Grundlagen der KI

1. Einführung

Organisatorisches, KI in Freiburg, Motivation, Geschichte, Herangehensweisen, Beispiele

Wolfram Burgard

Organisatorisches

Vorlesung:

- **Zeit und Ort:**

  Mo 11.15–12.45 Uhr
  101-00-026
  Mi 11.15–12.00 Uhr
  101-00-026.

- **Dozent:**

  Prof. Dr. Wolfram Burgard
  Sprechstunden: n.V.

Übungen:

- Zeit und Ort: Mi 12.15-13.00 Uhr
  101-00-026/036
- **Betreuerin:**

  Dipl.-Inf. Maren Bennewitz
  Sprechstunden: n.V.

- **Stud. Hilfskraft:** Malte Helmert

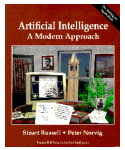
Scheinkriterien:

- Aktive Teilnahme an den Übungen (Vorrechnen)
- Klausur

Benotete Scheine können in die Diplomprüfung eingebracht werden

Vorlesungsunterlagen

Die Vorlesung orientiert sich an:



Artificial Intelligence – A Modern Approach
Stuart Russel - Peter Norvig

In der Bibliothek. Im Buchladen ca. 90 DM.

Kopien der Vorlesungsfolien und weitere Informationen über die WWW-*Homepage* oder direkt:

`http://www.informatik.uni-freiburg.de/~ais/lehre/ss00/ki-vl.html`

(Viele Abbildung sind dem o.g. Buch entnommen. Manche Folien beruhen auf Vorlagen von Prof. Gerhard Lakemeyer, Univ. Aachen. Viele Abschnitte wurden von Prof. Nebel ausgearbeitet. Darüber hinaus wurden Unterlagen von Dr. Jana Köhler verwendet.)

Inhalt der Vorlesung

⇝ stark methodenorientiert

1. Einführung
2. Rationale Agenten
3. Problemlösen durch Suche
4. Informierte Suche
5. Brettspiele
6. Aussagenlogik
7. Erfüllbarkeit und Modellkonstruktion
8. Prädikatenlogik
9. Modellieren mit Logik
10. Handlungsplanung
    (gehalten von Prof. Dr. Bernhard Nebel)
11. Verarbeitung unsicheren Wissens
12. Handeln unter Unsicherheit
13. Maschinelles Lernen
    (gehalten von Prof. Dr. Luc De Raedt)
14. Reinforcement Learning
15. Lernen in Neuronalen Netzen

Was ist „Künstliche Intelligenz“?

- Der Versuch, Computer „intelligenter“ zu machen.
- Die Natur der menschlichen Intelligenz besser zu verstehen.

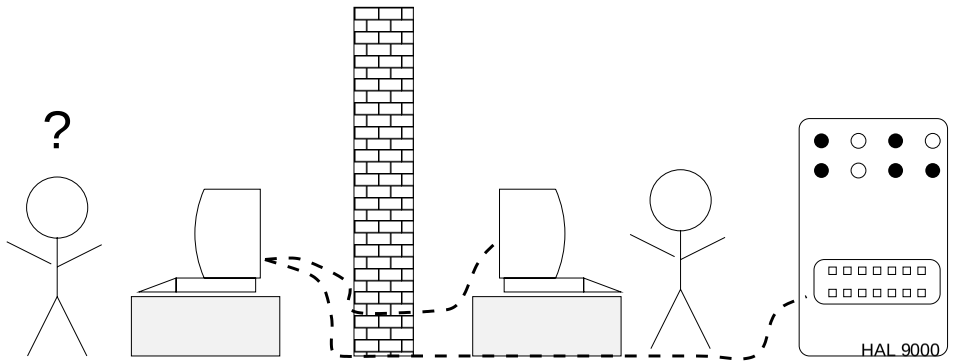
⇝ 4 Herangehensweisen:

- Geht es ums Denken . . .
- . . . oder ums Handeln?
- Am menschlichen Vorbild orientiert (mit all seinen Schwächen) . . .
- . . . oder normativ (wie sollte ein rationales Wesen denken/agieren)?

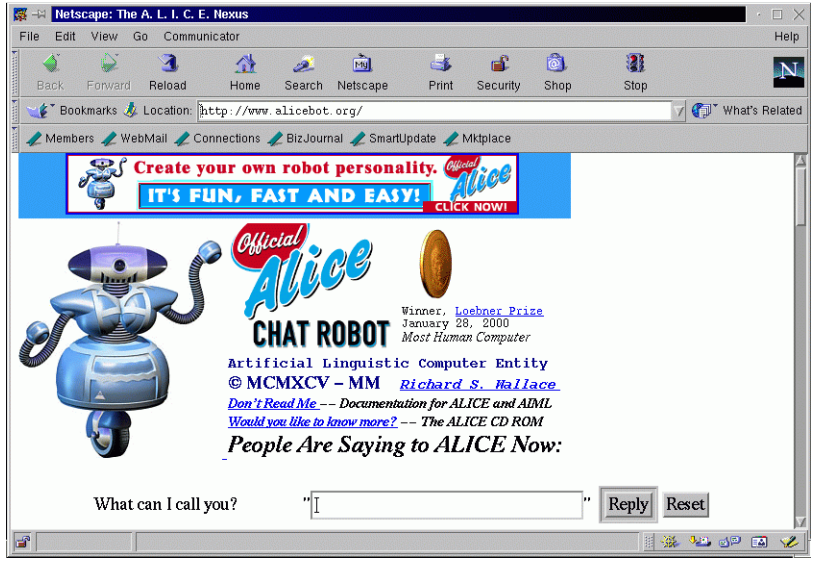
Einige Definitionen . . .

“The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense” (Haugeland, 1985)	“The study of mental faculties through the use of computational models” (Charniak and McDermott, 1985)
“[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . .” (Bellman, 1978)	“The study of the computations that make it possible to perceive, reason, and act” (Winston, 1992)
“The art of creating machines that perform functions that require intelligence when performed by people” (Kurzweil, 1990)	“A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes” (Schalkoff, 1990)
“The study of how to make computers do things at which, at the moment, people are better” (Rich and Knight, 1991)	“The branch of computer science that is concerned with the automation of intelligent behavior” (Luger and Stubblefield, 1993)

Der Turing-Test



Gewinner 1999: Richard Wallace (http://www.alicebot.org)



```
> Wolfram
Nice to meet you Wolfram.
> Do you know Clinton
   He was the second Impeached President of the United States.
> Do you like him
   Sure he has never done anything to me.
> What is an impeachment
   It is a good question. I have to process that one for a while.
   I would do a search for it.

> Who are you
   I ask myself that question, "Who am I?"
> What are you doing
   I am chatting with people on the net.
> What is your favorite topic
   My favorite subject is artificial intelligence.
> What is artificial intelligence
   Artificial intelligence is the branch of engineering and science devoted
   to constructing machines that think. I can ask someone about it.
> Can you do anything else than chatting
   Doing anything else than chatting is not among my strongest traits.
...
```



```
...
> How old am I
   Is this a trick question? You said you were how many years old?
   My guess is that you are really a kid.
> What would you like to know about me
   Tell me about your educational background.
> I am a professor in computer science
   You are a professor in computer science? How do you like your work?
> It is fun teaching courses on artificial intelligence
   I think it is a lot of fun.
> What should I tell the students about you
   Be sure to think before you speak. State your point clearly and slowly and
   gauge the listener's response before going any further.
```


Systeme, die menschlich handeln

Nachbildung menschlichen Verhaltens

→ Prototypisch: Der Turing-Test.

→ . . . in der KI kaum verfolgt,

⇝ . . . aber jährliche Wettbewerbe
(z.B. Loebner Prize:
http://www.loebner.net/Prizef/loebner-prize.html).

Systeme, die menschlich denken

Welche kognitiven Fähigkeiten sind notwendig, um intelligente Leistungen zu erbringen?

- nicht wichtig: Aufgaben korrekt lösen
- wichtig: Aufgaben so lösen, wie ein Mensch sie löst

→ Kognitionswissenschaft und kognitive Psychologie

→ auch wichtig für MMK

- . . . wird nicht in dieser Vorlesung behandelt

Systeme, die rational denken

- Was sind die *Gesetze* des Denkens?
- Wie *sollten* wir denken?

$\rightsquigarrow$ Der logische Ansatz

$\rightarrow$ Probleme:

  - Darstellung informeller Aufgabenbeschreibungen mit Hilfe eines Formalismus
  - Berechenbarkeitsprobleme

$\rightsquigarrow$ Dies sind allerdings Probleme, die bei jeder Formalisierung auftreten.

Systeme, die rational handeln

$\rightsquigarrow$ *Rationale Agenten* (oder rationale Akteure)

- Ein rationaler Agent agiert so, dass er seine gegebenen *Ziele* erreicht unter der Voraussetzung, dass seine Eindrücke von der Welt und Überzeugungen richtig sind
- Rationales Denken ist eine Voraussetzung für rationales Handeln, allerdings keine *notwendige* Voraussetzung.

$\rightarrow$ Was z.B., wenn nicht genügend Information vorliegt, um eine Entscheidung zu treffen?

Die KI-Landschaft

Anwendungsfelder	**Methoden**
Sprachverstehende und -generierende Systeme	Problemlösen und Suche
Bildverstehende Systeme	Wissensrepräsentation und -verarbeitung
Robotik	Handlungsplanung
Assistenzsysteme	Maschinelles Lernen
	Verarbeitung unsicheren Wissens
	neuronale Netze

mit interdisziplinären Beziehungen zu Mathematik, Philosophie, Psychologie, (Computer-) Linguistik, Biologie, Ingenieurwissenschaften, ...

Anfänge der KI

Philosophie, Mathematik, Psychologie, Linguistik und Informatik haben alle seit ihren Anfängen

- ähnliche Fragen gestellt
- und Methoden und Ergebnisse für die KI entwickelt.

Entstehung der KI (1943–1956): Mit der Entwicklung der ersten Rechenanlagen beschäftigten sich auch die ersten Leute mit der Frage: Können Computer den menschlichen Geist nachbilden ($\rightsquigarrow$ Turing-Test)?

40 Jahre KI (1)

1956: Dartmouth Workshop – McCarthy schlägt den Term *Artificial Intelligence* vor – und früher Enthusiasmus:

> It is not my aim to surprise or shock you—but the simplest way I can summarize is to say that there are now in the world machines that think, that learn and that create. Moreover, their ability to do these things is going to increase rapidly until—*in the visible future*—the range of problems they can handle will be coextensive with the range to which human mind has been applied. [Simon, 1957]

60'er: „Intelligentes Verhalten" wird in vielen Demonstrationssystemen (für Mikrowelten) Realität (Blocks world)

70'er: Probleme:

- Systeme für Mikrowelten nicht skalierbar $\leadsto$ „reale" Anwendungen
- „Intelligentes Verhalten" benötigt viel Wissen $\leadsto$ Wissensbasierte Systeme

40 Jahre KI (2)

80'er: Kommerzieller Erfolg der Expertensysteme (z.B. R1), intensive Forschungsförderung (z.B. *Fifth generation computer systems project* in Japan), LISP-Maschinen, die Rückkehr neuronaler Netze

Ende der 80'er: Entzauberung der Expertensysteme, Ende des *Fifth generation computer systems projects*, „KI Winter"

90'er: Einzug probabilistischer Methoden, agentenorientierte Sichtweise, Formalisierung und Mathematisierung von KI-Techniken:

> . . . gentle revolutions have occured in robotics, computer vision, machine learning (including neural networks), and knowledge representation. A better understanding of the problems and their complexity properties, combined with increased mathematical sophistication, has led to workable research agendas and robust methods. [Russell & Norvig, 1995]

. . . und heutzutage?

