menschliche Sprechorgan (Stimmbänder, Kehlkopf, Mundhöhle) ist wie bei kaum einem anderen Lebewesen in der Lage, differenzierte Geräusche zu artikulieren und damit Sprachklänge (Phoneme) zu formen. Wie alle anderen akustischen Signale ist Sprache somit eine komplexe Überlagerung zahlreicher Sinuskurven unterschiedlicher Frequenz, die über den Zeitverlauf stetig variieren. Diese Schwingungen werden durch verschiedene Medien, vor allem die Luft, aber auch durch Wasser und feste Körper, als Druckunterschiede übertragen, wodurch letztlich das Trommelfell des menschlichen Ohres in Schwingungen versetzt wird und über einen mehrstufigen Transformationsprozess im Innenohr Nervenzellen angeregt werden, die die Information an das Gehirn weiterreichen, wo sie verarbeitet werden kann. Der Prozess der Digitalisierung des Schalls funktioniert hier sehr ähnlich. Zunächst werden die Druckunterschiede über eine in Schwingung versetzte Mikrofonmembran und einen daran angeschlossenen Magneten in elektrische Signale umgewandelt. Diese können wiederum durch eine gleichmäßige Abtastung diskretisiert werden. Es werden also in regelmäßigen Abständen Pegelinformationen über die Wellenform des Signals gespeichert. Sowohl für die Zeitachse als auch für die Amplitude sind verschiedene Granularitäten möglich. Verbreitet ist z.B. das CD-Format mit 44.100 Samples pro Sekunde (Maximalfrequenz 22,05 kHz) und 16 Bit pro Sample (Rauschabstand 96 dBfs). Für die menschliche Stimme, die bis etwa 5 kHz relevante Informationen enthält und in der Regel auch einen begrenzten Dynamikumfang hat, wäre eine geringere Auflösung nicht nur akzeptabel sondern sogar wünschenswert. Durch die Reduktion auf den für die menschliche Sprache relevanten Tonumfang können bereits viele Störgeräusche eliminiert werden.

Beispiel einer Wellenform von Sprache. Man erkennt gut die Unterschiede zwischen langen, stimmhaften Vokalen und kürzeren, hochfrequenten Konsonanten.



Ein Problem bei der Transformation von Sprache in Text ist zunächst, dass wir keine eindeutige Kodierung von der digitalen Information in das gemeinte Zeichen haben. Dies beginnt bereits damit, dass unterschiedliche Wellenformen von Aufnahmen desselben Textes nur Ähnlichkeiten haben, aber niemals identisch sind. Sie variieren in Lautstärke, Geschwindigkeit, Tonhöhe, Artikulation und insbesondere auch am unterschiedlichen Zeichensatz. Denn Sprache agiert nicht mit Buchstaben, sondern mit Lauten, die von Person zu Person, z. B. durch regionale Dialekte bedingt, stark variieren können. Zunächst muss einem Laut daher ein Phonem zugewiesen werden, von

denen es etwa 100 gibt, und die in einem eigenen Alphabet erfasst werden können. Hierfür gibt es mehrere, international unterschiedlich verbreitete Varianten. Für ein Beispiel siehe z. B. [RN12, S. 1056].

Aber selbst wenn ein Phonem identifiziert werden kann, besteht noch keine 1:1 Beziehung zwischen Phonemen und Buchstaben. Je nach Kontext können viele Buchstaben mit verschiedenen Phonemen artikuliert werden und manche Phoneme werden durch mehrere bzw. unterschiedliche Buchstaben repräsentiert. Ein weiteres Problem des gesprochenen Wortes ist, dass Leerzeichen oft wegfallen. Das heißt, Sprache bindet Wörter häufig aneinander. Manche Buchstaben werden auch gar nicht ausgesprochen, bzw. je nach Sprecher verschluckt. Interpunktion wird durch Tonhöhen- und Pegelunterschiede ausgedrückt (Absenken der Stimme bei einem Punkt, Anhaben der Stimme bei einem Fragezeichen, Lautstärkeanhebung bei einem Ausrufezeichen).

14.3 Methoden

14.3.1 Theoretische Grundlage: Hidden-Markov-Modelle

Für die Erstellung eines Sprachmodells haben sich Hidden-Markov-Modelle (HMM) bewährt. Diese sind ein einfacher Spezialfall von Bayesschen Netzen und werden verwendet, um eine probabilistische Annahme über eine zeitliche Abfolge von Zuständen zu treffen. Ein Zustand ist eine Menge X von unbekannten Zustandsvariablen zu einem Zeitpunkt t (kurz: Xt).

Uns liegen Kenntnisse (eigentlich Annahmen, s. u.) über frühere Zuständen zu jedem Zeitpunkt t-k ($0 < k \leq t$) vor, aus denen wir die Wahrscheinlichkeit für das Eintreten von Xt in einem Übergangsmodell erhalten. Prinzipiell könnten wir alle Zustände von X0 bis Xt-1 (kurz: X0:t-1) auswerten. Dies entspricht dem Übergangsmodell P(Xt | X0:t-1). Da t aber unbegrenzt sein kann, wäre es notwendig, nur eine endliche Menge an vorherigen Zuständen zu betrachten. In Bezug auf die Komplexität und den Aufwand des Modells wäre es zudem von Nutzen, die Anzahl der für die Vorhersage von Xt benötigten vorausgegangenen Zustände möglichst klein zu halten, also nur P(Xt | Xt-n:t-1) mit kleinem n auszuwerten. Nach der Markov-Annahme reicht es für verschiedene Prozesse – darunter auch die Spracherkennung – völlig aus, eine Markov-Kette mit niedriger Ordnung zu verwenden. Eine Markov-Kette n-ter Ordnung ist ein stochastischer Prozess, dessen Zustände nur von n vorherigen Zuständen abhängen. Ein einfaches Beispiel wäre eine Markov-Kette erster Ordnung, bei der nur der unmittelbar vorausgehende Zustand für die Wahrscheinlichkeitsverteilung von Xt betrachtet wird.

Des Weiteren gehen wir davon aus, dass die Gesetze, die die Wahrscheinlichkeit für das Eintreten einer bestimmten Ausprägung von Xt auf Basis der vorherigen Zustände bestimmen, im Laufe der Beobachtung konstant bleiben, also unabhängig von t sind. Das Modell ist also ein stationärer Prozess, für den die Übergangswahrscheinlichkeiten bereits bei der Modellierung festgelegt werden können.

Wie bereits angemerkt treffen wir nur Annahmen über die Zustandsvariablen Xt, da uns die tatsächlichen Werte xt niemals bekannt sind. Grundlage für diese Annahmen ergeben sich nicht nur aus dem Übergangsmodell, sondern aus Beobachtungen/Messdaten, die für eine Menge von bekannten Evidenzvariablen E zu jedem Zeitpunkt t herangezogen werden. Die Ausprägung von Et ist die Menge von Werten et. Nach der Sensor-Markov-Annahme gehen wir davon aus, dass Et nur von Xt und nicht von weiteren Vorgängerzuständen abhängt, somit $P(Et|X0:t, E0:t-1) = P(Et|Xt)$ gilt. Dies ist unser Sensormodell. Letztlich interessiert uns aber natürlich die Gegenrichtung, also $P(Xt \mid Et = et)$. Wir wollen ja Erkenntnisse über Xt gewinnen, während die Daten von Et vorliegen.

Kombinieren wir jetzt Übergangsmodell und Sensormodell ergibt sich für ein beliebiges t die Gleichung $P(X_{0:t}, E_{1:t}) = P(X_0)\Sigma_{i=1}^{t} P(X_i|X_{i-1})P(E_i|X_i)$.

Angemerkt sei, dass unsere Beobachtungen erst bei E1 beginnen, während wir für X0, also unseren Startzustand, eine A-priori-Wahrscheinlichkeitsverteilung P(X0) angeben.

Als Bayessches Netz lässt sich dieses kombinierte Modell z. B. Darstellen wie in [RN12, S. 665].

Hier ist beispielsweise die Wahrscheinlichkeit, dass es an Tag t regnet 70%, wenn es an Tag t-1 geregnet hat, und die Wahrscheinlichkeit, dass eine Person an Tag t einen Regenschirm trägt 90%, wenn es an Tag t tatsächlich regnet. Das Beispiel setzt voraus, dass wir nur den Regenschirm U (Evidenzvariable) beobachten können, nicht aber den Regen R (Zustandsvariable). Ein solches Modell kann natürlich durch Erhöhung der Ordnung (Zeithorizont) und Vergrößerung der Variablenmengen (Messdaten) verbessert werden.

Diese Grundlagen des temporalen probabilistischen Schließens gelten für verschiedene Modelle, von denen uns in Bezug auf die Spracherkennung vor allem das Hidden-Markov-Modell interessiert. In einem HMM wird der Zustand durch eine einzige diskrete Zustandsvariable abgebildet. Das kann auch ein Wertetupel sein, in dem einzelne Variablen gebündelt werden. Dies erlaubt die Erstellung einer Matrix für die Übergangswahrscheinlichkeiten

zwischen zwei Zuständen. Sei S die Anzahl der möglichen Zustände, so erstellen wir aus dem Übergangsmodell $P(Xt|Xt-1)$ die *SxS*-Matrix T mit $Tij = P(Xt = j|Xt-1 = i)$, also die Wahrscheinlichkeit eines Übergangs von Zustand i in Zustand j. Nach dem obigen Beispiel wäre das $T = P(X_t|X_{t-1}) = \begin{pmatrix} 0,7 & 0,3 \\ 0,3 & 0,7 \end{pmatrix}$.

Das Sensormodell wird als *SxS*-Diagonalmatrizen Ot dargestellt. Zunächst benötigen wir für jeden möglichen Zustand $P(et|Xt = i)$. Diese Werte werden in den i-ten Diagonaleintrag geschrieben: $O_1 = \begin{pmatrix} 0,9 & 0 \\ 0 & 0,2 \end{pmatrix}$; $O_3 = \begin{pmatrix} 0,1 & 0 \\ 0 & 0,8 \end{pmatrix}$.

Wobei in diesem Beispiel U1 = true und U3 = false war.

14.3.2 Praktische Anwendung: akustisches Modell

Auf der unteren akustischen Ebene müssen zunächst die Phoneme erkannt werden, d. h. man betrachtet den Zeitverlauf des Audiosignals und schließt anhand eines auf einem HMM beruhenden akustischen Modells auf den Inhalt. Es wäre nicht nur zu aufwendig, sondern auch wenig hilfreich jedes einzelne Sample als Evidenzvariable zu begreifen. Da die menschliche Sprache keine relevanten Informationen unterhalb von 100Hz liefert, bietet es sich stattdessen an, Frames dieser Länge (= 10 ms) zu bilden und die aus diesen Frames gewonnenen Informationen als Folge von Beobachtungen et zu nutzen. Bei 44,1 kHz Samplerate enthält ein solcher Frame also 441 Samples. Da auch die Informationen oberhalb von ca. 4 – 5 kHz für die Spracherkennung nicht mehr relevant sind, bietet auch eine geringere Samplerate von 8 – 12 kHz, wie sie in einfachen Spracherkennungssystemen oftmals verwendet wird, noch eine genügend hohe Auflösung. Hier fasst ein Frame entsprechend 80 – 120 Samples. Um keine relevanten Signaländerungen durch ungünstige Platzierung der Framegrenzen zu verlieren, überlappen sich die Frames.