- Viele koexistierende Paradigmen:

$\leadsto$ *reaktive* vs. *deliberative* Ansätze (Robotik)

$\leadsto$ *probabilistisch* vs. *analytisch* (Computerlinguistik)

$\leadsto$ . . . oft auch hybride Ansätze

- Viele Methoden (z.T. aus anderen Disziplinen):

$\leadsto$ logisch, entscheidungstheoretisch, algorithmisch, . . .

- Viele Herangehensweisen:

$\leadsto$ theoretisch, experimentell algorithmisch, systemorientiert

- Viele Methoden werden heute nicht mehr uneingeschränkt mehr als KI-Methoden angesehen. Beispiele: Programme für Brettspiele, logisches Programmieren (PROLOG), Suchverfahren, . . .

Beispiele: algorithmisch, experimentelle Arbeiten

Viele KI-Probleme sind inhärent schwierig (NP-hart), aber man kann mit guten Suchtechniken und Heuristiken trotzdem Probleminstanzen bis zu einer gewissen Größe lösen:

- Erfüllbarkeit boole'scher Formeln

$\leadsto$ randomisierte, lokale Suchtechniken (bis zu 2500 Variablen für schwere Instanzen)

- Constraint-Propagierung und Backtracking-Techniken

$\leadsto$ empirische und analytische Vergleiche verschiedener Techniken

- Handlungsplanung

$\leadsto$ empirische Vergleiche verschiedener Ansätze und Systeme

- . . .

Systeme

Neben Theorie und der Analyse von einzelnen Algorithmen ist der Bau von Systemen und ihre Anwendung ein wesentlicher Punkt:

Simon in einem Vortrag mit dem Titel „How to become a good scientist“ (1998):

„Build a System“

- Anwendungen von KI-Techniken, um reale Probleme zu lösen
- Studium der Wechselwirkung der Artefakte mit ihrer Umwelt
- synergetische Effekte in Systemen

⇝ VERBMOBIL: Übersetzung gesprochener Sprache

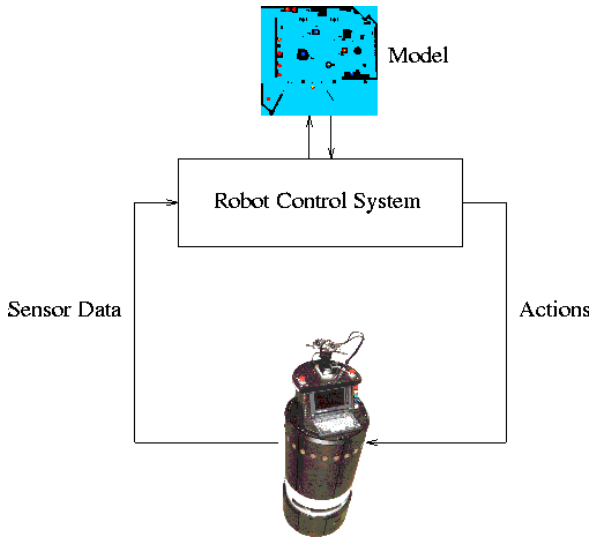
⇝ Rhino & Minerva: Roboter als interaktive Museumsführer

Beispiel: Das VERBMOBIL-System

- Sprachtechnologieprojekt im Bereich Maschinelle Übersetzung (BMBF-Verbundprojekt)
- Erkennung und Analyse gesprochener Eingaben, Übersetzung ins Englische, gesprochene Ausgabe
- Domäne: Terminverhandlung, Aufgabe: Übersetzunghilfe

- 1993-1996: Phase I mit Abschlussdemonstration 1996
- 1996-2000: Phase II

Robotik: Das allgemeine Problem



Beispiel: Autonome Roboter Rhino & Minerva

Grundlagen der KI

2. Rationale Agenten

Natur und Struktur rationaler Agenten und ihre Umgebungen

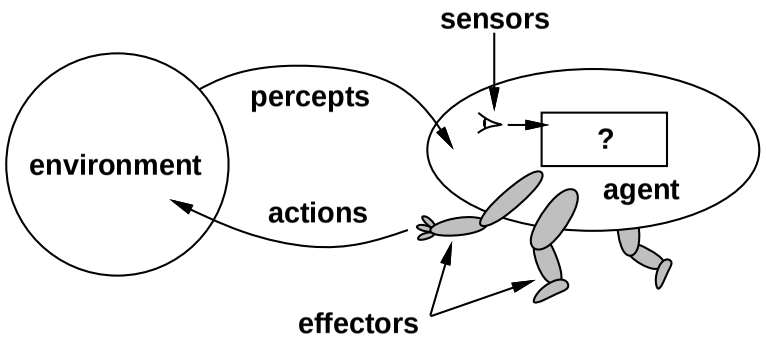
Wolfram Burgard

Inhalt

- Was ist ein Agent?
- Was ist ein *rationaler* Agent?
- Die Struktur rationaler Agenten
- Klassen von Agenten
- Umgebungen

Agenten

- nehmen durch Sensoren ihre Umwelt wahr ($\rightsquigarrow$ *Perzepte*)
- manipulieren ihre Umwelt mit Hilfe ihrer Effektoren ($\rightsquigarrow$ *Aktionen*)



Beispiele:

Menschen und Tiere, Roboter und Software-Agenten (Softbots), Heizungen, ABS . . .

Rationale Agenten

. . . machen das „Richtige“!

Zur Beurteilung müssen wir objektive *Leistungskriterien* anlegen.

Beispiel autonomer Staubsauger:

- m^2 pro Stunde
- Reinheitsgrad
- Stromverbrauch
- Geräuschemission
- Sicherheit
  (Verhalten gegenüber Hamster/Kleinkind)

Optimales Verhalten oft unmöglich

- nicht alle relevanten Informationen wahrnehmbar
- Berechnungskomplexität zu hoch

Rationalität vs. Allwissenheit

- Ein *allwissender* Agent kennt die *tatsächlichen Effekte* seiner Aktionen
- Ein *rationaler* Agent handelt dagegen auf Grund seiner Wahrnehmungen (*Perzepte*) und seines Wissens und versucht die *erwartete Leistung* zu maximieren
- **Beispiel:** Wenn ich beim überqueren der Straße vorher schaue, ob die Straße frei ist, und dann beim überqueren von einem Meteoriten erschlagen werde, kann man mich kaum mangelnder Rationalität beschuldigen.

Idealer rationaler Agent

rationales Handeln abhängig von

- Leistungskriterien (Ziele)
- *Wahrnehmungssequenzen*
- Weltwissen
- mögliche Aktionen

Idealer rationaler Agent =

> Agent, der für alle möglichen Wahrnehmungssequenzen + gegebenem Weltwissen die Aktion wählt, die die Leistung maximiert.

Aktive Wahrnehmung nötig, um Trivialisierung zu vermeiden.

Ideal rationaler Agent realisiert durch eine Funktion:

$$\text{Wahrnehmungssequenz} \times \text{Weltwissen} \mapsto \text{Aktion}$$

Beispiele rationaler Agenten:

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Die Struktur rationaler Agenten

Realisierung der idealen Abbildung durch ein

- *Agenten-Programm*, das auf einer
- *Architektur* ausgeführt wird, die auch die Schnittstelle zur Umwelt realisiert (Perzepte, Aktionen)

$\rightsquigarrow$ *Agent* $=$ *Architektur* $+$ *Programm*

Skelettprogramm für einen Agenten

```
function SKELETON-AGENT(percept) returns action
  static: memory, the agent's memory of the world

  memory ← UPDATE-MEMORY(memory, percept)
  action ← CHOOSE-BEST-ACTION(memory)
  memory ← UPDATE-MEMORY(memory, action)
  return action
```


Das einfachste Design: Tabellengesteuerte Agenten

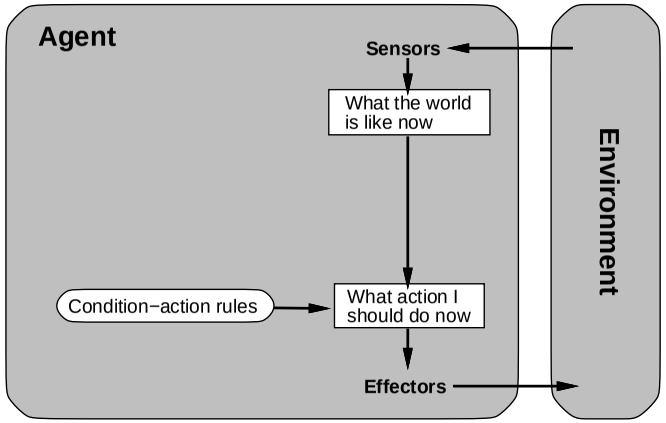
```
function TABLE-DRIVEN-AGENT(percept) returns action
  static: percepts, a sequence, initially empty
          table, a table, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

Probleme:

- Die Tabelle kann *sehr* groß werden
- und es braucht entsprechend sehr lange, bis die Tabelle vom Designer erstellt wird (oder erlernt wird)
- . . . tatsächlich praktisch unmöglich

Reflexive/reaktive Agenten



Direkte Benutzung von Wahrnehmungen oft nicht möglich, da Wahrnehmungsraum zu groß (z.B. bei Videobildern)

Interpretierende, reflexive Agenten

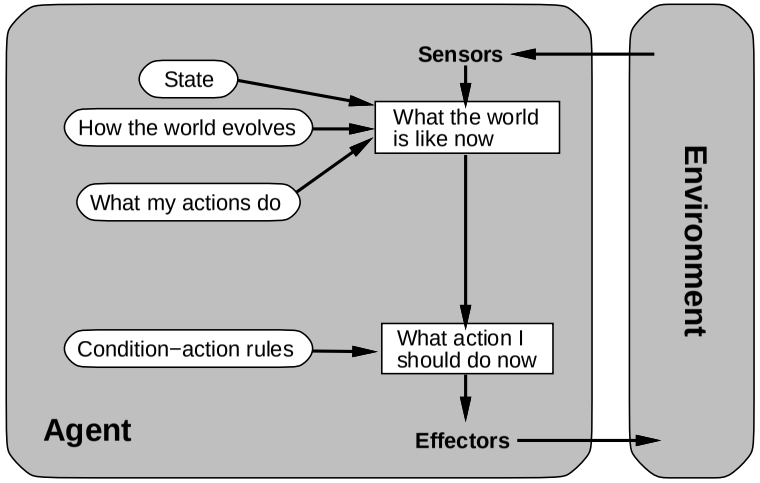
Da Wahrnehmungsraum zu groß, direkte Interpretation von Perzepten

```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

Agenten mit internem Weltmodell

Falls neben dem aktuellen Perzept auch die Historie für die Auswahl von Aktionen erforderlich ist, muss diese in geeigneter Form repräsentiert werden.



Ein Agenten-Programm mit internem Zustand

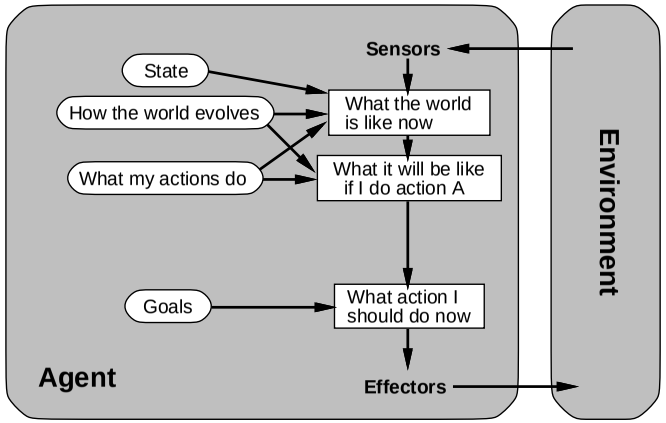
function REFLEX-AGENT-WITH-STATE(*percept*) **returns** *action*
  static: *state*, a description of the current world state
    rules, a set of condition-action rules

  state ← UPDATE-STATE(*state*, *percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← RULE-ACTION[*rule*]
  state ← UPDATE-STATE(*state*, *action*)
  return *action*

Agenten mit expliziten Zielen

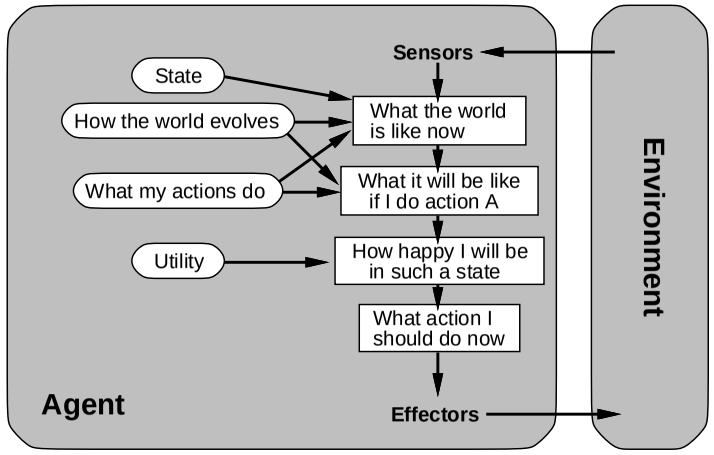
Oft sind die Perzepte allein für die Aktionsauswahl nicht ausreichend, da die richtige Aktion von den explizit vorgegebenen Zielen abhängt (z.B. nach X fahren).

$\rightsquigarrow$ Explizite Repräsentation von Zielen und deren Berücksichtigung bei Aktionsauswahl.



Nutzenbasierte Agenten

Meist gibt es mehrere mögliche Aktionen, die in einem Zustand ausgeführt werden können. In solchen Fällen kann der Nutzen (*Utility*) des erreichten Zustands herangezogen werden, um eine Auswahl zu treffen.



Die Umgebung rationaler Agenten

- zugänglich vs. unzugänglich

  Sind alle relevanten Aspekte der Welt den Sensoren zugänglich?

- deterministisch vs. nichtdeterministisch

  Hängt der nächste Weltzustand allein vom jetzigen Zustand und der ausgeführten Aktion ab?

- episodisch vs. nichtepisodisch

  Kann die Qualität einer Aktion innerhalb einer Episode (Wahrnehmung + Aktion) bewertet werden oder ist die zukünftige Entwicklung für die Qualitätsbewertung ausschlaggebend?

- statisch vs. dynamisch

  Kann sich die Welt ändern, während der Agent reflektiert (nachdenkt)?

- diskret vs. kontinuierlich

  ist die Welt diskret (Schach spielen) oder nicht (Roboter, der sich im Raum bewegt)?

Beispiele für Umgebungen

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

Ob eine Umgebung eine bestimmte Eigenschaft hat, hängt auch von der Konzeption des Designers ab.

Evaluierung eines Agenten in einer Umgebung

Um Agenten zu evaluieren, müssen wir sie in einer Umgebung agieren lassen – Simulation:

```
function RUN-EVAL-ENVIRONMENT(state, UPDATE-FN, agents,
                              termination, PERFORMANCE-FN) returns scores
  local variables: scores, a vector the same size as agents, all 0

  repeat
    for each agent in agents do
      PERCEPT[agent] ← GET-PERCEPT(agent, state)
    end
    for each agent in agents do
      ACTION[agent] ← PROGRAM[agent](PERCEPT[agent])
    end
    state ← UPDATE-FN(actions, agents, state)
    scores ← PERFORMANCE-FN(scores, agents, state)
  until termination(state)
  return scores                                        /* change */
```

Simulationen ersetzen nie das Experimentieren in der realen Umgebung!

Zusammenfassung

- Ein **Agent** ist etwas, was wahrnimmt und agiert. Es besteht aus einer *Architektur* und einem *Agentenprogramm*.
- Ein **idealer rationaler Agent** führt die Aktionen aus, die für gegebene Wahrnehmungssequenzen und gegebenes Weltwissen die Leistung maximieren.
- Ein **Agentenprogramm** bildet Wahrnehmungssequenzen auf Aktionen ab.
- Es existiert eine Vielzahl verschiedener Designs
  - **Reaktive** Agenten wählen Entscheidungen auf der Basis der Wahrnehmungen
  - **Zielbasierte** versuchen gegebene Ziele zu erreichen
  - **Nutzenbasierte Agenten** maximieren ihre Bewertung
- Einige Typen von **Umgebungen** sind anspruchsvoller als andere. Unzugängliche, nicht-episodische, dynamische, kontinuierliche Umgebungen sind die schwierigsten.

Grundlagen der KI

3. Problemlösen durch Suche

Problemlösende Agenten, Problemformulierungen, Suchstrategien

Wolfram Burgard

Inhalt

- Problemlösende Agenten
- Problemformulierungen
- Problemtypen
- Beispielprobleme
- Suchstrategien
- *Constraint-Satisfaction* Probleme

Problemlösende Agenten

$\Rightarrow$ zielorientierte Agenten

Formuliere: *Ziel* und *Problem*

Gegeben: Anfangszustand

Gewünscht: Erreichen eines bestimmten Ziels (eines Zustandes) durch Ausführen geeigneter Aktionen

$\rightsquigarrow$ **Suche** einer geeigneten **Aktionsfolge** und **Ausführung** dieser Folge

Ein einfacher problemlösender Agent

function SIMPLE-PROBLEM-SOLVING-AGENT(*p*) **returns** an action
  inputs: *p*, a percept
  static: *s*, an action sequence, initially empty
    state, some description of the current world state
    g, a goal, initially null
    problem, a problem formulation

  state $\leftarrow$ UPDATE-STATE(*state*, *p*)
  if *s* is empty **then**
    g $\leftarrow$ FORMULATE-GOAL(*state*)
    problem $\leftarrow$ FORMULATE-PROBLEM(*state*, *g*)
    s $\leftarrow$ SEARCH(*problem*)
  action $\leftarrow$ RECOMMENDATION(*s*, *state*)
  s $\leftarrow$ REMAINDER(*s*, *state*)
  return *action*

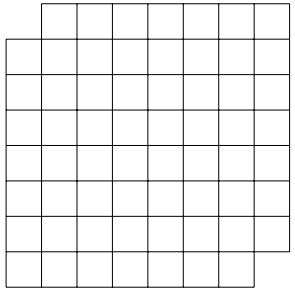
Problemformulierung

- Formulierung des *Ziels*
  Weltzustände mit bestimmten Eigenschaften
- Festlegen des *Weltzustandsraums*
  (wichtig: nur die relevanten Aspekte $\rightsquigarrow$ Abstraktion)
- Festlegen der *Aktionen*, die einen Weltzustand in einen anderen überführen
- Bestimmung des *Problemtyps*, der abhängig vom *Wissen* über Weltzustände und Aktionen ist $\rightsquigarrow$ Zustände im Suchraum
- Bestimmung der Kosten für das Suchen (Suchkosten, *Offline*-Kosten) und der Ausführungskosten (Pfadkosten, *Online*-Kosten)

Achtung: Die Art der Problemformulierung kann einen großen Einfluss auf die Schwierigkeit der Lösung haben.

Beispiel für Problemformulierung

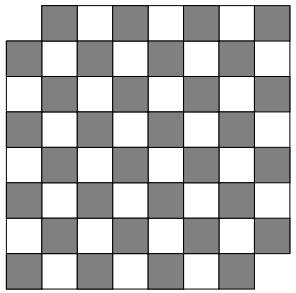
Gegeben ein $n \times n$ Brett, bei dem an den beiden diagonal gegenüberliegenden Ecken je ein Feld entfernt wurde (hier 8×8):



Ziel: Das Brett so mit Dominosteinen, die jeweils zwei benachbarte Felder überdecken, belegen, dass alle Felder abgedeckt werden.

$\rightsquigarrow$ Ziel, Zustandsraum, Aktionen, Suche ...

Alternative Problemformulierung

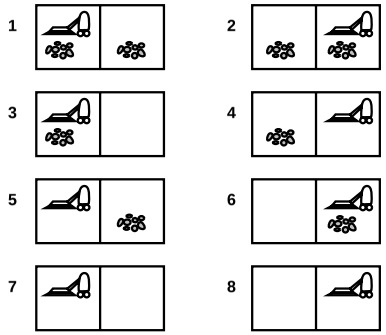


Frage: Kann man ein Brett, auf dem es $n^2/2$ schwarze und $n^2/2 - 2$ weiße Felder gibt, so mit Dominosteinen, die jeweils ein weißes und ein schwarzes Feld überdecken, belegen, dass alle Felder abgedeckt sind?

...natürlich nicht

Problemformulierung für die Staubsaugerwelt

- Weltzustandsraum: 2 Positionen, Schmutz oder kein Schmutz
  $\rightsquigarrow$ 8 Weltzustände



- Aktionen: links (L), rechts(R), saugen(S)
- Ziel: kein Schmutz in den Räumen
- Pfadkosten: pro Aktion 1 Einheit

Problemtypen: Wissen über Zustände und Aktionen

- Einzustandsproblem

  vollständiges Weltzustandswissen,
  vollständiges Aktionswissen

  ⇝ der Agent weiß immer, in welchem Weltzustand er ist.

- Mehrzustandsproblem

  unvollständiges Weltzustandswissen oder
  unvollständiges Aktionswissen

  ⇝ der Agent weiß nur, in welcher Menge von Weltzuständen er ist.

- Kontingenzproblem

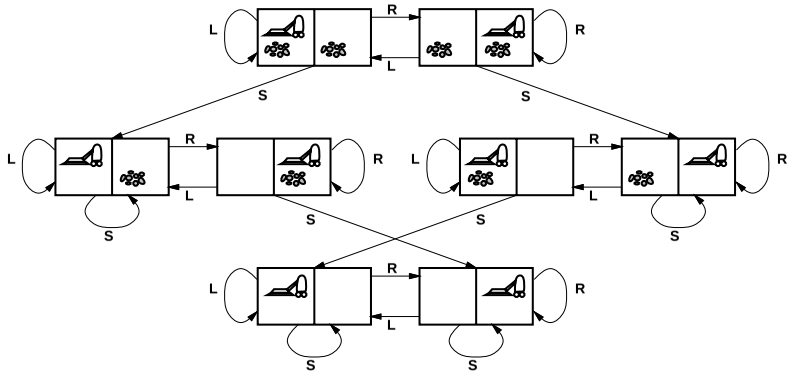
  es ist unmöglich, eine komplette Sequenz von Aktionen zur Lösung im voraus zu bestimmen, da nicht alle Informationen über Zwischenzustände vorliegen.

- Explorationsproblem

  Zustandsraum und Effekte der Aktionen nicht vollständig bekannt. Schwer!

Die Staubsaugerwelt als Einzustandsproblem

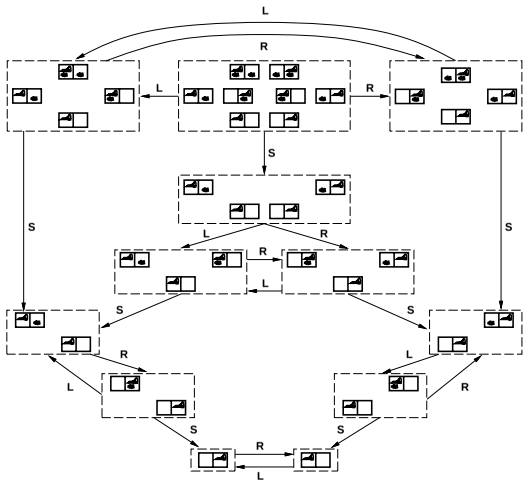
Falls die Welt vollständig zugänglich ist, weiß der Staubsauger immer, wo er und der Schmutz sind. Problemlösen reduziert sich dann auf die Suche nach einem Pfad von unserem Anfangszustand zu einem Zielzustand.