Das Signal eines jeden Frames wird nun mittels Fourier-Transformation einer Spektralanalyse unterzogen und die Energieniveaus von 12 verschiedenen Frequenzen sowie des gesamten Frames ausgewertet. Zusätzlich wird die Differenz zu den Werten des vorherigen Frames sowie die Differenz der Differenzen berechnet. Folglich stehen nun 39 Werte zur Verfügung die in einem Vektor gespeichert werden und unsere Evidenzmerkmale Et für unsere Lautzustände Xt darstellen. Somit haben wir das Sensormodell $P(Et|Xt)$ definiert. Um aus dem Verlauf von Frames auf ein Phonem schließen zu können wird zunächst ein Übergangsmodell für Laute verwendet, das aus einem Anfang-, Mitte- und Endzustand besteht. Damit können charakteristische Merkmale eines Lautes, wie Beginn und Endung, gut abgebildet werden. Jeder Zustand kann durch Schleifen mehrfach hintereinander durchlaufen werden, womit sich verschieden lang gesprochene Phoneme modellieren lassen. Jeder Laut hat sein eigenes charakteristisches Laut-HMM. Ein Beispiel für ein Laut-HMM ist z.B. in [RN12, S. 1058] zu finden.

Wir benötigen aber nicht nur Lautmodelle, sondern auf der nächst höheren Ebene auch Übergangsmodelle für Wörter, also für jede Vokabel ein Wortmodell. Das Prinzip ist hierbei ähnlich wie bei den Laut-HMMs, allerdings verwenden wir keine Schleifen, da Phoneme nicht ungewollt wiederholt werden. Stattdessen benötigen wir alternative Pfade für alternative Aussprachen. Ein Wort kann z. B. aufgrund einer schnellen, gebundenen Sprechweise (Koartikulation) oder aufgrund eines Dialekts unterschiedlich ausgesprochen werden.

In einem solchen Wortmodell-Graphen besteht also jeder Knoten aus einem Lautmodell. Beide Ebenen ergeben zusammen das Übergangsmodell P(Xt | Xt-1). Ein Beispiel für ein Wortmodell-Graphen ist z. B. in [RN12, S. 1058] zu finden.

Beim Sprechen werden Worte oft aneinandergebunden (fehlende Segmentierung). Durch ungenaue Aussprache oder sogar dem Gleichlaut zweier Worte (Homophone) besteht darüber hinaus Verwechslungsgefahr. Es kann daher sinnvoll sein, die Wahrscheinlichkeit des Auftretens eines Wortes P(wort) bzw. einer Wortfolge P(wort-n:t) im Kontext zu betrachten und die wahrscheinlichste Folge hiervon anzunehmen. Um diese Wahrscheinlichkeitswerte zu ermitteln, bietet es sich an, auf einen Textkorpus zurückzugreifen. Idealerweise wird dieser aus gesprochener Sprache (Transkriptionen) gespeist, da Schrifttexte oft eine andere Struktur aufweisen. Da die Erstellung eines solchen Korpus sehr aufwendig ist und um die Komplexität der Analyse zu reduzieren, werden oft aufgabenspezifische Korpora verwendet. Das heißt wir begnügen uns für einfachere Systeme mit denjenigen Vokabeln, die für eine gegebene Anwendung relevant sind.

14.4 Sprachanalyse

14.4.1 Problemstellung

Im vorherigen Abschnitt haben wir Grundlagen kennengelernt, wie man mittels Spracherkennung gesprochene Sprache in einen Text transformiert, wobei viele Probleme (z. B. Interpunktion) noch ausgeklammert wurden. Bei der Sprachanalyse möchten wir nun einen Schritt weiter gehen und den Inhalt eines Textes logisch erfassen. Diese Analyse erfolgt in zwei Schritten. In einer syntaktischen Analyse ermitteln wir zunächst die Phrasenstruktur eines Satzes, um die Satzelemente in Subjekt, Prädikat und Objekt einteilen zu können. Wir bedienen uns hierfür einer Grammatik, die eine natürliche Sprache definiert. Mit deren Hilfe können wir einen Text parsen, das heißt ausgehend von seinen Zeichen die strukturellen Elemente identifizieren. Wir bestimmen dabei auch weitere Satzelemente wie Attribute (Adjektive, Adverbien), Tempus, Kasus, Numerus etc. Im nächsten Schritt wollen wir auf der semantischen Ebene mit Hilfe von Prädikatenlogik einen Zusammenhang zwischen den nun bekannten Satzelementen herstellen und damit den Inhalt des Textes schließlich erfassen. Jedenfalls wäre dies die Idealvorstellung einer Textanalyse. In der Praxis ergeben sich zahlreiche Schwierigkeiten. Zunächst ist es nicht möglich ein lückenloses Vokabular und eine vollständige Grammatik einer natürlichen Sprache zu erstellen. Natürliche Sprachen haben zahlreiche (z. B. regionale) Unterschiede und insbesondere das Vokabular ist in einem stetigen Fluss begriffen. Sprecher einer Sprache halten sich selber nicht immer streng an die Sprachregeln und können meistens dennoch verstanden werden. Grammatiken natürlicher Sprachen sind zudem hochkomplex, enthalten zahlreiche Ausnahmeregeln und sind daher nur schwer in ein festes Regelwerk zu überführen. Wir müssen uns daher vorerst mit einer Teilmenge einer Sprache begnügen, also mit einem verkürzten Vokabular und einer reduzierten Grammatik. Im Folgenden beschäftigen wir uns vor allem mit der englischen Sprache, da diese zu den grammatikalisch einfachsten natürlichen Sprachen gehört. Das Textverständnis mit Mitteln der Logik ist ebenfalls schwierig zu implementieren, weil hierfür eine umfassende Logik-Datenbank der Begriffe, ihrer Bedeutungen und Zusammenhänge vorhanden sein müsste, auf die bei der Interpretation zugegriffen werden könnte.

14.4.2 Methoden

Eine Grammatik definiert eine Sprache mit Hilfe von Regeln. Anhand dieser werden abstrakte Nichtterminalsymbole, die eine Struktur markieren, durch andere Nichtterminale und/oder Terminalsymbole (hier: die Vokabeln einer Sprache) ersetzt, bis nur noch Terminal-Symbole vorhanden sind, die einen grammatikalisch korrekten Text ergeben. Eine solche Regel hat beispielsweise die Form $S \rightarrow a\ S\ b$. Nichtterminale sind hier S, Terminale sind a, b. Bei der Erstellung einer Grammatik beginnen wir bei einem Startsymbol, das sich zu feineren Strukturen und schließlich zum einem fertigen Text hin immer weiter verzweigt. Ein Text lässt sich also in einem Ableitungsbaum darstellen, dessen Wurzel das Startsymbol bildet. Seine inneren Knoten sind Nichtterminale und die Blätter Terminale. Ein Ableitungsbaum für den aus obiger Grammatik erzeugten Text „aabb“ sähe entsprechend so aus:

Haben wir nun einen Text und die Grammatik einer natürlichen Sprache gegeben, können wir umgekehrt auch überprüfen, ob ein in dieser Sprache gültiger Text vorliegt, indem wir die Grammatikregeln quasi von rechts nach links betrachten. Dabei prüfen wir, ob passende Satzstrukturen gefunden werden können. Wir bauen den Ableitungsbaum also von den Blättern her auf. Dieser Vorgang wird Parsen genannt, entsprechend spricht man auch von einem Parse- oder Syntaxbaum. Auf dem Weg von den Blättern zur Wurzel des Baumes lernen wir die Satzstruktur kennen und können uns dadurch den Inhalt des Textes erschließen. Die in vielen Implementierungen von Prolog vorhandene Erweiterung DCG (definite clause grammars) erlaubt es sehr einfach, formale Grammatiken in Prolog zu implementieren. Die Syntax einer solchen Grammatik ähnelt dem obigen Beispiel mit kleinen Abweichungen, die sich schnell aus dem Kontext erschließen. Im Folgenden wird die DCG-Syntax verwendet.

Betrachten wir eine sehr reduzierte Grammatik für eine Teilmenge der englischen Sprache. Wir folgen dem Beispiel aus [Bra12, Kapitel 23].

```
sentence --> noun_phrase, verb_phrase.
verb_phrase --> verb, noun_phrase.
noun_phrase --> determiner, noun.
determiner --> [the].
noun --> [cat].
noun --> [mouse].
verb --> [hates].
```

Terminale werden in DCG als Prolog-Listen in eckigen Klammern angegeben. Prolog konvertiert diese DCG-Regeln in Prolog-Regeln. Die erste Regel hat konvertiert die Form

```
sentence (List, Rest) :-
 noun_phrase(List, List1),
 verb_phrase(List1, Rest).
```

Abfragen für einen Text auf diese Grammatik haben die Form

```
?- sentence([the, mouse, hates, the, cat], []).
yes
```

Prolog verwendet hier als Parameter Differenzlisten, also die Differenz aus List und Rest.
Fügen wir der Grammatik als zusätzliche Terminale die Pluralform der Substantive und Verben hinzu,

```
noun --> [cats].
noun --> [mice].
verb --> [hate].
```

entsteht das Problem, dass unsere Grammatik noch keinen Kontext zwischen Nominalphrase und Verbphrase herstellt. Eigentlich inkorrekte Sätze wie „the mice hates the cat“ werden noch akzeptiert (Übergenerierung). Wir müssen den Nichtterminalen für Subjekt und Prädikat also einen Parameter für den Numerus mitgeben, der für beide Phrasen identisch ist. In DCG lässt sich eine solche Parametrisierung folgendermaßen realisieren:

```
sentence(Number) --> noun_phrase(Number), verb_phrase(Number).
verb_phrase(Number) --> verb(Number), noun_phrase(Number1).
noun_phrase(Number) --> determiner(Number), noun(Number).
noun(singular) --> [mouse].
noun(plural) --> [mice].
verb(singular) --> [hates].
verb(plural) --> [hate].
...
```

Fragen müssen ebenfalls um diesen Parameter modifiziert werden:

```
?- sentence(plural, [the, mice, hates, the, cat], []).
no
?- sentence(plural, [the, mice, hate, the, cat], []).
yes
?- sentence(Number, [the, mice, hate, the, cat], []).
Number = plural
```

Je präziser eine Grammatik eine natürliche Sprache repräsentieren soll, desto aufwendiger wird auch die Konstruktion nichtterminaler Parameter.