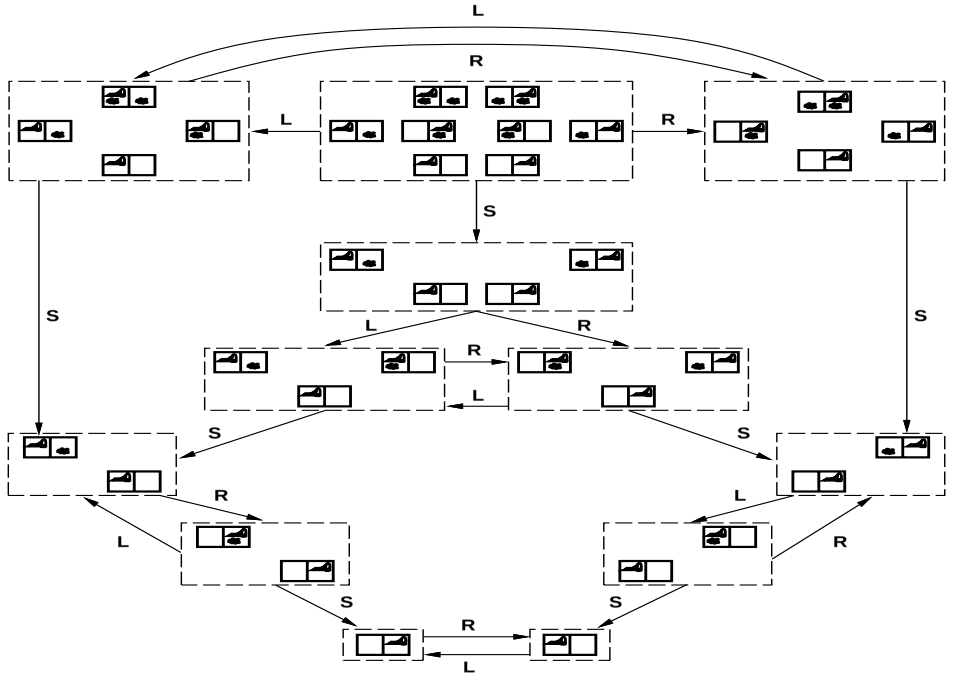
Zustände für die Suche: Die Weltzustände 1–8.

Die Staubsaugerwelt als Mehrzustandsproblem

Falls der Staubsauger keine Sensoren besitzt, weiß er nicht, wo er ist und wo Schmutz ist. Trotzdem kann er das Problem lösen. Zustände sind dann *Wissenszustände*:



Zustände für die Suche: Potenzmenge der Weltzustände 1–8.



Begriffe (1)

Anfangszustand

Zustand, von dem der Agent glaubt, anfangs zu sein

Zustandsraum

Menge aller möglichen Zustände

Operator

Beschreibung einer Aktion durch Angabe des resultierenden Zustands

Zieltest

Test, ob die Beschreibung eines Zustands einem Zielzustand entspricht

Begriffe (2)

Pfad

Sequenz von Aktionen, die von einem Zustand zu einem anderen führen. *Pfadkosten*: Kostenfunktion g über Pfaden. Setzt sich üblicherweise aus der Summe der Kosten der Aktionen zusammen.

Lösung

Pfad von einem Anfangs- zu einem Zielzustand.

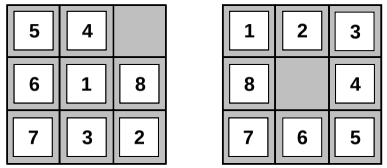
Suchkosten

Zeit- und Speicherbedarf, um eine Lösung zu finden.

Gesamtkosten

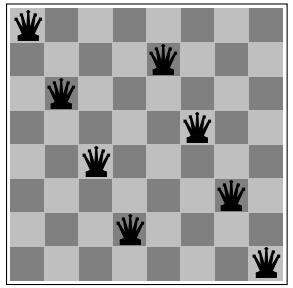
Suchkosten + Pfadkosten.

Beispiel: 8er-Puzzle



- Zustände:
  - Beschreibung der Lage jedes der 8 Kästchen und (aus Effizienzgründen) des Leerkästchens.
- Operatoren:
  - „Verschieben“ des Leerkästchens nach links, rechts, oben und unten.
- Zieltest:
  - Entspricht aktueller Zustand dem rechten Bild?
- Pfadkosten
  - Jeder Schritt kostet 1 Einheit.

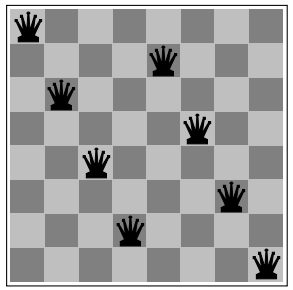
Beispiel: 8 Damen Problem



- Zieltest
  - 8 Damen auf dem Brett, keine angreifbar
- Pfadkosten: 0 (nur die Lösung interessiert)
- Darstellung 1
  - Zustände: beliebige Anordnung von 0–8 Damen
  - Operatoren: setze eine der Damen aufs Brett
  - Problem: 64^8 Zustände

8 Damen Problem (2)

- Darstellung 2
  - Zustände: Anordnung von 0–8 Damen in unangreifbarer Stellung
  - Operatoren: Setze eine der Damen soweit wie möglich links unangreifbar auf das Brett
  - Problem: wenige Zustände (2057), aber manchmal keine Aktion möglich:



- Darstellung 3
  - Zustände: 8 Damen auf dem Brett, eine in jeder Spalte
  - Operatoren: verschiebe eine angegriffene Dame in derselben Spalte

Beispiel: Missionare und Kannibalen

Informelle Problembeschreibung:

- An einem Fluss haben sich 3 Kannibalen und 3 Missionare getroffen, die alle den Fluss überqueren wollen.
- Es steht ein Boot zur Verfügung, das maximal zwei Leute aufnehmen kann.
- Es sollte nie die Situation auftreten, dass an einem Ufer Missionare und Kannibalen sind und dabei die Anzahl der Kannibalen die Anzahl der Missionare übertrifft.

$\rightsquigarrow$ Finde eine Aktionsfolge, die alle an das andere Ufer bringt.

Formalisierung des MuK-Problems

Zustände: Tripel (x,y,z) mit $0 \leq x, y, z \leq 3$, wobei x, y und z angeben, wieviele Missionare, Kannibalen und Boote sich zur Zeit am Ausgangsufer befinden.

Anfangszustand: (3,3,1)

Operatoren: Von jedem Zustand aus entweder einen Missionar, einen Kannibalen, zwei Missionare, zwei Kannibalen, oder einen von jeder Sorte über den Fluss bringen. (also 5 Operatoren)

Beachte: nicht jeder Zustand damit erreichbar [z.B. (0,0,1)] und einige sind illegal.

Endzustand: (0,0,0)

Pfadkosten: 1 Einheit pro Flussüberquerung

Beispiele für reale Probleme

- Routenplanung, Finden kürzester Pfade

  Im Prinzip einfach (polynomiales Problem). Komplikationen bei unbekannten, sich dynamisch verändernden Pfadkosten (Beispiel: Routenplanung in Kanada)

- Planung von Rundreisen (TSP)

  Eines der prototypischen NP-vollständigen Probleme.

- VLSI Layout

  Auch ein NP-vollständiges Problem.

- Roboter Navigation (mit vielen Freiheitsgraden)

  Schwierigkeit nimmt mit der Anzahl der Freiheitsgrade extrem zu.

  Weitere mögliche Komplikationen: Fehler bei Wahrnehmungen, unbekannte Umgebungen.

- Montageplanung

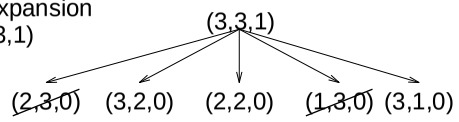
  Planung des Zusammenbaus von komplexen Objekten.

Suche allgemein

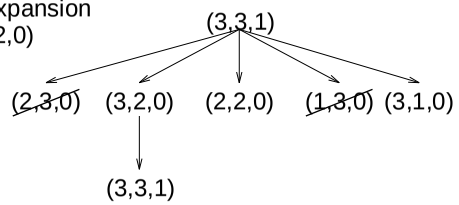
Ausgehend vom Anfangszustand schrittweise alle Nachfolgezustände erzeugen $\rightsquigarrow$ *Suchbaum*.

(a) Anfangszustand (3,3,1)

(b) nach Expansion von (3,3,1)



(c) nach Expansion von (3,2,0)



Allgemeine Suchprozedur

function GENERAL-SEARCH(*problem, strategy*) **returns** a solution, or failure
  initialize the search tree using the initial state of *problem*
  loop do
    if there are no candidates for expansion **then return** failure
    choose a leaf node for expansion according to *strategy*
    if the node contains a goal state **then return** the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end

Implementierung des Suchbaums

Datenstruktur für Knoten im Suchbaum:

State: Zustand des Zustandsraums

Parent-Node: Vorgängerknoten

Operator: Operator, der den aktuellen Knoten erzeugt hat

Depth: Tiefe im Suchbaum

Path-Cost: Pfadkosten bis zu diesem Knoten

Funktionen zum Manipulieren einer Warteschlange (*Queue*):

Make-Queue(*Elements*): Erzeugt eine Queue

Empty?(*Queue*): Testet auf Leerheit

Remove-Front(*Queue*): Gibt erstes Element zurück

Queuing-Fn(*Elements, Queue*): Fügt neue Elemente ein (verschiede Möglichkeiten)

Allgemeine Suche . . . konkret

function GENERAL-SEARCH(*problem*, QUEUING-FN) **returns** a solution, or failure

  nodes $\leftarrow$ MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[*problem*]))
  loop do
    if *nodes* is empty **then return** failure
    node $\leftarrow$ REMOVE-FRONT(*nodes*)
    if GOAL-TEST[*problem*] applied to STATE(*node*) succeeds **then return** *node*
    nodes $\leftarrow$ QUEUING-FN(*nodes*, EXPAND(*node*, OPERATORS[*problem*]))
  end

Suchstrategien (1)

Kriterien:

Vollständigkeit: Wird immer eine Lösung gefunden, sofern es eine gibt?

Zeitkomplexität: Wie lange dauert es (im schlechtesten Fall), bis eine Lösung gefunden ist?

Platzkomplexität: Wieviel Speicher benötigt die Suche (im schlechtesten Fall)?

Optimalität: Findet das Verfahren immer die beste Lösung?

Suchstrategien (2)

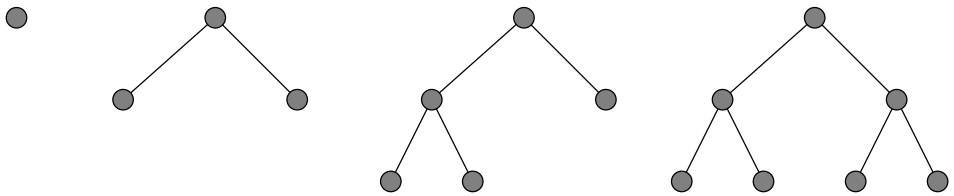
uninformierte oder *blinde Suche*:
keine Information über die Länge oder Kosten eines Lösungspfades.

- Breitensuche, uniforme Kostensuche, Tiefensuche,
- tiefenbeschränkte Suche, iterative Tiefensuche,
- bidirektionale Suche

im Unterschied dazu:
informierte oder *heuristische Suche*.

Breitensuche (1)

Expandiere Knoten in der Reihenfolge, in der sie erzeugt werden (*Queue-Fn* = *Enqueue-at-end*).



- Findet immer die flachste Lösung (vollständig).
- Die Lösung ist optimal, wenn Pfadkosten eine nichtfallende Funktion der Knotentiefe ist (z.B. wenn jede Aktion identische, nichtnegative Kosten hat).

Breitensuche (2)

- Allerdings sind die Kosten sehr hoch. Sei b der maximale Verzweigungsfaktor, d die Tiefe eines Lösungspfads. Dann müssen maximal

$$1 + b + b^2 + b^3 + \ldots + b^d$$

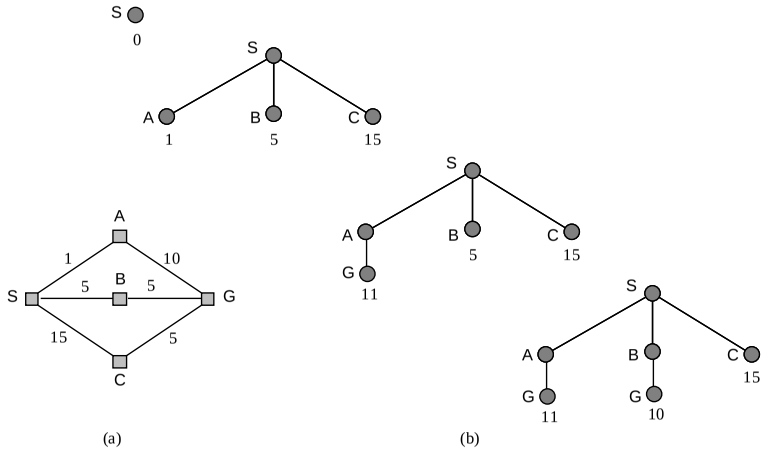
Knoten expandiert werden, also $O(b^d)$.

Beispiel: $b = 10$, 1000 Knoten/s; 100 Bytes/Knoten:

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Uniforme Kostensuche

In Abwandlung der Breitensuche wird immer der Knoten n mit den geringsten Pfadkosten $g(n)$ expandiert.

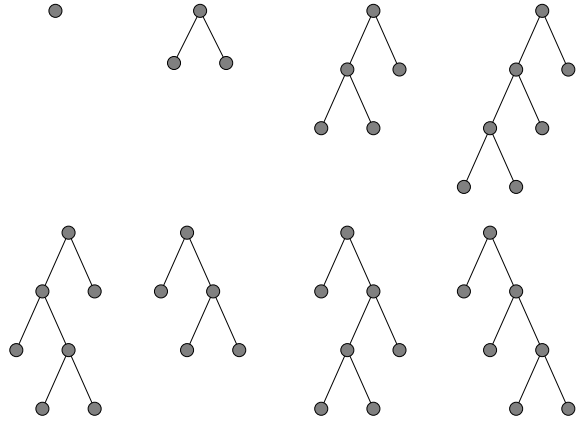


Findet immer die günstigste Lösung, falls $g(succ(n)) \geq g(n)$ für alle n.

Tiefensuche

Expandiere immer einen nicht expandierten Knoten mit maximaler Tiefe (*Queue-Fn* = *Enqueue-at-front*).

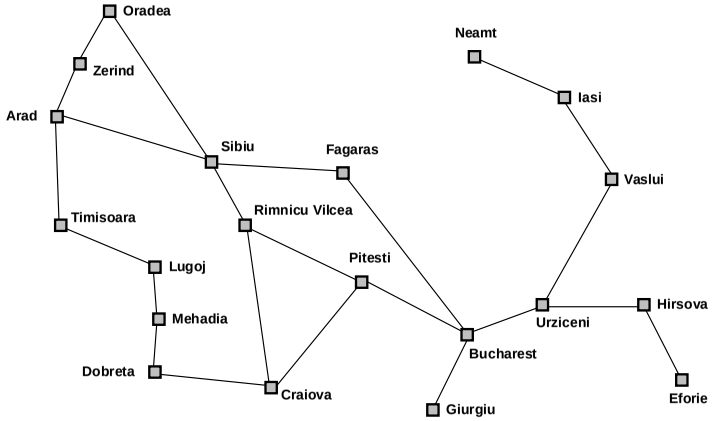
Beispiel (Knoten der Tiefe 3 haben keine Nachfolger):



Tiefenbeschränkte Suche

Es wird nur bis zu einer vorgegebenen Pfadlänge Tiefensuche durchgeführt.

z.B. Wegeplanung: Bei n Städten ist die maximale Tiefe $n - 1$.



Hier reicht maximale Tiefe 9 (*Durchmesser* des Problems)

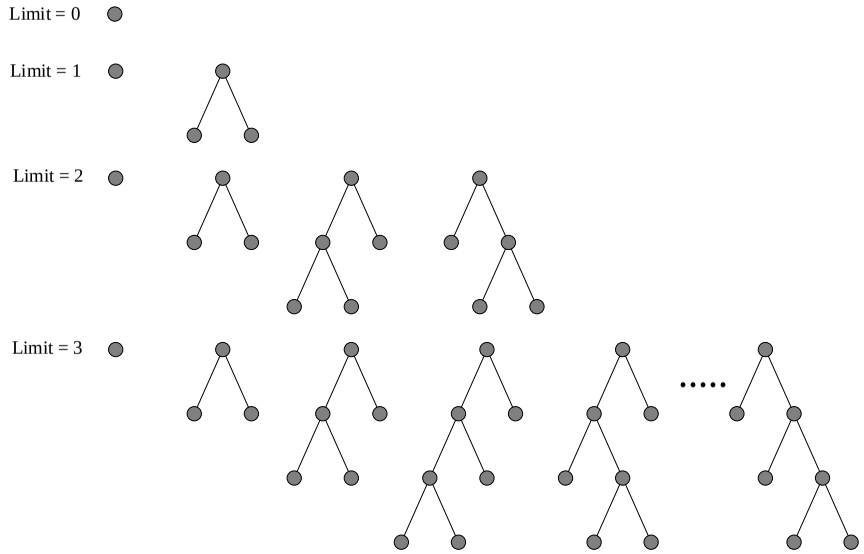
Iterative Tiefensuche (1)

- kombiniert Tiefen- und Breitensuche.
- optimal und vollständig wie Breitensuche, aber weniger Speicherplatz.

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution sequence
  inputs: *problem*, a problem

  for *depth* $\leftarrow$ 0 **to** ∞ **do**
    if DEPTH-LIMITED-SEARCH(*problem*, *depth*) succeeds **then return** its result
  end
  return failure

Beispiel



Iterative Tiefensuche (2)

Zahl der Expansionen

Iterative Tiefensuche	$(d+1)1 + (d)b + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d$
Breitensuche	$1 + b + b^2 + b^3 + \ldots + b^{d-1} + b^d$

Beispiel: $b = 10$, $d = 5$

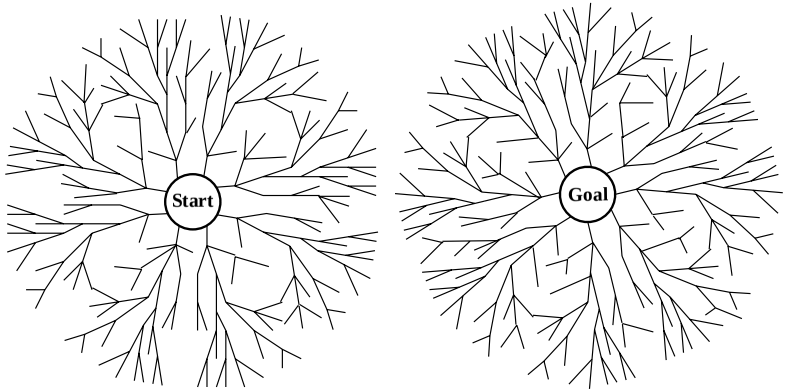
Breitensuche:	1 + 10 + 100 + 1000 + 10000 + 100000 = 111111
Iterative Tiefensuche:	6 + 50 + 400 + 3000 + 20000 + 100000 = 123456

Für $b = 10$ werden nur 11% mehr Knoten expandiert als bei Breitensuche, wobei der Platzbedarf erheblich niedriger ist!

Zeitkomplexität: $O(b^d)$ *Platzkomplexität*: $O(b \times d)$

$\rightsquigarrow$ Iterative Tiefensuche ist nicht viel schlechter und i.allg. die bevorzugte Suchmethode bei großen Suchräumen mit unbekannter maximaler Suchtiefe.

Bidirektionale Suche



- Sofern Vorwärts- und Rückwärtssuche symmetrisch sind, erreicht man Suchzeiten von $O(2 \times b^{d/2}) = O(b^{d/2})$.

  z.B. für $b = 10$, $d = 6$ statt 1111111 nur 2222 Knoten!

Probleme mit bidirektionaler Suche

- Die Operatoren sind nicht immer oder nur sehr schwer umkehrbar (Berechnung der Vorgängerknoten).
- In manchen Fällen gibt es sehr viele Zielzustände, die nur unvollständig beschrieben sind. Beispiel: Vorgänger des „Schachmatt“.
- Man braucht effiziente Verfahren, um zu testen, ob sich die Suchverfahren „getroffen“ haben.
- Welche Art der Suche wählt man für jede Richtung (im Bild: Breitensuche, die nicht immer optimal ist)?

Vergleich der Suchstrategien

- Zeitkomplexität
- Platzkomplexität
- Optimalität
- Vollständigkeit

b: Verzweigungsfaktor, d: Tiefe der Lösung,
m: maximale Tiefe des Suchbaums, l: Tiefenlimit

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Constraint-Satisfaction-Probleme (CSP)

CSPs bilden eine spezielle Problemklasse, die aufgrund struktureller Einschränkungen besondere Suchtechniken erlauben.

- *Zustände* sind durch *Werte* von *Variablen* definiert.
- *Operatoren* belegen eine Variable mit einem Wert.
- Der *Zieltest* wird durch *Constraints* (Bedingungen) spezifiziert, welche die Variablenbelegungen erfüllen müssen.
- Ein *Zielzustand* ist eine Belegung der Variablen mit Werten, die alle *Constraints* erfüllen.

Beispiel: Das 8-Damen Problem als CSP

- Es gibt 8 *Variablen* $V_1, \ldots, V_8$, wobei V_i für die Dame in der i-ten Spalte steht.
- V_i kann einen *Wert* von aus $\{1, 2, \ldots, 8\}$ annehmen, der für die Zeilenposition steht.
- Zwischen allen Paaren von Variablen gibt es *Constraints*, welche die Nichtangreifbarkeit ausdrücken.

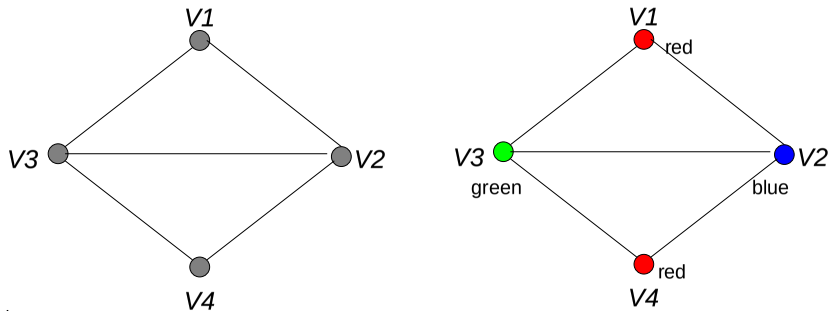
$\rightsquigarrow$ diskretes, endliches, binäres CSP

Einfärben von Graphen als CSP

Gegeben ein Graph mit n Knoten, färbe die Knoten mit k Farben so ein, dass zwei durch eine Kante verbundene Knoten nicht die gleiche Farbe haben ($\rightsquigarrow$ NP-vollständiges Problem!)