Zur Analyse der Bedeutung eines Textes ist es wie zuvor erwähnt hilfreich, einen Syntaxbaum zu erstellen. Ein solcher hat für das letzte Beispiel die Form

In Prolog kann ein solcher Baum mittels DCG erstellt werden, indem jedem Nichtterminal der Teilbaum als Argument übergeben wird, für den dieses Nichtterminal die Wurzel darstellt. Unsere Grammatik wird so erweitert zu

```
sentence(Number, sentence(NP, VP)) -->
  noun_phrase(Number, NP),
  verb_phrase(Number, VP).
verb_phrase(Number, verb_phrase(Verb, NP)) -->
  verb(Number, Verb),
  noun_phrase(Number1, NP).
```

```
noun_phrase(Number, noun_phrase(Det, Noun)) -->
  determiner(Number, Det),
  noun(Number, Noun).
determiner(determiner(the)) --> [the].
noun(singular, noun(cat)) --> [mouse].
noun(plural, noun(cat)) --> [mice].
verb(singular, verb(hates)) --> [hates].
verb(plural, verb(hate)) --> [hate].
...
```

Eine dazu passende Anfrage wäre

```
?- sentence(Number, ParseTree, [the, mice, hate, the, cat], []).
Number = plural
ParseTree = sentence(noun_phrase(determiner(the), noun(mice)),
            verb_phrase(verb(hate), noun_phrase(determiner(the),
            noun(cat))))
```

In einem Satz stellt das Verb eine Beziehung zwischen Subjekt und eventuellen Objekten her. Grundsätzlich unterscheidet man intransitive und transitive Verben. Letzteren ist ein Objekt zugeordnet, auf das die Handlung des Subjekts gerichtet ist. Die Beziehungsstrukturen einfacher Sätze lassen sich gut in Prädikatenlogik formalisieren: intransitives_verb(subjekt), transitives_verb(subjekt, objekt). Der einfache Satz „Alice paints“ mit intransitivem Verb ‚paints‘ wird so zu paints(alice) umgeformt. „Alice likes Bob“ mit transitivem Verb ‚likes‘ wird zu likes(alice, bob). Betrachten wir nun, wie sich diese Prädikatenlogik in Prolog mit DCG darstellen lässt. Eine Möglichkeit wäre, die Semantik aus dem Syntaxbaum herzuleiten. Es gibt aber noch eine andere gebräuchliche Option, die Bedeutung der Syntax in der Grammatik zu enkodieren und durch nach außen sichtbare Parameter für Abfragen zugänglich zu machen. Eine Grammatik mit der sich die einfachen Sätze „Alice paints“ und „Alice likes Bob“ und ihre Bedeutung definieren lassen, hätte mit dieser Syntax folgende Form:

```
sentence(VP) --> noun_phrase(X), verb_phrase(X, VP).
noun_phrase(NP) --> proper_noun(NP).
verb_phrase(X, VP) --> intrans_verb(VP).
verb_phrase(X, VP) --> trans_verb(X, Y, VP), noun_phrase(Y).
intrans_verb(X, paints(X)) --> [paints].
trans_verb(X, Y, likes(X, Y)) --> [likes].
proper_noun(alice) --> [Alice].
proper_noun(bob) --> [Bob].
```

Dadurch, dass wir die Bedeutung der Terminale in den Nichtterminalen als Argumente kodieren, und beim Parsen des Textes an die darüber liegenden Satzstrukturen als Parameter weiterreichen, können wir die logische Struktur eines Satzes herleiten. Der Parameter X wird wie im Fall intrans_verb(X, paints(X)) noch einmal separat vorangestellt, um von außen sichtbar zu sein, so dass wir eine Prolog-Anfrage auf ihn formulieren können.

Verwenden wir nicht konkret bezeichnete Subjekte (hier: bestimmte Personen), sondern abstraktere Akteure, wie ‚a woman‘ oder ‚every woman‘, so benötigen wir für den Logik-Ausdruck zusätzlich Quantoren. „a woman paints“ müssen wir dann ausdrücken durch: Es existiert ein x, dieses x ist eine Frau und x malt $(\exists x : woman(x) \wedge paints(x))$. Den Allquantor nutzen wir entsprechend für bspw. „every woman likes Bob“: $\forall x : woman(x) \Rightarrow likes(x, bob)$. Der Akteur ist hier eine Property und die Handlung eine Assertion. Allgemein hätte eine solche Aussage in Prolog die Form exists(X, Property and Assertion) bzw. all(X, Property => Assertion). Für die Konstruktion einer solchen Aussage nimmt natürlich auch der grammatikalische Aufwand deutlich zu:

```
sentence(S) --> noun_phrase(X, Assn, S), verb_phrase(X, Assn).
noun_phrase(X, Assn, s) --> determiner(X, Prop, Assn, S),
                            proper_noun(NP, Prop).
noun_phrase(X, Assn, Assn)2 --> proper_noun(X).
```

```
verb_phrase(X, Assn) --> intrans_verb(X, Assn).
intrans_verb(X, paints(X)) --> [paints].
Determiner(X, Prop, Assn, exists(X, Prop and Assn)) --> [a].
noun(X, woman(X)) --> [woman].
proper_noun(alice) --> [Alice].
```

Anfragen an diese Grammatik wären beispielsweise:

```
?- sentence(S, [a, woman, paints], []).
S = exists(X, woman(X) and paints(X))

?- sentence(S, [Alice, paints], []).
S = paints(alice)
```

In diesem Beispiel wurde der Übersicht halber sogar auf Allquantoren und transitive Verben verzichtet. Man bekommt aber bereits einen guten Eindruck davon, wie aufwendig eine solche Grammatik schon für einfache Sätze wird. Es ist leicht einsehbar, dass eine Grammatik, die weitere grammatikalische Eigenheiten wie Tempus, Numerus, Kasus, Fragen, Befehle, Konjunktivsätze, Nebensätze (z. B. Beziehungen in Relativsätzen), Satzzeichen, Sonderformen und Ausnahmeregeln berücksichtigen muss, einen sehr hohen Komplexitätsgrad annimmt. Und dabei haben wir es hier mit einer grammatikalisch noch verhältnismäßig einfachen Sprache wie Englisch zu tun. Noch schwieriger gestaltet es sich allerdings, einen Logik-Korpus zu generieren, der die Bedeutung eines umfassenden Vokabulars einer Sprache umfasst. Ein solches kann einschließlich Verkleinerungsformen und vergleichbaren Wortabwandlungen bis zu 1 Millionen Begriffe enthalten. Diese müssten so kodiert sein, dass ähnliche Begriffe auch gemeinsam kategorisiert sind. Aber auch damit wäre noch keine eigentliche Kommunikation mit einem maschinellen Agenten möglich. Hierzu benötigt dieser zusätzlichen eine umfassende Wissensdatenbank in Logikform mit Beziehungen zwischen Begriffen, Akteuren und Weltwissen. Offensichtlich gerät die Sprachverarbeitung hier noch an ihre Grenzen. Daher basieren Sprachanalysesysteme, die zum Beispiel für die maschinelle Übersetzung eingesetzt werden, in der Regel auch auf probabilistischen Modellen, die große Ähnlichkeit zu den Methoden der Spracherkennung aufweisen, und zum Beispiel auf Hidden-Markov-Modellen beruhen können.

14.5 Verwandte Themen

Hidden-Markov-Modelle sind ein Spezialfall der Bayesschen Netze. Einen weiteren Aspekt von Sprachverarbeitung ist die Digitalisierung von Schrift, sei es gedruckt oder handschriftlich. Dies fällt aber in den Bereich der Objekterkennung/Computervision und wurde hier daher ausgeklammert. Ähnliches gilt für die Erkennung von Gebärdensprache.

14.6 Fazit

Die Sprachverarbeitung ist gewissermaßen die Königsdisziplin in der Entwicklung künstlicher Intelligenz. Nicht zuletzt basiert der Turing-Test auf sprachlicher Kommunikation mit maschinellen Agenten. Dass der Turing-Test noch von keiner künstlichen Intelligenz zufriedenstellend bestanden wurde, deutet darauf hin, dass auch die Sprachverarbeitung noch vor vielen Problemen steht. In diesem Kapitel wurden zwei Ansätze vorgestellt, die verschiedene Anwendungsfälle abdecken. Im ersten Abschnitt wurde mit einer zustandsbasierten, probabilistischen Methode, einem Hidden-Markov-Modell, gesprochene Sprache in Text umgewandelt. Im zweiten Abschnitt wurden Ansätze beschrieben, um Text mit Mitteln formaler Sprachen und Logik systematisch zu strukturieren und auf ihre Bedeutung hin zu dekodieren. Während probabilistische Ansätze etwa durch Spracherkennung oder zum maschinellen Übersetzen bereits in spezifischen Einsatzbereichen gute Ergebnisse erzielen, ist die systematische Formalisierung natürlicher Sprachen und ihre Überführung in ein Logiksystem noch vor allem Forschungsgegenstand. Es lässt sich darüber streiten, ob nicht erst mit der Entwicklung einer solchen strukturellen Sprachverarbeitung von echter künstlicher Intelligenz gesprochen werden kann, wenn also ein künstlicher Agent also nicht nur – wie im bereits bekannten Chinese Room Gedankenexperiment – lediglich Grammatikregeln befolgt, sondern seinen Wissensschatz in der Kommunikation abfragen, anpassen und erweitern kann. Die probabilistischen Ansätze, die in der Praxis Anwendung finden, sind zwar sehr nützlich und oft recht erfolgreich, scheinen mir jedenfalls aber nur eine Simulation von Intelligenz zu darzustellen.

14.7 Quellen und Literatur

- Russell, Stuart / Norvig, Peter: Künstliche Intelligenz. Ein moderner Ansatz. 3., aktualisierte Auflage. München: Pearson 2012.
  - Kapitel 15: Probabilistisches Schließen über die Zeit
  - Kapitel 22: Verarbeitung natürlicher Sprache
  - Kapitel 23: Natürliche Sprache für die Kommunikation
- Bratko, Ivan: Prolog Programming for Artificial Intelligence. Fourth Edition. Essex: Pearson 2012.
  - Kapitel 23: Language Processing with Grammar Rules (hieraus Codebeispiele im Abschnitt Sprachanalyse)
- MIT Open Course: Artificial Intelligence. Fall 2010
  - Lecture 14: Learning: Sparse Spaces, Phonology `http://ocw.mit.edu/courses/electrical-engineering-6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-14-learning-sparse-spac`

VI Philosophische & Ethische Aspekte

15. Intelligenzbegriff, Wahrnehmung & Bewusstsein

15.1 Intro & Motivation

Intelligenzbegriff, Wahrnehmung und Bewusstsein bilden als das, was wir mittels einer Künstlichen Intelligenz simulieren das Fundament für das Umsetzen eben dieser. Je besser man nun also diese Grundlagen versteht, desto besser kann man diese dann natürlich auch umsetzen. Das Definieren der Begriffe Intelligenz und insbesondere von Bewusstsein überschneidet sich dabei schon mit der Philosophie und wie es dieser Bereich an sich hat, kommen natürlich auch Fragen auf, auf die jeder eine andere Antwort hat.

So haben wir direkt zu Beginn des Seminars darüber diskutiert, wie man Intelligenz definieren könnte. Da allerdings ähnlich viele, teils drastisch verschiedene, Meinungen wie Personen existieren, blieben wir ohne ein einheitliches Ergebnis. Zum Beispiel beim Chinese Room Experiment wurde gezeigt, wie ein Mensch sich mittels Text auf Chinesisch unterhält, indem er einfach If-Then-Regeln befolgt, ohne auch nur ein bisschen Chinesisch zu verstehen, was im Endeffekt genau das ist, was ein Rechner tut. Egal wie kompliziert diese Regeln sind. Ist der Algorithmus wirklich intelligent? Kann er auf diese Weise eine Stufe erreichen, wo man davon sprechen kann, dass die KI ein Bewusstsein entwickelt hat?

Über Bewusstsein einer KI wären wohl ähnlich viele verschiedene Meinungen vorgekommen, wenn die Diskussion in diese Richtung gegangen wäre. Dass diese Themen so kontrovers sind, liegt vermutlich an hauptsächlich zwei Dingen:

Der Mensch kann sich nicht wirklich vorstellen, wie er selbst funktioniert, selbst wenn, wie wir ja bereits in Neuronale Netze gehört haben, die Grundlagen des Gehirns bereits bekannt sind, womit eben auch keine einheitlichen Definitionen für die Begriffe existieren. Zweitens sind diese Themen für viele auch etwas Emotionales. Welcher Mensch mag es schon Argumente dafür zu hören, dass er durch einen Computer simuliert werden könnte, auch wenn schon heute KIs gerade aufgrund ihres Tempos und Spezialisierung auf einzelne Bereiche dem Menschen bei weitem überlegen sind und sie mit den Jahren immer stärker werden. Das Potential von Künstlichen Intelligenzen lässt sich dabei noch nicht absehen. Die Hardware wird immer schneller, der Speicher immer größer und gerade die Stärke von neuronalen Netzwerken steigt dadurch schnell. Jede Abschätzung, auf welchem Level das ganze endet, kann falsch sein. Selbst Alan Turing lag was seine Einschätzungen für die heutige Zeit betrifft teils komplett falsch (zum Beispiel, dass Maschinen niemals lernen können werden), insofern ist jede Abschätzung reine Spekulation.