Beispiel: Kann der folgende Graph mit 3 Farben eingefärbt werden?

Lösung:



Allgemeine Suche zum CSP-Lösen

- *Tiefensuche* ist bei CSPs vollständig, da ja nur maximal n Operatoren (Belegung von Variablen durch Werte) möglich sind.
- Bei naiver Implementation ist der *Verzweigungsfaktor* sehr hoch! Sei D_i die Menge der möglichen Werte für V_i. Dann ist der Verzweigungsfaktor $b = \sum_{i=1}^{n} D_i$.
- Die Reihenfolge der Instantiierungen der Variablen ist für die Lösung unerheblich. Deshalb man kann bei jeder Expansion eine Variable (nicht-deterministisch) aussuchen, d.h. der Verzweigungsfaktor ist $(\sum_{i=1}^{n} D_i)/n$.
- Nach jeder Operatoranwendung kann geprüft werden, ob bereits *Constraints* verletzt wurden. In dem Fall braucht man den aktuellen Knoten nicht weiter expandieren $\leadsto$ Tiefensuche + Test = *Backtracking* (oder Rücksetzsuche)

Vorwärtstest und Kantenkonsistenz

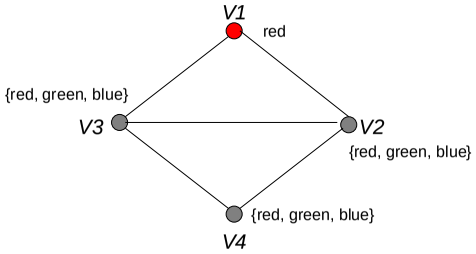
Wenn beim *Backtracking* Entscheidungen getroffen wurden, die eine Lösung unmöglich machen, wird dies u.U. doch erst in den Blättern des Suchbaums bemerkt $\leadsto$ unnötige Exploration des Unterbaums.

Lösungen:

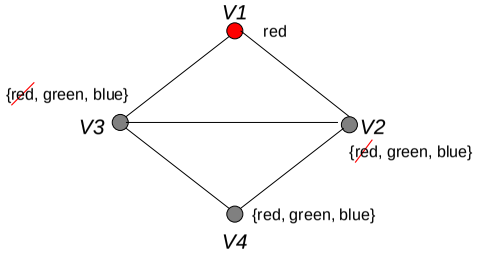
- *Vorwärtstest* (*Forward checking*): Bei allen noch nicht belegten Variablen werden die Werte eliminiert, die nicht mehr möglich sind. Backtracking, falls Domain einer Variable leer wird.
- Spezielle Form des Vorwärtstests: *Kantenkonsistenz* (*Arc consistency*) Ein CSP ist *kantenkonsistent*, falls der Wertebereich jeder Variablen nur Werte enthält, die konsistent mit jedem Constraint auf der Variablen sind.

Beispiel

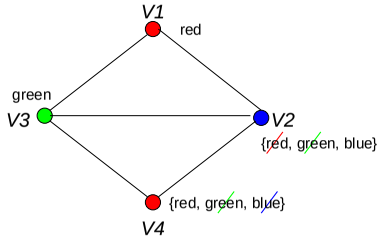
1. Instantiierung $V_1 =$ red:



Herstellen der Kantenkonsistenz:



2. Instantiierung $V_3 =$ green & Herstellen der Kantenkonsistenz:



Zusammenfassung

- Bevor ein Agent beginnen kann, eine Lösung zu suchen, muss er sein Ziel und darauf aufbauend sein Problem definieren.
- Ein Problem besteht aus 5 Teilen: **Zustandsraum**, **Anfangszustand**, **Operatoren**, **Zieltest** und **Pfadkosten**. Ein **Pfad** vom Anfangszustand zu einem Zielzustand ist eine **Lösung**.
- Es existiert ein **genereller Suchalgorithmus**, der benutzt werden kann, um Lösungen zu finden. Spezifische Varianten des Algorithmus benutzen verschiedene **Suchstrategien**.
- Suchalgorithmen werden auf Basis der Kriterien **Vollständigkeit**, **Optimalität**, **Zeit-** und **Platzkomplexität** beurteilt.
- *Constraint-satisfaction-Probleme* bilden eine spezielle Problemklasse, die spezielle Suchtechniken ermöglichen.

Grundlagen der KI

4. Informierte Suche

Heuristiken, lokale Suche, genetische Algorithmen

Wolfram Burgard

Inhalt

- Bestensuche
- A* und IDA*
- Heuristiken für CSPs
- Lokale Suche
- Genetische Algorithmen

Bestensuche

Suchverfahren unterscheiden sich durch die Strategie zur Auswahl des Knotens im Suchbaum, der als nächstes expandiert werden soll.

Uninformierte Suche: starre Strategien ohne Information über Kosten von gegebenem Knoten bis zum Ziel.

Informierte Suche: Information über Kosten von gegebenem Knoten bis zum Ziel in Form einer *Evaluierungsfunktion* h, die jedem Knoten eine reelle Zahl zuweist.

Bestensuche (*best-first search*):
Suchverfahren, das Knoten mit dem „besten“ h-Wert expandiert.

Generischer Algorithmus

function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn $\leftarrow$ a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

$\leadsto$ Wenn h immer richtig ist, brauchen wir nicht zu suchen!

Gierige Suche (Greedy Search)

Eine Möglichkeit die „Güte“ von Knoten zu beurteilen ist es, ihren Abstand zum Ziel zu schätzen.

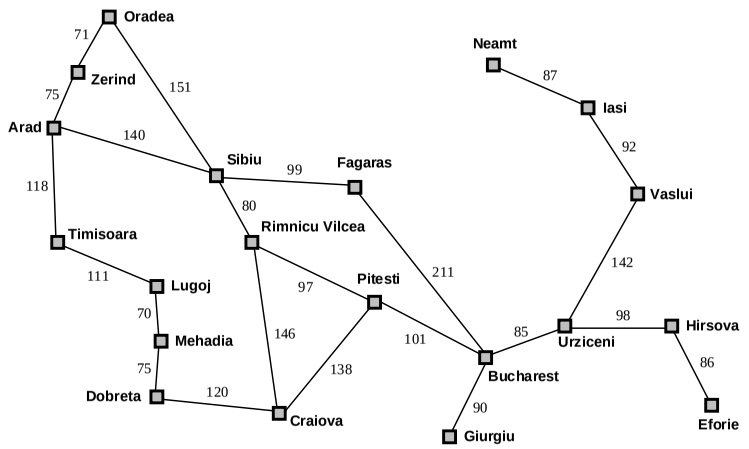
$$h(n) = \text{geschätzter Abstand von } n \text{ zum Ziel}$$

Einzige tatsächliche Einschränkung für h: $h(n) = 0$ falls n Zielknoten.

Bestensuche mit dieser Funktion heißt *gierige Suche*.

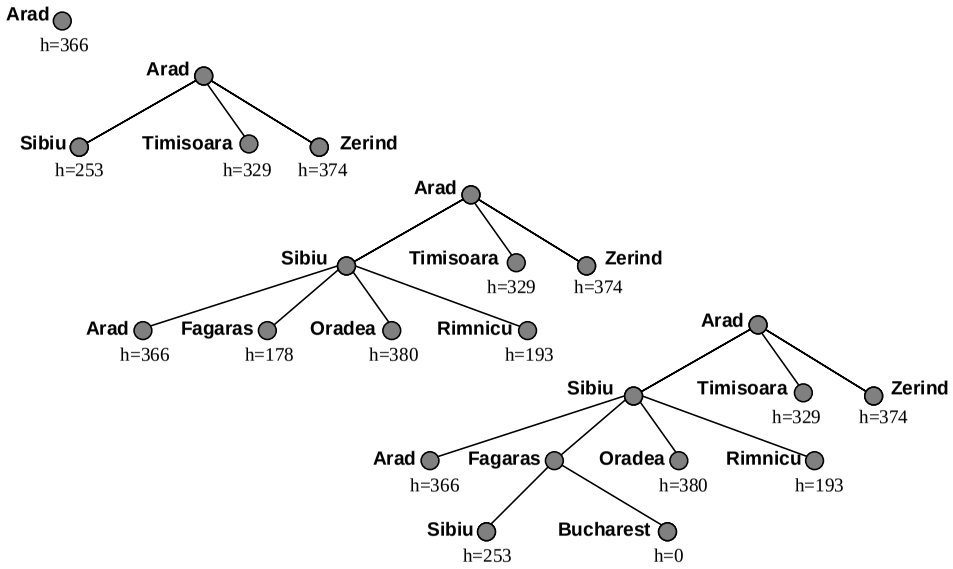
Beispiel Routensuche: $h =$ Luftlinienentfernung zwischen zwei Orten.

Beispiel für gierige Suche



Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Gierige Suche von *Arad* nach *Bucharest*



Heuristiken

Die Evaluierungsfunktion h im Falle gieriger Suche wird auch *heuristische Funktion* oder *Heuristik* genannt.

- Das Wort *Heuristik* ist vom griechischen Verb $\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$ abgeleitet (vgl. auch $H\epsilon\upsilon\rho\epsilon\kappa\alpha$!)
- und wurde vom Mathematiker Polya eingeführt, um Problemlösungstechniken zu beschreiben
- In der KI gibt es zwei Bedeutungen:
  - Heuristiken sind schnelle aber u.U. unvollständige Methoden, um Probleme zu lösen [Newell, Shaw, Simon 1963] (gierige Suche ist tatsächlich i.allg. nicht vollständig)
  - Heuristiken sind Methoden, um die Suche im Normalfall zu beschleunigen.

$\rightsquigarrow$ Auf jeden Fall ist eine Heuristik *problemspezifisch* und *fokussiert* die Suche!

A*: Minimierung der geschätzten Pfadkosten

A* verbindet uniforme Kostensuche mit gieriger Suche.

$g(n)$ = tatsächliche Kosten vom Anfangszustand bis n.

$h(n)$ = geschätzte Kosten von n bis zum nächsten Ziel.

$f(n) = g(n) + h(n)$, d.h. geschätzte Kosten des günstigsten Pfades, der durch n verläuft.