15.2 Inhaltliche Ausarbeitung des Themas

15.2.1 Intelligenzbegriff

Intelligenz kommt vom lateinischen „intellegere, was „verstehen“ bedeutet. Diese Übersetzung macht es uns leider nicht wirklich einfacher. Verstehen ist ein genauso abstrakter Begriff, wie es schon Intelligenz war, womit es nicht

lösbarer klingt Verständnis, anstatt Intelligenz umzusetzen.

Einfacher wird es mit der wörtlichen Übersetzung „Wählen zwischen“. Dies hat es damit allerdings auf einen Schlag soweit vereinfacht, dass nun jedes Programm, welches zwischen Möglichkeiten wählen kann, also eine je nach Eingabe unterschiedliche Lösung ausgeben kann, als intelligent bezeichnet werden könnte.

Die Meinungen zur Intelligenz-Definition lagen im kompletten Spektrum zwischen diesem komplizierten „Verstehen“ oder dem ganz simplen „Wählen zwischen“. Ob es überhaupt möglich ist, einem Programm das Komplizierte beizubringen, ist nicht ersichtlich. Dagegen genügen dem simplen Kriterium so gut wie alle sinnvollen Programme. Intelligenz ist damit also nicht unbedingt ein Punkt, ab dem man sagen kann, dass ein Programm intelligent ist, sondern eher eine Skala, wie hoch seine Intelligenz ist.

Das, was man normalerweise damit meint, wenn man ein Programm intelligent machen will, ist, dass es ähnlich arbeiten soll wie ein Mensch, also dass es selbst in dem Bereich, in dem es arbeiten wird, übt. Allerdings hat es natürlich immer noch die Vorteile eines Programms, kann also die Übungsdaten in kürzester Zeit verarbeiten, damit natürlich auch sehr schnell lernen und letztlich die Arbeit wesentlich schneller erledigen als ein Mensch, der vorher vielleicht Jahre gebraucht hat um genauso gut in dem jeweiligen Bereich zu werden.

Außerdem haben Rechner wesentlich genaueren Speicherzugriff und können Muster auf Daten erkennen, wo ein Mensch längst keines mehr sehen würde. So ist zum Beispiel die KI AlphaGo dem besten menschlichen Go-Spieler bei weitem überlegen und nutzt dabei teilweise Züge, die für einen Menschen längst nicht mehr nachvollziehbar sind, allerdings gleichzeitig ausreichen um gegen jegliche menschliche Spieler zu gewinnen. Diese KI ist eines der bekanntesten Beispiele für eine KI, welche mit neuronalen Netzen arbeitet. Sie ist so stark, obwohl gerade Go lange als besonders schwieriges Spiel für eine KI galt, da es extrem komplex ist. Es lässt sich nicht mit klassischen Algorithmen lösen.

Die vorher bekannteste KI, der Schachcomputer Deep Blue, welcher den Alpha-Beta Algorithmus nutzt, war noch wesentlich unselbstständiger und musste um gegen den amtierenden Schachweltmeister zu gewinnen noch häufig von den Entwicklern modifiziert werden. Deep Blue hatte also nichts mit dem System, nach dem ein Mensch arbeitet, zu tun, da er die sinnvollen Züge anhand des Vorausberechnens sämtlicher weiterer Züge auswählt.

15.2.2 Wahrnehmung

Wahrnehmung ist etwas unkomplizierter zu verstehen. Wie das Wort wahrnehmen schon sagt ist es das, was die Intelligenz als wahr nimmt. Ohne etwas, was die Intelligenz wahrnimmt, hat sie nichts, was sie verstehen oder zwischen dem sie wählen könnte, womit selbst die schlauste Intelligenz rein gar nichts kann.

Beim Menschen ist das einerseits Exterozeption, das ist das, was er über seine Sinne aufnimmt. Eine KI, die einen Menschen simuliert, braucht entsprechend Sensoren, die die Rolle der Sinne übernehmen. Da die menschlichen Sinne über elektrische Signale funktionieren, lässt sich dies so gut wie 1 zu 1 umsetzen, wobei Sensoren noch den Vorteil haben, dass sie wesentlich präziser ausgewertet werden können und auch mit der richtigen Bauweise unter extremen Bedingungen eingesetzt werden können.

Insofern kann man mit genug Aufwand auch die Wahrnehmung einer KI den Sinnen des Menschen überlegen umsetzen. So können Sensoren zum Beispiel Temperaturen aushalten, an dem die menschlichen Nerven längst absterben. Außerdem nehmen wir sowieso keine genauen Temperaturen wahr.

Auf der anderen Seite steht Interozeption, die Selbstwahrnehmung. Diese ist grundsätzlich bei einer KI nicht anders als die Exterozeption. Man kann ebenfalls Sensoren benutzen, um den eigenen Zustand zu überwachen, nur dass diese dann eben zum Beispiel Computerkomponenten überwacht, anstatt wie beim Menschen Organe.

15.2.3 Bewusstsein

Bewusstsein ist noch schwerer zu definieren als der Intelligenzbegriff. Um genau zu sein gibt es schlicht keine einheitliche Definition. So listet Wikipedia vier für uns relevante verschiedene Definitionen für Bewusstsein:

- Phänomenales Bewusstsein: Das Erleben des Wahrgenommenen, statt dem reinen Verarbeiten.
- Gedankliches Bewusstsein: Wer denkt, plant, erwartet und sich erinnert hat ein Bewusstsein.
- Selbstbewusstsein: Das Bewusstsein, dass man Phänomenales oder Gedankliches Bewusstsein besitzt.
- Individualitätsbewusstsein: Benötigt Selbstbewusstsein und das Bewusstsein der eigenen Einzigartigkeit.

Das Gedankliche Bewusstsein kann man wohl auch einem genügend kompliziertem Algorithmus zuschreiben. Das Erinnern kann sich ein Algorithmus, wenn ihm gesagt wird, dass er Dinge abspeichern soll. Zukunft erwarten lässt sich mittels Methoden der Stochastik anhand der Erinnerung, also dem Speicher umsetzen. Auch Planen lässt sich mittels des Erwartens der Zukunft anhand von eigenen Verhaltensweisen ebenfalls umsetzen. Das Denken selbst

ist wieder etwas abstrakter, aber man könnte das Denken als das Verarbeiten definieren, womit ein Algorithmus Gedankliches Bewusstsein haben kann.

Bei phänomenalem Bewusstsein lässt sich dagegen nicht sagen, ob es umsetzbar ist. Ein Algorithmus verarbeitet, aber ob er etwas erleben, anders gesagt fühlen kann, ist die Frage. Natürlich kann man ihm Variablen für das, was er fühlt, geben. Zum Beispiel könnte man ohne weiteres Dopamin simulieren, um das Glück des Algorithmus umzusetzen und dies anhand des Erlebten beeinflussen. Ob das dafür reicht, dass das Programm „erlebt“, bleibt wohl der Meinung von jedem einzelnen überlassen.

Selbstbewusstsein ist natürlich erst einmal von der Existenz der vorherigen Bewusstseinsarten abhängig, also gehen wir erst einmal einfach davon aus, dass wir eine solche KI besitzen. Wenn dies der Fall ist, müssen wir der KI die Funktion geben, dass es weiß, dass es diese besitzt. Da wir bereits das Erinnern, oder Wissen durch das Gedankliche Bewusstsein haben, können wir in dieses natürlich auch das Wissen einspeisen, dass es diese hat.

Wenn wir nun also wieder das Selbstbewusstsein in einer KI haben, müssen wir ihr nur noch das Wissen geben, dass es keine KI gibt, die genau wie sie ist. Dies ist genau wie beim Selbstbewusstsein theoretisch möglich. Dieses Wissen kann allerdings falsch sein, da wir eine KI natürlich 1 zu 1 kopieren können. Allerdings wäre dies auch nicht anders, als bei einem Menschen, da es theoretisch möglich ist einen Menschen zu kopieren, wenn man jedes bisschen von ihm nachbauen würde.

Letztlich liegt es damit ausschließlich daran, ob es möglich ist einen Algorithmus erleben zu lassen, ob es möglich ist ihm die letzten drei Bewusstseinsarten zu geben. Auf der anderen Seite ist die Frage, ob ein Mensch wirklich erlebt, was ja das Kriterium für diese Art Intelligenz ist. Vielleicht verarbeiten wir schlicht nur und Gefühle sind ein Teil dieser Verarbeitung, womit phänomenales Bewusstsein schlicht nicht möglich ist.

15.2.4 Methoden

Intelligenz

Die unterste Stufe eines intelligenten Programms, die wir ermittelt haben, war das Ausgeben einer Lösung anhand von Eingaben. Das sind schlicht Programme mit If-Then-Regeln, entsprechen also den Expertensystemen aus dem Anfang des Seminars.

Die dem Menschen nächste Stufe der künstlichen Intelligenz sind die neuronalen Netze, die auf der Gehirnstruktur von Lebewesen basiert und mit genügend simulierten Neuronen und Verbindungen sehr nah an uns selbst sein könnten.

Alle anderen Methoden, die wir bereits hatten, sind was das angeht zwischen diesen beiden Systemen einzuordnen. Diese Systeme haben alle eines gemein. Sie sind, wenn sie genügend kompliziert gemacht werden, Lebewesen bei weitem überlegen, allerdings gibt es praktisch dabei einige Probleme: Zum Beispiel bei den Expertensystemen bräuchte man um einen Menschen zu simulieren unglaublich große Mengen an Regeln.

Für neuronale Netze ist die Technik bei weitem noch nicht so weit die 100 Milliarden Neuronen mit ihren je 50000, also 5 Billiarden, Verbindungen zu simulieren.

Ein in der Informatik geläufiges Intelligenz-Kriterium, wobei es eher ein Kriterium ist, wann ein Algorithmus so schlau ist wie ein Mensch, ist der Turing-Test. Ein Mensch kommuniziert mit dem Algorithmus und darf, damit der Test bestanden ist, nicht mehr unterscheiden können, ob er gerade eine Maschine oder einen Menschen vor sich hat, wobei der Mensch es auch mit gezielten Fragen versuchen kann herauszufinden.

Als Kritik an diesem Kriterium wird immer wieder genannt, dass sich die Maschine nicht bewusst ist, was sie tut. Einer der Verfechter dieser Meinung ist John Searle, der Erfinder des Chinese-Room-Experiments. Bei diesem wird ein Mensch, der kein Chinesisch spricht, quasi als Prozessor, in einen abgeschlossenen Raum gesetzt. Er enthält dabei ein Buch mit Anweisungen, um alle chinesischen Sätze beantworten zu können. Wenn jetzt ein chinesisch sprechender Mensch solche Sätze in diesen Raum hineinwirft, kann dieser Mensch diese also beantworten. Wenn dieses Buch perfekt ist, besteht dieser Raum also den Turing-Test.