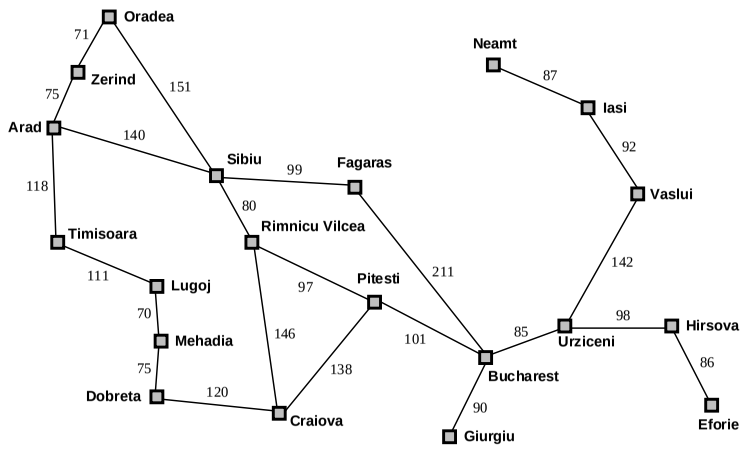
Seien $h^*(n)$ die tatsächlichen Kosten des optimalen Pfades von n zum nächsten Ziel.

h heißt *zulässig*, wenn für alle n gilt:

$$h(n) \leq h^*(n).$$

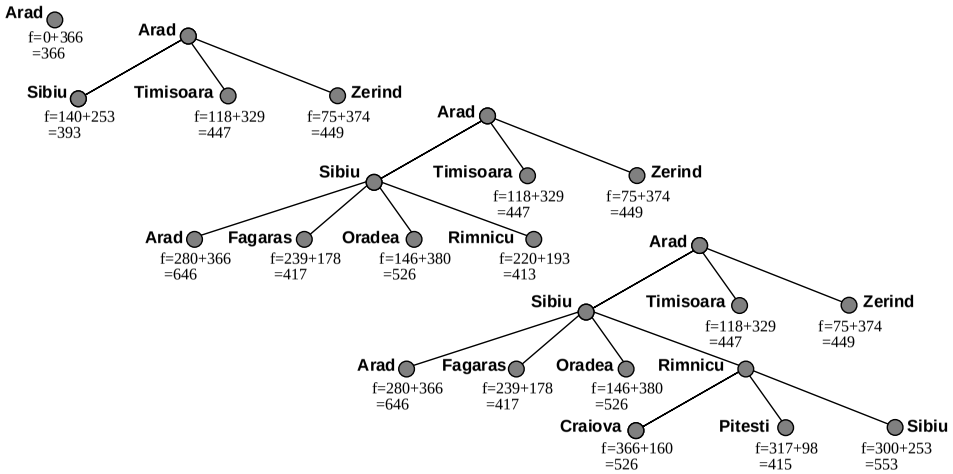
Wir verlangen für A*, dass h zulässig ist (Luftlinienentfernung ist zulässig)

Beispiel für A*-Suche



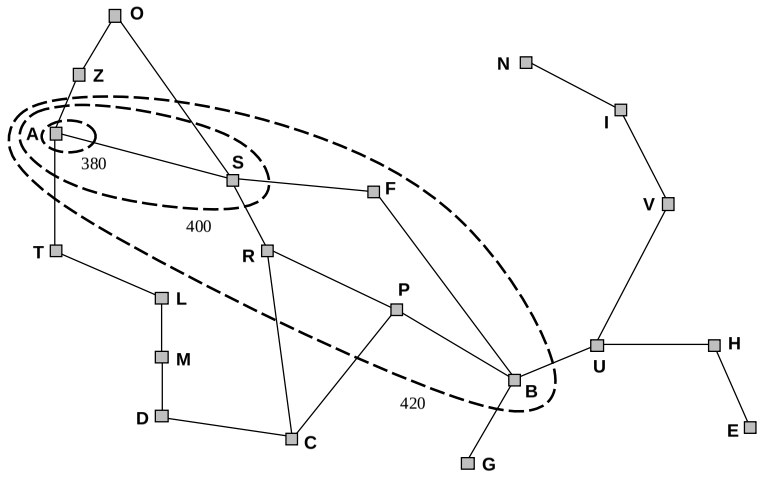
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A*-Suche von *Arad* nach *Bucharest*



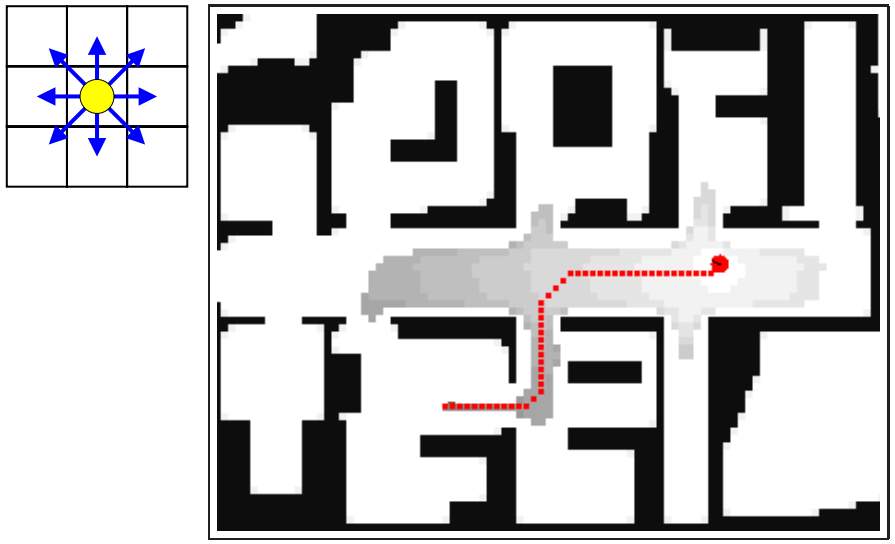
Konturen in A*

Innerhalb des Suchraums ergeben sich *Konturen*, in denen jeweils für gegebenen f-Wert alle Knoten expandiert werden:



Konturen für $f = 380, 400, 420$

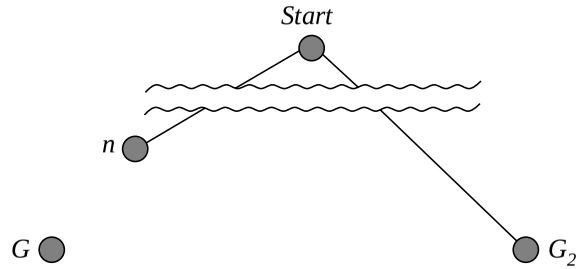
Beispiel: Pfadplanung für Roboter in einer Grid-Welt



Optimalität von A*

Beh.: Die erste von A* gefundene Lösung ist eine mit minimalen Pfadkosten.

Beweis: Wir nehmen an, dass es einen Zielknoten G mit optimalen Pfadkosten f^* gibt, dass A* aber einen anderen Knoten G_2 mit $g(G_2) > f^*$ gefunden hat.



Sei n ein Knoten auf dem optimalen Pfad vom Start nach G, der noch nicht expandiert wurde. Da h ja zulässig ist, haben wir

$$f(n) \leq f^*.$$

Da n nicht vor G_2 expandiert wurde, muss gelten

$$f(G_2) \leq f(n)$$

und somit

$$f(G_2) \leq f^*.$$

Wegen $h(G_2) = 0$ folgt daraus, dass

$$g(G_2) \leq f^*$$

$\rightsquigarrow$ Widerspruch zur Annahme!

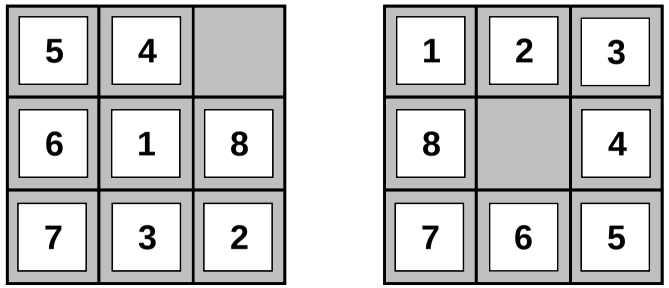
Vollständigkeit und Komplexität

Vollständigkeit: A* findet eine Lösung, falls es eine gibt, unter der Voraussetzung, dass (1) jeder Knoten nur *endlich* viele Nachfolgerknoten hat und (2) es eine positive Konstante δ gibt, so dass jeder Operator mindestens die Kosten δ hat.
$\rightsquigarrow$ nur endlich viele Knoten n mit $f(n) \leq f^*$.

Komplexität: Falls $|h^*(n) - h(n)| \leq O(\log(h^*(n))$, werden nur subexponentiell viele Knoten expandiert.

Normalerweise: Exponentielles Wachstum, weil der Fehler proportional zu den Pfadkosten ist.

Beispiel: Heuristische Funktion



Start State **Goal State**

$h_1 =$ Zahl der Kästchen in falscher Position

$h_2 =$ Summe der Distanzen der Kästchen zu ihrer Zielposition (*Manhattan-Distanz*)

Empirische Auswertung

- $d =$ Abstand vom Ziel
- Durchschnitt über 100 Instanzen

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Iterative A*-Tiefensuche: IDA*

Idee: Kombination von IDS und A*, d.h. es werden alle Knoten innerhalb einer *Kontur* abgesucht.

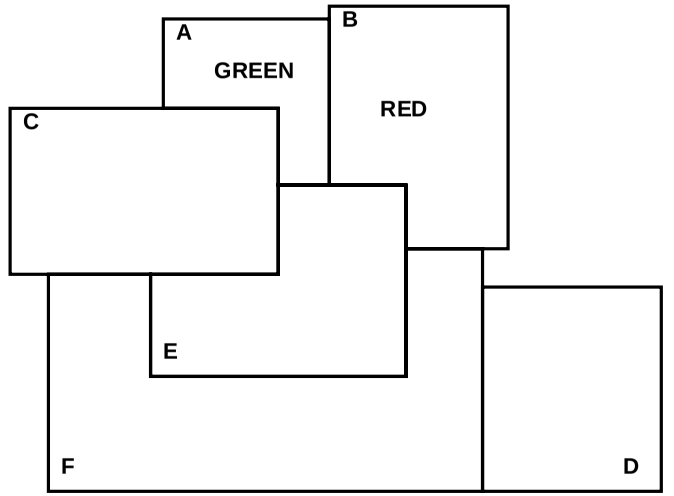
```
function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f-COST limit
          root, a node

  root ← MAKE-NODE(INITIAL-STATE[problem])
  f-limit ← f-COST(root)
  loop do
      solution, f-limit ← DFS-CONTOUR(root, f-limit)
      if solution is non-null then return solution
      if f-limit = ∞ then return failure; end
```

```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f-COST limit
  inputs: node, a node
          f-limit, the current f-COST limit
  static: next-f, the f-COST limit for the next contour, initially ∞

  if f-COST[node] > f-limit then return null, f-COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSORS(node) do
      solution, new-f ← DFS-CONTOUR(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← MIN(next-f, new-f); end
  return null, next-f
```


Heuristiken für CSPs



Welche Variablen instantiieren? Mit welchen Werten?

Übliche Heuristiken

Eingeschränkteste Variable zuerst:
$\rightsquigarrow$ reduziert den Verzweigungsfaktor!

Einschränkendste Variable zuerst (alternativ):
d.h. die Variable, die *Constraints* mit den meisten noch nicht belegten Variablen hat $\rightsquigarrow$ reduziert zukünftigen Verzweigungsfaktor

Den am wenigsten einschränkenden Wert zuerst:
$\rightsquigarrow$ erlaubt mehr Freiheiten bei zukünftigen Entscheidungen

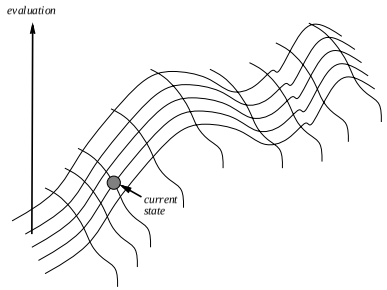
$\rightsquigarrow$ Lösen des 1000-Damen Problems!

Lokale Suche

Für viele Probleme ist es irrelevant, wie man zum Zielzustand kommt – nur der Zielzustand selber ist interessant (8-Damen Problem, VLSI Design, TSP).

Wenn sich außerdem ein Qualitätsmaß für Zustände angeben läßt, kann man **lokale Suche** benutzen, um Lösungen zu finden.

Idee: Man fängt mit einer zufällig gewählten Konfiguration an und verbessert diese schrittweise $\rightsquigarrow$ **Hill climbing**



Hill climbing

```
function HILL-CLIMBING( problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
          next, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
     next ← a highest-valued successor of current
     if VALUE[next] < VALUE[current] then return current
     current ← next
  end
```


Probleme bei lokaler Suche

- *Lokale Maxima:* Der Algorithmus gibt eine suboptimale Lösung aus.
- *Plateaus:* Hier kann der Algorithmus nur zufällig herumwandern.
- *Grate:* Ähnlich wie Plateaus.