Laut Searle hat der Mensch kein Bewusstsein, was er da tut, da er ja kein Chinesisch spricht und die Regeln in dem Buch können auch kein Bewusstsein haben und damit fehlt aus seiner Sicht Intelligenz in diesem Raum.

Dies wird ebenfalls wieder kritisiert. Einerseits gibt es das Argument, dass die Intelligenz die Person ist, die das Buch, also den Algorithmus geschrieben hat. Andererseits, dass schlicht der Raum als ganzes die Intelligenz ist, da er ja auch als ganzes Chinesisch spricht. Der Mensch nimmt allein nur die Rolle des Prozessors ein, der die Regeln befolgt, aber als ganzes beherrscht ein Rechner mit Prozessor, Speicher, sowie natürlich dem Algorithmus Chinesisch. Beim Menschen ist das entsprechende schlicht im Gehirn zusammengefasst, also lässt sich der perfekte Rechner als ganzes theoretisch als genauso intelligent bezeichnen wie das menschliche Gehirn.

Wahrnehmung

Sensoren werden in Physik und Chemie zuhauf eingesetzt, sind also noch der einfachste Teil einer KI. Vorausgesetzt eine Schnittstelle dafür ist programmiert, kann sie damit alles wahrnehmen, was ein Lebewesen wahrnehmen kann, und, wie schon vorher gesagt, präziser.

So lässt sich zum Beispiel mittels eines Temperatursensors die Temperatur sehr genau aus dem Widerstand des Sensors berechnen. Platin-Messwiderstände können dabei zum Beispiel zwischen Temperaturen von -200 °C und 850 °C eingesetzt werden, Temperaturen, in denen niemand überleben könnte. Dies ist dem Menschen bei weitem überlegen, der einen wesentlich kleineren Bereich hat, den er verträgt, und dabei auch nur relative Temperaturen zu sich wahrnimmt.

Viele Sensoren erkennen auch Dinge, die für den Menschen schlicht nicht wirklich bemerkbar sind, wie zum Beispiel Magnetfelder.

Bewusstsein

Da wir selbst nicht einmal annähernd verstehen, was Bewusstsein wirklich ist, ist das Umsetzen einer bewussten KI bisher ein ungelöstes Problem. Selbst wenn wir den Rechner hätten, der genau gleich intelligent ist wie ein Mensch und damit den Turing-Test besteht, kann man nie wirklich sagen, dass der Mensch und die Maschine genau gleich bewusst sind, einfach weil man nie wirklich sagen kann, was uns bewusst macht.

Wenn der Mensch genau wie der Computer nur eine Summe seiner Teile ist, also selbst nur eine sehr komplexe Maschine, bedeutet das, dass er und der Computer beide Bewusstsein haben.

Wenn dies nicht der Fall ist, ist ein Computer mit einem Bewusstsein vielleicht niemals möglich, egal wie komplex er ist. Es lässt sich schlicht nicht eindeutig sagen, was einen Menschen zum Menschen macht, oder allgemeiner ein Lebewesen zu einem Lebewesen. Neuronale Netzwerke, die ausreichend komplex sind, sollten zwar ausreichen, um eine KI zu bauen, welche sich in jeder Situation genau gleich verhält wie der Mensch, aber ob sie auch wirklich gleich ist, kann man nur mit dem uns fehlenden Verständnis von uns selbst sagen können.

Gerade an dieser Stelle wird das Thema besonders philosophisch und spekulativ. Viele Menschen würden vermutlich zum Beispiel mit einer Seele dagegen argumentieren, dass man einer Maschine Bewusstsein geben kann, aber es gibt schlicht keinen Nachweis von Dingen, die man nicht auch technisch in einer Maschine umsetzen könnte.

Dadurch, dass wir dies alles nicht wissen, kennen wir auch nicht die Wirkung, was passieren würde, wenn es gelänge einer KI Bewusstsein zu geben. Vielleicht würde sich gar nichts ändern. Vielleicht folgt die von Filmen immer wieder beschworene Roboterapokalypse, in der wir von unserer eigenen Schöpfung beherrscht werden. Vielleicht folgt das auch schon ohne das Bewusstsein, wenn es den Zielen, die der KI gegeben sind, am ehesten entspricht.

15.3 Einordnung von vorherigen Kapiteln

Wir haben einige verschiedene Arten von KIs kennengelernt. Angefangen mit Expertensystemen, die in ihren Gebieten zwar gut sein mögen, aber als KI, die auf jede beliebige Situation reagieren können sollte, ungeeignet ist, da man Mengen von Regeln bräuchte, die vielleicht sogar unendlich sind, um auf jede beliebige Situation reagieren zu können. Da sie im Normalfall auch keine Regeln selbst hinzufügen, lernen sie nicht einmal von selbst. Sie bilden also sehr limitierte Intelligenzen, selbst wenn sie noch so kompliziert gemacht sind.

Theorembeweiser ermitteln mithilfe von Logik aus Axiomen sämtliche möglichen Folgerungen mittels Brute Force, sind also wieder eine simple KI, welche noch dazu für größere Berechnungen ineffizient sind. Sie sind ebenfalls äußerst limitiert.

Neuronale Netze haben theoretisch das gleiche Potential wie ein Gehirn, wenn man genügend starke Hardware hat und die KI viel lernen lassen kann. Letztlich scheitert es hier aber an der Hardware, die noch lange nicht auf einem Level ist, wo man eine solche KI mit einem Menschen vergleichen kann.

Viele der restlichen Kapitel hatten entweder mit Anwendungen von diesen KI-Arten, wie zum Beispiel Sprachverarbeitung, oder auch mehr mit der Wahrnehmung zu tun. Zum Beispiel Objekterkennung ist für eine gute Wahrnehmung einer KI unabdingbar. Auch wenn die Sensoren noch so präzise sind, muss eine gute KI diese auch interpretieren können. Anhand von dieser Objekterkennung, welche aus den optischen Sensoren Daten extrahiert, müssen diese Daten klassifiziert werden, zum Beispiel anhand von Support Vector Machines.

15.4 Fazit

Das gesamte Semester haben wir uns mit verschiedensten Arten von Techniken auseinandergesetzt, um künstliche Intelligenzen zu erschaffen. Dabei sind wir von den Regelbasierten Expertensystemen oder Theorem-Beweiser bis auf die vom Menschen relativ unabhängig arbeitenden neuronalen Netze gekommen. Aber was all diese Systeme gemeinsam haben ist, dass mit ihrer Hilfe keine perfekte KI erstellt werden kann, da die Grenzen zumindest aktuell noch zu hoch sind.

Bei Regelbasierten Expertensystemen zum Beispiel brauchen wir mehr Regeln, als jemals geschrieben werden könnten. Bei den neuronalen Netzen ist die Grenze die zur Verfügung stehende Hardware, welche nicht einmal annähernd an die Komplexität einer menschlichen Intelligenz heran kommt. Das menschliche Gehirn ist einfach wesentlich stärker beim Lernen auf dem kleinen Raum als es ein Rechner bisher sein kann.

Aber in einzelnen Bereichen sind Rechner schon jetzt wesentlich stärker, allein schon durch ihr Tempo, wie zum Beispiel Deep Blue oder AlphaGo.

Das Bewusstsein willentlich zu simulieren ist ohne das Wissen, was unser Bewusstsein genau ist, völlig unmöglich. Um eine KI, die ihre Aufgabe richtig erfüllt, zu schaffen, ist dieses allerdings auch nicht unbedingt nötig und vermutlich auch gar nicht wünschenswert. Eine KI mit Bewusstsein wird ihre Aufgabe vermutlich nicht unbedingt besser machen, als eine KI ohne. Sie könnte sogar Nachteile haben. Einer KI mit Bewusstsein müsste wohl dieselben Rechte haben wie ein Mensch, ist sich logischerweise auch ihrer Überlegenheit bewusst, wird also vermutlich schwerer zu kontrollieren sein. Die Spekulationen, was so eine KI tun würde, übernehmen genügend Filme.

15.5 Quellen

- https://de.wikipedia.org/wiki/K%C3%BCnstliche_Intelligenz
- https://de.wikipedia.org/wiki/Intelligenz
- https://de.wikipedia.org/wiki/Bewusstsein
- https://de.wikipedia.org/wiki/Wahrnehmung
- https://de.wikipedia.org/wiki/Sensor
- https://en.wikipedia.org/wiki/Computing_Machinery_and_Intelligence
- https://de.wikipedia.org/wiki/AlphaGo
- https://de.wikipedia.org/wiki/Deep_Blue
- https://de.wikipedia.org/wiki/Turing-Test
- https://de.wikipedia.org/wiki/Chinesisches_Zimmer

16. Grenzen künstlicher Intelligenz

16.1 Motivation

"The real risk with AI isn't malice but competence. A super-intelligent AI will be extremely good at accomplishing its goals, and if those goals aren't aligned with ours, we're in trouble." (Stephen Hawking, 2015)

Der Begriff der künstlichen Intelligenz ist ein ebenso aktueller wie auch umstrittener Begriff, an dessen Definition sich schon seit Längerem die Geister scheiden. Sie gilt als eine der zukunftsträchtigsten Forschungsthemen unserer Zeit. So werden Methoden der künstlichen Intelligenz bereits erfolgreich in der Mustererkennung oder Computerspielentwicklung eingesetzt und stetig weiterentwickelt, um nur wenige Beispiele zu nennen. Oft stellt sich dabei aber die Frage, wie weit die Kompetenzen einer KI gehen können bzw. wie weit sie aus ethischer Sicht gehen sollten. Denn neben den unstrittigen Vorteilen gibt es auch Vorbehalte und Ängste, wie das obige Zitat von Stephen Hawking deutlich macht. Dieser spricht die Gefahren an, die entstehen könnten, wenn unsere Ziele nicht mit denen einer hoch entwickelten KI konform laufen. Mit dieser Meinung steht Hawking nicht alleine da, denn mit Stuart J.Russell, Peter Norvig und Elon Musk unterschrieben im Jahr 2014 gleich mehrere Experten des Gebiets zusammen mit Hawking den „Open Letter on Artificial Intelligence", in welchem vor den Gefahren einer unkontrollierbaren KI gewarnt und zu einer verantwortungsvollen Forschung in diesem Bereich aufgerufen wird. Sind diese Bedenken berechtigt? Wie gefährlich bzw. wie mächtig kann eine KI überhaupt sein? Inwiefern ähnelt sie der menschlichen Intelligenz und welche Stärken und Schwächen weist sie im Vergleich auf? Müssen wir uns vor einer Weiterentwicklung der KI fürchten, wie es manche Science-Fiction-Filme suggerieren oder liegt es an uns, sie zu kontrollieren und ihren Fähigkeiten Grenzen zu setzen? Wieweit sollte man die Forschung in diesem Bereich aus ethischer Sicht vorantreiben? Die folgende Ausarbeitung beschäftigt sich mit all diesen Fragen und versucht, einen Überblick über die Risiken und Grenzen künstlicher Intelligenz zu geben.

16.2 Grenzen der Logik

Methoden der künstlichen Intelligenz bedienen sich oft der Methoden der Logik, so wie es teilweise bei Bayesschen Netzen oder umfassender bei der automatischen Beweisführung geschieht. Daher werden im Folgenden die Grenzen der Logik als Grundlage der künstlichen Intelligenz betrachtet. Nennenswert ist in diesem Zusammenhang das Suchraumproblem. Bei der Beweissuche gibt es für die Anwendung von Inferenzregeln in jedem Teilschritt unzählige bzw. eventuell sogar unendlich viele Möglichkeiten, wodurch der Suchraum sehr schnell wächst. Die Anwendung all dieser Möglichkeiten zur Beweisfindung ist für einen Menschen unzumutbar, während ein automatischer Beweiser tausende Inferenzen pro Sekunde durchführen kann. Ein Mensch schafft u. U. eine Inferenz pro Sekunde. Menschen inferieren also langsamer. Überraschenderweise kann ein Mathematiker dennoch schwierigere Probleme schneller lösen als ein automatischer Beweiser. Menschen können nämlich aufgrund ihrer Intuition und Anschauung mehrere,

einfache Inferenzen des Beweisers in nur einem Schritt durchführen oder bereits bewiesene Lemmata benutzen, um sich Beweisschritte zu sparen, während ein maschineller Beweiser in jedem Schritt sehr viele Möglichkeiten ausprobiert, obwohl nur wenige zielführend sind. Mittlerweile imitieren automatische Beweiser dieses menschliche Metawissen, sind aber bei Weitem nicht so effektiv wie ein erfahrener Mathematiker. Unter anderem ist dies darauf zurückzuführen, dass Menschen bei komplexen Problemen zumeist selbst nicht in der Lage sind, ihre Intuition zu formalisieren. Nicht formalisierbares Wissen kann in der Regel auch nicht programmiert werden, was den Möglichkeiten eines maschinellen Beweisers klare Grenzen setzt.

Eine andere Schwäche der Logik ist ihre Monotonie. Das bedeutet, dass mit der Erweiterung der Formelmenge zwar neue Aussagen bewiesen werden können, alle zuvor ableitbaren Formeln aber nach wie vor bewiesen werden können, sodass die Menge der ableitbaren Aussagen monoton wächst. Dieser Umstand kann allerdings zu Widersprüchen führen. Am Beispiel des „fliegenden Pinguins“ kann diese Problematik ganz einfach illustriert werden: Angenommen man weiß, dass Tweety ein Pinguin ist, Pinguine Vögel sind und Vögel fliegen. Daraus lässt sich per Resolutionsbeweis ableiten, dass Tweety fliegen kann. Da Pinguine aber nicht fliegen können, fügt man eine Regel hinzu, die besagt, dass Pinguine nicht fliegen. Daraus lässt sich ableiten, dass Tweety nicht fliegen kann. Es lässt sich aus dieser Wissensbasis aber nach wie vor auch ableiten, dass Tweety fliegen kann, da die neue Regel die bestehenden Regeln nicht ersetzt, sondern erweitert. Wenn man nun die Wissenbasis um die prädikatenlogische Formeln „Abraxas ist ein Rabe“ und „Raben sind Vögel“ ergänzt, kann man nicht mit Sicherheit darauf schließen, dass Abraxas fliegen kann, da keine Aussage darüber gemacht wird, ob Abraxas ein Pinguin ist oder nicht. Um das Problem zu lösen, müsste die Tatsache „Abraxas ist kein Pinguin“ hinzugefügt werden, während dies für einen Menschen einen selbstverständlichen Fakt darstellt. Wie man an diesem Beispiel sehen kann, muss für jedes Objekt in der Wissensbasis nicht nur angegeben werden, welche Merkmale es besitzt, sondern auch welche es nicht besitzt. Für ein komplexeres Problem impliziert dies ein explosionsartiges Wachstum des Programmieraufwands, wenn die Wissensbasis erweitert oder verändert wird.

Weiterhin ist festzustellen, dass mit der Prädikatenlogik eine Reihe von Aussagen nicht formuliert werden kann, da z. B. Quantoren für Funktionen nicht eingesetzt werden dürfen. Logiken höherer Stufen bieten diese gewünschten Erweiterungen, allerdings hat Gödel gezeigt, dass schon eine minimale Erweiterung der Prädikatenlogik zum Verlust der Vollständigkeit führt, d. h. es gibt wahre Formeln, die nicht beweisbar sind. Folglich sind die Prädikatenlogik wie auch Logiken höherer Stufen noch nicht mächtig genug. Außerdem ist es fraglich, ob etwas derart Komplexes wie das menschliche Denken bzw. Verhalten durch eine Menge von Formeln beschrieben werden kann. In der Regel ist es nicht möglich, die menschliche Intelligenz umfassend in Formeln auszudrücken, was auch als Qualifizierungsproblem bezeichnet wird. Das Problem liegt darin, dass Menschen nicht in Formeln denken oder anhand von Formeln Entscheidungen treffen, sondern implizit den gesamtheitlichen Kontext betrachten und davon abhängig handeln. Sie haben einen Sinn dafür, wie in einer Situation zu verfahren ist bzw. was man für ein Verhalten erwarten würde, so wie es bspw. der Fall ist, wenn man ein Geschenk erhält und dann den Drang empfindet, selbst etwas an den Schenker zu verschenken. Insgesamt kann man also festhalten, dass die Logik als Grundlage einer künstlichen Intelligenz dem menschlichen Verstand in vielerlei Aspekten trotz Imitationsversuchen noch weit unterlegen ist.

16.3 Experimente

Sehr wichtig für das Thema dieser Arbeit sind Experimente und Tests, die zum Ziel haben, eine intelligente Maschine als solche einzustufen, denn anhand dessen kann festgestellt werden, wie nah eine KI der menschlichen Intelligenz ist und wo ihre Grenzen liegen. Der Turing-Test ist der wohl bekannteste Versuch. Dabei sitzt eine Person A in einem Raum an zwei Computern, über die sie Fragen eintippen kann, wobei ein Computer mit der vermeintlichen künstlichen Intelligenz und der andere Computer mit einer weiteren Person B verbunden ist. Nach fünf Minuten muss A entscheiden, mit welchem Computer die KI verbunden ist. Wenn A in 30% aller Fälle falsch entscheidet, besteht die Maschine den Test und weist somit intelligentes Verhalten auf. Mittlerweile gibt es eine Reihe von sogenannten Chatterbots, die zu Beginn einer Konversation überraschend oft passgenaue Antworten liefern. Bekannt und besonders effektiv in der Täuschung sind z. B. Eliza oder Mgonz, aber auch CyberLover. Letzterer geht bereits soweit, dass er seinem Gesprächspartner persönliche Daten zu entlocken versucht, um seine Identität festzustellen und diese zu stehlen. Diese Programme offenbaren aber im Laufe des Gesprächs relativ schnell ihre begrenzten Antwortmöglichkeiten, da sie entweder nicht lernfähig sind oder nur einen auf bestimmte Bereiche beschränkten Wissensstand aufweisen. Dies erkennt man sehr gut am folgenden Beispiel einer in das

Deutsche übersetzten Konversation mit Eliza:

```
ELIZA: Wie geht es Ihnen.
Bitte schildern Sie Ihr Problem.
Ich verbringe zu viel Zeit am Computer.
ELIZA: Warum erwähnen Sie Computer?
Du bist einer.
ELIZA: Macht es Ihnen Spaß zu glauben, ich sei einer?
Ja.
ELIZA: Sie scheinen sehr optimistisch zu sein.
Bin ich.
5
ELIZA: Wie lange sind Sie schon?
31 Jahre.
ELIZA: Ich bin mir nicht sicher, ob ich sie verstehe.
Willkommen im Club
ELIZA: Bitte fahren Sie fort.
Was du sagst, ist auch nicht sehr sinnvoll.
ELIZA: Wir reden über Sie, nicht über mich.
```

Man sieht an diesem Beispiel relativ klar, dass Eliza zunehmend verwirrende Antworten gibt, je einfältiger die Antworten des Kommunikationspartners von Eliza ausfallen. Der noch blamablere Fall von TayTweets verdeutlicht um einiges besser die fehlende Intelligenz der derzeitigen Chatterbots. Turing prognostizierte seinerzeit, dass ein Computer im Jahr 2000 den Test besteht. Diese Vermutung hat sich noch nicht bewahrheitet, da Computer heute einiges leisten können, die Führung einer offenen Konversation aber derzeit nicht dazugehört. Zurückzuführen ist dies darauf, dass der Mensch ein immenses Hintergrundwissen besitzt, wodurch die Menge an möglichen Fragen eine intelligente Maschine schier überfordert. Es gab zwar einen Fall einer russischen Software, die den Test bestanden hat. Allerdings wurden Mängel am Versuchsaufbau und an den Fragen der Jury festgestellt. Was den Turing-Test aber so interessant macht, ist die Frage, ob damit ein maschinelles Denken nachgewiesen werden kann. Es gibt Meinungen, die behaupten, dass der Turing-Test höchstens die Simulation des Denkens und nicht das Denken selbst nachweisen kann. Es wird an dem Test kritisiert, dass der Computer die Ein- und Ausgaben nicht wirklich versteht und lediglich richtige Antworten gibt, weshalb man nicht von einer Darstellung von Verstand sprechen kann. Versuche, den Test zu bestehen, waren darüber hinaus eher darauf ausgerichtet, den Kommunikationspartner zu überlisten und nicht darauf, eine zu einer Konversation fähige, echte Intelligenz zu entwickeln.

Ebenso wie die Kritik am Turing-Test setzt das Experiment „Chinese Room Experiment“ von John R. Searle am Verständnis an. Dabei sitzt eine Versuchsperson, die nur Englisch versteht, in einem Raum mit einer kleinen Öffnung. Ausgestattet ist sie dabei mit einem in englischer Sprache verfassten Regelbuch mit Anweisungen sowie mit mehreren teils leeren, teils mit für die Versuchsperson nicht entzifferbaren Aufschriften versehenen Papierstapeln. Über die Öffnung erhält er von außen Papierstreifen mit unbekannten Symbolen. Die Versuchsperson sucht diese Zeichen im Regelbuch, führt die einschlägigen Anweisungen durch und muss dafür gegebenenfalls die Papierstapel nach Symbolen durchsuchen oder Symbole auf leere Papierstreifen schreiben. Am Ende jeder Anweisung muss ein Papierstreifen beschrieben und über die Öffnung nach außen durchgereicht werden. Außerhalb des Raums werden die Papierstreifen von einem chinesischen Muttersprachler beschrieben und über die kleine Öffnung in den Raum gegeben. Man erhält dann nach einer gewissen Zeit einen Papierstreifen aus dem Inneren des Raums, wobei dieses mit einer passgenauen, in chinesischer Sprache verfassten Antwort zu dem zuvor hineingegebenem Papierstreifen beschriftet ist. Der chinesische Muttersprachler schließt daraus, dass sich ein chinesisch sprechender Mensch im Raum aufhält. In diesem Fall verstehen weder die Versuchsperson noch das Regelbuch oder die Papierstapel Chinesisch, erzeugen zusammen aber über Vergleiche der Schriftzeichen Antworten, wie es der Turing-Test erfordert.

Searle wollte damit zeigen, dass die Ausführung des richtigen Programms ohne die Erzeugung eines Verständnisses auskommt, weshalb auch eine KI seine Ein- und Ausgaben nicht verstehen muss, um nach Turing als intelligent zu gelten. So wollte er als Kritik am Turing-Test verdeutlichen, dass man nie feststellen kann, ob eine Maschine Verstand bzw. Verständnis besitzt. Ein neueres Unterfangen ist der Lovelace-2.0-Test von Mark O. Riedl, welcher als Kritik am Turing-Test entstand. Eine Konversation zu führen, verkörpere keine wahre Intelligenz.

Diese bestünde stattdessen in der Kreativität. Diesem vorangegangen war der einfache Lovelace-Test, welcher die kreative Schöpfung von etwas Neuem als einziges Indiz für Intelligenz sieht. Dieser Test wurde abgeändert, da er impliziert, dass das Werk der KI nur als kreativ gewertet werden kann, wenn der Entwickler der KI die Entstehung des Werks nicht versteht. Da es deshalb nahezu unmöglich gewesen wäre, diesen Test zu bestehen, erfand Riedl den Lovelace-2.0-Test. Hierbei wird die KI aufgefordert, zu einem vorgegebenen Thema eine Geschichte zu erzählen. Wenn die KI dies schafft, wird das Thema immer weiter eingegrenzt und die KI immer wieder aufgefordert, eine Geschichte zu erzählen, bis die KI es nicht mehr schafft. Damit kann man das Ausmaß der Intelligenz einer Maschine einordnen. Die Programme von Riedl selbst haben nur die ersten zwei Stufen des Tests bestanden, was für ein niedriges Maß an Intelligenz spricht, da ein Autor oder Dichter erst an weitaus höheren Stufen scheitern würde. Auch dieser Test zeigt also die Begrenztheit der aktuellen KI-Programme.

16.4 Unfähigkeit

Die herrschende Meinung in der Gesellschaft zum Thema „künstliche Intelligenz“ besagt, dass Maschinen trotz aller Fortschritte zu bestimmten Dingen nicht fähig sind und in absehbarer Zeit auch nicht sein werden. Turing listete in einem ganzen Katalog solche Dinge auf, zu denen beispielsweise „etwas wirklich Neues tun“, „initiativ sein“, „Fehler machen“, „verlieben“ oder „Erdbeereis mit Sahne mögen“ zählen. Zwar können sich Computer nicht verlieben, wie es Filme/Romane in Bezug auf die Liebe zwischen Mensch und Maschine seit jeher prophezeien. Allerdings ist es geradezu erstaunlich, wie viel Computer von dem Katalog von Turing bereits können bzw. besser können als Menschen. Ein Beispiel ist, dass Computer ebenso wie der Mensch Fehler machen, sodass man diesen Punkt aus dem oben genannten Katalog streichen kann. Es gibt sogar Fälle, in denen ein Computer ungeduldig werden kann. Es bleibt alles in allem dennoch dabei, dass Maschinen in bestimmten Dingen an ihre Grenzen stoßen. Menschen haben beispielsweise die Fähigkeit, einen unbekannten Raum zu betreten, sofort die Szene zu erfassen, darauf aufbauend Aktionen zu planen und Entscheidungen zu treffen. Zurückzuführen ist dies auf das Adaptivitätsmerkmal der Menschen, sie passen sich an verschiedenste Umweltbedingungen an und ändern ihr Verhalten, indem sie lernen. Diese Anpassungsfähigkeit und das ausgeprägte Lernverhalten fehlt den Maschinen, wobei man sich dem letzteren zunehmend durch die Nutzung von Methoden des Machine Learning annähert.

Ähnlich verhält es sich mit der Sprache. So spricht Noam Chomsky den Maschinen die Sprachkompetenz ab, da den Menschen nicht erlernbare sprachliche Fähigkeiten angeboren seien, die es erlauben, noch nie gehörte, komplexe Sätze hervorzubringen. Den Menschen sei eine universelle Grammatik in die Wiege gelegt, anhand derer die Nutzung und das Erlernen von Sprache möglich sei. Maschinen sind darüber hinaus nicht in der Lage, Dinge aus eigenem Antrieb hervorzubringen und spontan zu handeln. Vielmehr führen sie Anweisungen auf gegebenen Daten aus, wenn eine bestimmte Bedingung erfüllt ist, was nicht für die Existenz einer Intelligenz spricht. Menschen haben weiterhin ein Bewusstsein, d. h. sie können über sich selbst nachdenken oder darüber, dass sie über sich selbst nachdenken können usw.

Dies ist der ausschlaggebende Unterschied zwischen dem Menschen und einer künstlichen Intelligenz. Es reicht also bspw. nicht, ein Gedicht zu schreiben, sondern es muss einem bewusst sein, dass man das Gedicht geschrieben hat. Einem KI-Programm fehlt es zusätzlich an Emotionen, allen voran an Empathie, die über die bloße Einschätzung des Gefühlszustands des Gegenübers hinausgeht und in der Regel in einer emotionalen Reaktion besteht. Zuletzt ist festzustellen, dass bisher kein Programm die Leistung des menschlichen Gehirns bei einem derart niedrigen Energieverbrauch auch nur annähernd erreicht hat.

Wie man sieht, gibt es noch eine Reihe an Dingen, die intelligente Maschinen nicht beherrschen, aber auch Fähigkeiten, die sie im Laufe der Zeit dazugewonnen haben. Unfähigkeit von Computern ist also ein generisches Problem. Was eine künstliche Intelligenz heute nicht kann, macht sie morgen vielleicht um ein Vielfaches effizienter als ein Mensch.

16.5 Ethische Aspekte und Risiken

Eine weitere Frage nach den Grenzen der künstlichen Intelligenz ist die, ob man künstliche Intelligenzen vor dem Hintergrund moralischer Aspekte weiterentwickeln sollte. Die Forschung steht im Falle einer Weiterentwicklung vor dem Problem der unvorhersehbaren, möglicherweise negativen Konsequenzen. Die Erfindung der Kernspaltung, die unter anderem zur Katastrophe von Tschernobyl führte, oder des Verbrennungsmotors, welcher zur globalen Erwärmung und Luftverschmutzung maßgeblich beiträgt, wird dabei als negatives Vergleichsbeispiel gesehen. Eine Befürchtung ist der Verlust von Arbeitsplätzen bei einer fortschreitenden Automatisierung. Beispielsweise werden

in den USA manche Kreditkartenanwendungen und die Erkennung von Kreditkarten-Betrug von Programmen aus der künstlichen Intelligenz ausgeführt. Es liegt also die Vermutung nahe, dass in diesem konkreten Fall Arbeitsplätze verloren gegangen sind. Dem ist aber in der Regel nicht so, da diese Arbeitsplätze gar nicht existieren würden, weil sie für das Unternehmen nicht wirtschaftlich wären. So hat eine Bank in der Regel keinen Mitarbeiter, der nur Kreditkartenbetrug aufdeckt. Es geht also auch kein Arbeitsplatz verloren, wenn eine künstliche Intelligenz diese Aufgabe übernimmt. Dieses Beispiel kann man auf die meisten Fälle übertragen, in denen eine KI bestimmte Leistungen bietet. Aber angenommen, die Arbeitslosigkeit würde steigen. Dann würde das Bruttosozialprodukt eines Landes dennoch nicht sinken bzw. sogar steigen, da die Wertschöpfung durch den Einsatz von effizienteren Maschinen steigen würde. Die arbeitslos gewordenen Menschen würden dann womöglich durch die Verteilung des neu hinzugewonnenen Reichtums bedient werden, weshalb sich die Arbeitslosigkeit nicht unbedingt negativ auswirken müsste. Falls es aber gelingt, hochintelligente Maschinen zu entwickeln, die alle Aufgaben der Menschen im Beruf oder im Haushalt übernehmen können, wäre es durchaus denkbar, dass es überhaupt keine von Menschen zu verrichtende Arbeit mehr gibt, die Menschen in hohem Maß von den Maschinen abhängig sind und sie daher keinerlei Druck verspüren bzw. keinerlei Anstrengungen für ihre Existenz unternehmen, sodass es zu einem gesellschaftlichen Kollaps kommen kann.

Als eine weitere mögliche Gefahr wird in der Literatur oft der Umstand genannt, dass eine sehr weit entwickelte KI dem Menschen sein Selbstverständnis als einzigartiges Individuum rauben kann und ihm somit das Gefühl eines imitierbaren, lediglich in einem System funktionierenden Automaten vermittelt. Diese Gefahr ist allerdings zu relativieren, denn in der Geschichte gibt es viele ähnliche Beispiele, die dem Menschen ihr Einmaligkeitsgefühl streitig machten. So stellte Darwin mit seiner Evolutionstheorie den Menschen auf eine Stufe mit den Tieren, was seinerzeit umstritten war, jedoch bis dato nicht zu einem Verlust jenes Selbstverständnisses geführt hat. Zudem macht es den Menschen wiederum zu etwas Besonderem, wenn er in der Lage ist, eine der menschlichen Intelligenz ähnliche Intelligenz künstlich zu erschaffen.

Darüber hinaus könnte die zunehmende Automatisierung durch Programme der künstlichen Intelligenz bewirken, dass man sich seiner Verantwortung entzieht. Haftet beispielsweise der Arzt, wenn eine KI eine falsche medizinische Diagnose liefert? Wem sind Schulden zuzuordnen, wenn eine KI, die Geldgeschäfte für ein Unternehmen erledigt, diese ohne weitere Nachfrage aufgenommen hat? Diese Frage wird umso relevanter werden, je stärker die Automatisierung voranschreitet.

Auch begründet ist die Sorge nach dem Einsatz von künstlicher Intelligenz zu unerwünschten Zwecken. Mit diesem Problem ist man derzeit im Militär konfrontiert, bspw. nutzten die USA im Irakkrieg über 5000 unbemannte, autonome Luftfahrzeuge, mit denen Menschen umgebracht werden konnten, ohne dass je ein Mensch sein Leben aufs Spiel setzen musste. Zudem stellt sich für den Menschen dann gar nicht die Frage nach der moralischen Verwerflichkeit seines Handelns, da er außer Reichweite ist und nicht merkt, was sein Handeln für Konsequenzen hat.

Ein anderes Problemgebiet ist das Potenzial der künstlichen Intelligenz über die Spracherkennung die Privatsphäre der Menschen zu verletzen. Ihr ist zu verdanken, dass Staaten mittlerweile regelmäßig Telefongespräche abhören. Eine KI kann also leicht opportunistisch genutzt werden, wenn sie in die falschen Hände gerät. Einige Meinungen gehen sogar so weit, dass sie einer KI die Fähigkeit einräumen, das Ende der Menschheit herbeizuführen. Es ist durchaus umstritten, ob diese Einschätzung realistisch ist oder nur in der Fiktion eintreten kann. Die falsche Zustandsabschätzung einer KI kann zu erheblichen Schäden führen. Beispielsweise könnte ein autonomes Raketenabwehrsystem eine Bewegung im Luftraum fälschlicherweise für einen Angriff halten und entsprechend reagieren, sodass es zu einem Verlust von Menschenleben kommen kann. Allerdings besteht dieses Risiko auch stets, wenn ein Mensch das System bedienen würde, sodass das Problem nicht nur auf eine KI zurückzuführen ist.

Weiterhin kann es sein, dass die Nutzenfunktion einer KI, nach der sie letztlich handelt, fehlspezifiziert ist. Sie könnte bspw. derart programmiert werden, dass sie das menschliche Leid zu minimieren versucht. Der Mensch neigt nun aber dazu, sich immer wieder beabsichtigt oder unbeabsichtigt neue Situationen des Leidens zu schaffen. Die vermeintliche Konsequenz wäre mit der oben genannten Nutzenfunktion als Handlungsrahmen die Ausrottung der Menschen. Dies suggerieren bisweilen auch viele Science-Fiction-Filme. Was hier aber implizit angenommen wird, ist zum einen die Tatsache, dass die KI die Nutzenfunktion zu wörtlich nimmt, und zum anderen die Aggressivität, die eine Maschine nicht unbedingt aufweisen muss, dem Menschen aber im Zuge der natürlichen Selektion in die Wiege gelegt ist. Außerdem sollte erwartet werden können, dass eine Maschine, die eine derartige Intelligenz besitzt, dass sie den Menschen als bis dato intelligentestes Lebewesen auszulöschen in der Lage ist, auch die Intelligenz besitzt, die Absicht der Nutzenfunktion richtig zu deuten.

Gefährlicher ist dagegen eine statische Nutzenfunktion, da sie nur die Werteverhältnisse der Entstehungszeit einer KI umfassen kann. So würde eine KI, die im 18. Jahrhundert entwickelt wurde, heute versuchen, das Wahlrecht für Frauen abzuschaffen und damit aktuelle Moralvorstellungen untergraben, anstatt diese zu fördern. Sinnvoller erscheint dagegen, dass die KI ihre Nutzenfunktion selbst aktualisiert. Hier bestünde aber die Gefahr, dass die Maschine tiefste Abgründe des menschlichen Denkens erkennt und entsprechend handelt. Beispielsweise könnte sie erkennen, dass Menschen keine Scheu haben, lästige Insekten zu töten, weil letztere als primitiv angesehen werden. Sie würde daher womöglich das Töten von Menschen für moralisch unverwerflich halten, da diese wiederum im Vergleich zu ihr primitiv sind. Nimmt man nun aber an, dass eine hoch entwickelte KI nicht aggressiv ist, so gibt es auch dann problematische Punkte, die zu beachten sind. Wenn die Intelligenz einer Maschine die des Menschen übersteigt, so wird insbesondere sie in der Lage sein, eine noch weiter entwickelte Intelligenz hervorzubringen und letztere wird genauso verfahren, sodass es irgendwann zu einer sogenannten Intelligenzexplosion, auch als technologische Singularität bezeichnet, kommt und der Mensch im Vergleich zu den Maschinen zu einem primitiven Wesen verkommt. Dadurch würde der Mensch zwar nicht unbedingt im physischen Sinne bedroht werden, allerdings würde er die Entwicklung vom Herrscher über die Erde zum Beherrschten vollbringen, so wie einst mit der Evolution des Menschen die Ära der Tiere endete. KI könnten dann die Entwicklung der menschlichen Zivilisation lenken.

Dieses Szenario wird allerdings nicht ausschließlich negativ aufgenommen. Mit dem Transhumanismus gibt es eine Bewegung, die sich das Zusammenleben und die Zusammenarbeit von Mensch und Maschine herbeiwünscht, wobei einige sogar dem Ersatz des Menschen durch die Maschine entgegenfiebern. Ray Kurzweil ist der bekannteste Anhänger dieser Meinung. Er sieht hierin die Möglichkeit für den Menschen, über seine eigene Sterblichkeit zu entscheiden, indem die Grenzen des menschlichen Körpers gesprengt werden. Die KI könnten als nächste Evolutionsstufe des Menschen gesehen werden. Man solle aber die Ur-KI so programmieren, dass sie den Menschen gut behandelt, denn dann würden es ihr die von ihr entwickelten, intelligenteren Programme gleichtun, sodass keinerlei Gefahren für die Menschheit bestehen würden.

16.6 Grenzen der Hardware

Der Folgende Abschnitt befasst sich nur mit den Grenzen des Prozessors. Aktuell wird die Grenze der Leistungsfähigkeit einer CPU durch die Anzahl der Transistoren auf den Prozessoren, sowie die Anzahl der Prozessoren festgelegt. Das Wachstum der Transistoranzahl auf einem Chip konnte in den letzten knapp 100 Jahren durch das Mooresche Gesetz beschrieben werden.

Das Moorsche Gesetz besagt, dass sich die Komplexität integrierter Schaltkreise mit minimalen Komponentenkosten regelmäßig verdoppelt, also verdoppelt sich die Anzahl der Transistoren auf einem Chip alle 12 bis 18 Monate (je nach Quelle) auf einer CPU. Da die Größe eines Chips begrenzt ist muss folglich die Größe der Transistoren abnehmen.

16.6.1 Fertigung einer CPU

Bei der Abnahme der Transistorgröße gibt es allerdings einige Limitierungen wie z.B. wirtschaftliche Effizienz, da es immer aufwendiger wird in zunehmend kleineren Dimensionen Transistoren auf dem Chip aufzubringen, und das Auftreten von Quantenmechanischen Effekten, bei denen Elektronen, welche zum Ladungstransport innerhalb eines Transistors benutzt werden, Aufenthaltswahrscheinlichkeiten außerhalb des Transistors besitzen und folglich den Zustand des Transistors verfälschen. Diese Quantenmechanischen Effekte treten ab einer Fertigungsgröße von 5nm auf, welche auch als die „Magische Grenze“ bekannt ist.

Aktuell liegen die Fertigungsgrößen bei etwa 14nm bei der Skylake Architektur von Intel, diese CPUs werden mit einem Verfahren namens EUV-Lithografie hergestellt. Die EUV-Lithografie benutzt weiche Röntgenstrahlung, welche frei wird wenn man auf ein bestimmtes Gasgemisch mit einem Laser strahlt, um die Transistoren auf der Oberfläche des Chips zu befestigen.

16.6.2 Alternativen für die Zukunft

Folglich gibt es eine Leistungsgrenze für Prozessoren, daher wurden für die Zukunft Alternativen gesucht. Aktuell sind die vielversprechendsten Alternativen, Quantencomputer und DNA-Computer, welche bei bestimmten Problemstrukturen bis zu 100 Mio mal so schnell sind wie die heutige Durchschnittscomputer. Dennoch haben auch Quantencomputer schwächen wie Beispielsweise die sehr aufwändige Kühlung des Q-Bits auf 3 Grad Kelvin. Quantencomputer bergen auch ein gewisses Risiko, da sie Aufgaben wie die Primfaktorzerlegung, auf welcher fast

alle Modernen Verschlüsselungen basieren, binnen wenigen Sekunden durchführen können. DNA-Computer haben das Problem, dass die Ergebnisse welche sie ausgeben noch von einem Computer interpretiert werden müssen, was extrem aufwändig ist.

16.7 Fazit

Insgesamt kann man festhalten, dass Errungenschaften der KI-Forschung uns derzeit diverse Erleichterungen bieten und sich dieser Trend in Zukunft sogar verstärken wird. Nichtsdestotrotz sind die Möglichkeiten von Programmen der KI begrenzt. So sind Computer möglicherweise effizienter als der Mensch, aber nicht unbedingt auch effektiver. Vom mathematisch-logischen Standpunkt aus betrachtet, können Computer viele Berechnungen und Inferenzen sehr schnell durchführen. Wenn aber ein komplexeres Problem vorliegt, kann der Mensch durch seine Intuition bzw. sein Metawissen im Vergleich zum Computer schnellere Schlüsse ziehen. Auch generell gibt es bestimmte Fähigkeiten, die eine KI zumindest noch nicht beherrscht. So besitzen Maschinen kein Bewusstsein und sind nicht fähig, Emotionen zu empfinden. Diverse Tests und Experimente verdeutlichen, dass maschinelles Denken und Verstehen nicht unbedingt nur aus der Angabe von passenden Antworten in einer Konversation abgeleitet werden kann. Man kann zudem festhalten, dass einige Sorgen bezüglich der Macht einer KI schlicht unberechtigt sind, da bisher keine Fälle von gefährlichen, unkontrollierbaren Ereignissen auftraten. Auch ist die instinktiv angenommene Bösartigkeit und Aggressivität einer intelligenten Maschine kein Pflichtmerkmal, sondern eher eine Eigenschaft der dystopischen Fiktion. Andere Sorgen finden dagegen durchaus Berechtigung. Dazu zählen vor allem Risiken ethischer Natur. Es könnte bspw. zu einem Wandel der gesellschaftlichen Struktur kommen, je mehr intelligente Maschinen uns Aufgaben abnehmen und somit den Standardlebenszyklus Arbeit-Konsum untergraben. Problematisch könnte weiterhin die Spezifikation der Wertevorstellungen und der Nutzenfunktion einer KI sein, damit eine KI im Sinne ihres Schöpfers handelt. Es bleibt abzuwarten, welche der Risiken tatsächlich eintreten werden. Dennoch sollte wie in jedem anderen Bereich Wert auf eine verantwortungsvolle Forschung gelegt werden, insbesondere in den Fällen, in denen eine KI Menschenleben gefährden könnte, wie es bspw. bei autonomen Luftabwehrsystemen der Fall ist. Man sollte sich zum Thema KI nicht allzu sehr von der Fiktion irritieren lassen. Entgegen der Meinungen bekannter Persönlichkeiten wie Elon Musk oder Stephen Hawking ist festzustellen, dass KI-Programme noch nicht in der Lage sind, eine ernsthafte Gefahr für den Menschen darzustellen. Es ist zumindest zu früh, sich Weltuntergangsszenarien auszumalen, da zum derzeitigen Forschungsstand viele der in diesen Szenarien typischerweise auftretenden Fähigkeiten der Maschinen klar und deutlich außerhalb des Machbaren liegen. In absehbarer Zeit scheint ein Ende der Welt durch die Hand einer KI nicht wahrscheinlicher zu sein als durch die Handlungen der Menschen selbst.

16.8 Literatur

- Wolfgang Ertel, Grundkurs Künstliche Intelligenz – Eine praxisorientierte Einführung, 2013 (3.Auflage), Springer-Verlag.
- Stuart Russell & Peter Norvig, Künstliche Intelligenz – Ein moderner Ansatz, 2012 (3.Auflage), Pearson-Verlag.
- Karsten Hartmann, Einführung in die Expertensystem-Technologie, 2015, Hochschulverlag Merseburg.
- Nils J. Nilsson, Die Suche nach künstlicher Intelligenz – Eine Geschichte von Ideen und Erfolgen, 2014, AKA-Verlag.
- `http://www.wired.com/brandlab/2015/10/stephen-hawkings-ama/`
- `https://www.wired.de/collection/featured/ein-neuer-test-soll-nachweisen-wie-intelligent-co`
- `http://www.netzpiloten.de/emotionen-kuenstliche-intelligenz-ki/`
- `http://www.faz.net/aktuell/feuilleton/debatten/warum-die-kuenstliche-intelligenz-gefahren-`
- `https://en.wikipedia.org/wiki/Open_Letter_on_Artificial_Intelligence`

VII Appendix

Literaturverzeichnis

Bücher

[Bra12] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Pearson, 2012 (siehe Seite 129).

[RN12] Stuart Russel und Peter Norvig. *Künstliche Intelligenz. Ein moderner Ansatz*. Pearson, 2012 (siehe Seiten 127, 128).

Wissenschaftliche Artikel

[Fit98] Brandon Fitelson. „Using Mathematica to Understand the Computer Proof of the Robbins Conjecture“. In: *Mathematica in Education and Research* 7.1 (1998), Seiten 17–26 (siehe Seite 26).

[GGC14] Hongbiao Goa, Yuichi Goto und Jingde Cheng. „Research on Automated Theorem Finding: Current State and Future Directions“. In: *Future Information Technology: FutureTech 2014*. Herausgegeben von James J. Park u. a. Band 309. Lecture Notes in Electrical Engineering. Berlin: Springer, 2014, Seiten 105–110 (siehe Seite 32).

[Hal08] Thomas C. Hales. „Formal Proof“. In: *Notices of the AMS* 55.11 (2008), Seiten 1370–1380 (siehe Seiten 25, 26).

Webpages

[UT] University of Cambridge und TU München. *Isabelle Webpage*. URL: `http://isabelle.in.tum.de` (siehe Seite 32).