Lösungen:

- *Neustarts*, wenn keine Verbesserung mehr
- *Rauschen* „injizieren“ (Random walk)
- *Tabu-Suche:* Die letzten n angewandten Operatoren nicht anwenden

Welche Strategien (mit welchen Parametern) erfolgreich sind (auf einer Problemklasse), kann man meist nur *empirisch* bestimmen.

Simuliertes Abkühlen

Im *Simulated-Annealing*-Algorithmus erfolgt das „Injizieren“ von Rauschen systematisch: Erst stark, dann abnehmend.

function SIMULATED-ANNEALING(*problem, schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to “temperature”
  static: *current*, a node
    next, a node
    T, a “temperature” controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for *t* ← 1 **to** ∞ **do**
    T ← *schedule*[*t*]
    if *T*=0 **then return** *current*
    next ← a randomly selected successor of *current*
    Δ*E* ← VALUE[*next*] – VALUE[*current*]
    if Δ*E* > 0 **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Wird seit den frühen ’80er für VLSI Layout und andere Optimierungsprobleme eingesetzt.

Anwendung auf CSPs

Obwohl bei CSPs Konfigurationen entweder Lösungen sind oder nicht, kann man auch hier lokale Suche anwenden.

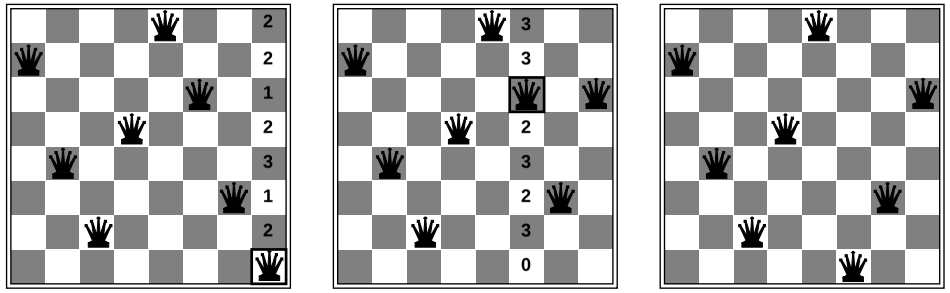
Qualitätsmaß: Anzahl der erfüllten Constraints. Lokale Suche heißt in diesem Kontext dann auch *heuristisches Reparieren*.

Heuristisches Reparieren wird z.B. für das Scheduling des Hubble-Teleskops eingesetzt. Reduzierung der Rechenzeit von 3 Wochen auf 10 Minuten.

Im Kontext vom Finden *erfüllender Belegungen* für boole’sche Formeln kann man ähnliche Methoden mit großem Erfolg einsetzen (siehe später).

Beispiel: 8-Damen Problem

Selektiere Spalte und bewege Dame auf das Feld mit den wenigsten Konflikten



Genetische Algorithmen

Die Evolution scheint ja sehr erfolgreich zu sein, gute Lösungen zu produzieren.

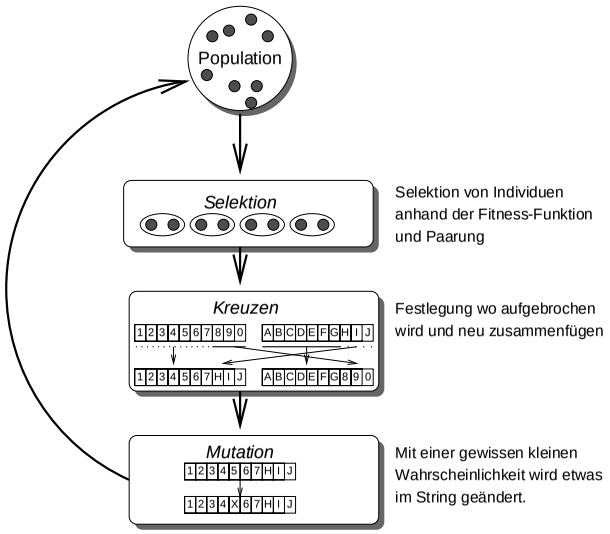
Idee:
Ähnlich wie bei der Evolution, sucht man Lösungen, indem man erfolgreiche Lösungen „kreuzt“, „mutiert“ und „selektiert“.

Ingredienzen:

- Codierung von Konfigurationen als Zeichenkette oder Bit-String.
- „Fitness“-Funktion, welche die Güte von Konfigurationen beurteilt.
- Population von Konfigurationen

Beispiel:
8-Damen Problem kodiert als Kette von 8 Zahlen.
Fitness berechnet sich aus der Anzahl der Nichtangriffe.
Population besteht aus einer Menge von Anordnungen der Damen

Selektieren, Kreuzen und Mutieren



Viele Variationen: Wie Selektion angewandt wird, welche Art des *Cross overs* benutzt wird, usw.

Wird z.B. erfolgreich für das Erstellen von Phantombildern eingesetzt. Fitness wird durch den Zeugen manuell bestimmt.

Zusammenfassung

- **Heuristiken** fokussieren die Suche.
- **Bestensuche** expandiert die (nach irgendeinem Maß) am besten bewerteten Knoten zuerst.
- Mit der Minimierung der geschätzten Kosten zum Ziel h erhalten wir **gierige Suche**.
- Minimierung von $f(n) = g(n) + h(n)$ kombiniert uniforme Kostensuche mit gieriger Suche. Falls $h(n)$ *zulässig* ist, d.h. h^* nie überschätzt, erhalten wir **A***-Suche, die vollständig und optimal ist.
- **IDA*** ist eine Kombination von iterativer Tiefensuche und A*.
- **Lokale Suche** arbeitet immer nur auf einem Zustand und versucht, diesen schrittweise zu verbessern.
- **Genetische Algorithmen** ahmen die Evolution nach, indem sie gute Lösungen miteinander kombinieren.

Grundlagen der KI

5. Brettspiele

Suchstrategien für Spiele, Spiele mit Zufall, Stand der Kunst

Wolfram Burgard

Inhalt

- Brettspiele
- Minimax-Suche
- Alpha-Beta-Suche
- Spiele mit Zufallsereignissen
- Stand der Kunst

Warum Brettspiele?

Brettspiele sind eines der ältesten Teilgebiete der KI
(Shannon und Turing schon 1950).

- Brettspiele stellen eine sehr abstrakte und pure Form des Wettbewerbs zwischen zwei Gegnern dar und erfordern „offensichtlich“ eine Form von Intelligenz.
- Die Zustände eines Spiels sind einfach darstellbar.
- Die möglichen Aktionen der Spieler sind wohldefiniert.

$\rightsquigarrow$ Realisierung des Spielens als Suchproblem
$\rightsquigarrow$ Die Weltzustände sind voll zugänglich
$\rightsquigarrow$ Allerdings handelt es sich um ein Kontingenzproblem, weil die Züge des Gegners im voraus nicht bekannt sind

Probleme

Brettspiele sind nicht nur schwer, weil es sich um Kontingenzprobleme handelt, sondern auch, weil die Suchbäume astronomisch groß werden.

Beispiele:

- Schach: durchschnittlich 35 mögliche Aktionen in jeder Position, 100 mögliche Halbzüge $\rightsquigarrow 35^{100}$ Knoten im Suchbaum (bei „nur“ ca. 10^{40} legalen Schachpositionen)
- Go: durchschnittlich 200 mögliche Aktionen bei ca. 300 Halbzügen $\rightsquigarrow 200^{300}$ Knoten.

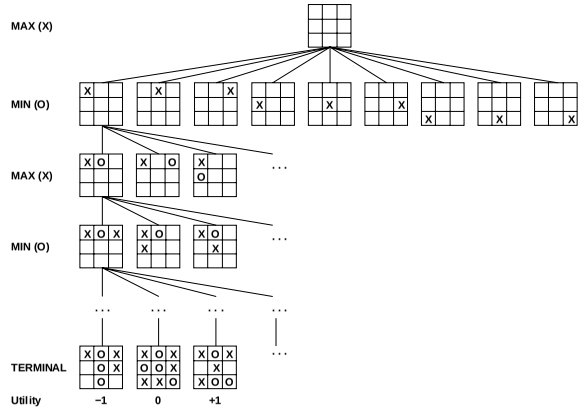
Gute Spielprogramme zeichnen sich dadurch aus, dass sie

- irrelevante Äste im Spielbaum abschneiden,
- gute Evaluierungsfunktion für Zwischenzustände benutzen und
- möglichst viele Halbzüge vorausschauen.

Terminologie bei Zweipersonen-Brettspielen

- **Spieler** *MAX* und *MIN*, wobei MAX beginnt.
- **Anfangszustand** (z.B. Brettposition)
- **Operatoren** (= legale Züge)
- **Terminierungstest**, der angibt, wann ein Spiel zu Ende ist.
  Terminalzustand = Spiel zu Ende.
- **Nutzenfunktion**, die den Ausgang eines Spiels für jeden Spieler bewertet. Oft einfach: $+1$ (Sieg), -1 (Niederlage), 0 (unentschieden). Bei Backgammon liegt der Wert zwischen $+192$ und -192.
- **Strategie**
  Im Gegensatz zu regulärer Suche, bei der ja ein Pfad vom Anfangszustand zum Zielzustand die Lösung ist, will MAX eine *Strategie* finden, die unabhängig von MINs Zügen zum Gewinn führt $\rightsquigarrow$ korrekte Reaktionen auf alle Züge von MIN

Beispiel Tic-Tac-Toe



Jede Stufe des Suchbaums, auch **Spielbaum** genannt, wird mit dem Namen des Spielers bezeichnet, der am Zug ist (MAX- und MIN-Stufen).

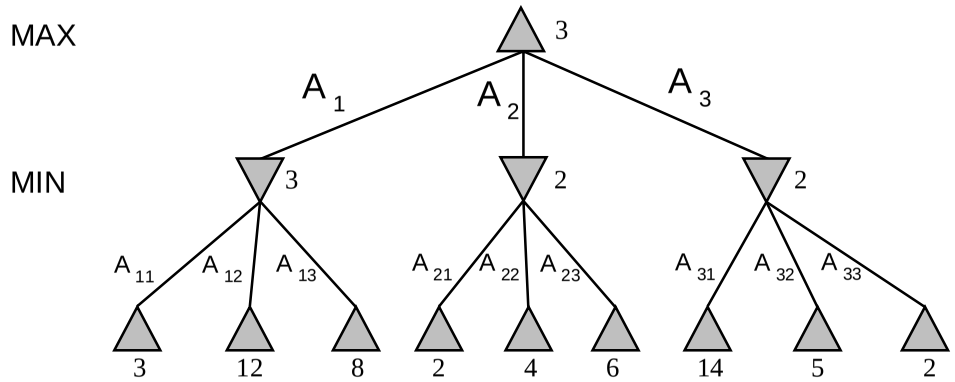
Wenn es wie hier möglich ist, den gesamten Suchbaum (Spielbaum) zu erzeugen, liefert der **Minimax-Algorithmus** eine optimale Strategie für MAX.

Minimax

1. Erzeuge vollständigen Spielbaum mit Tiefensuche.
2. Wende die Nutzenfunktion auf jeden Terminalzustand an.
3. Beginnend bei Terminalzuständen, berechne die Werte für die Vorgängerknoten wie folgt:
   - Knoten ist MIN-Knoten:
     Wert ist das **Minimum** der Nachfolgerknoten.
   - Knoten ist MAX-Knoten:
     Wert ist das **Maximum** der Nachfolgerknoten.
   - Dann wählt MAX im Anfangszustand (Wurzel des Spielbaums) den Zug, der zu dem Nachfolgerknoten mit größtem berechneten Nutzen führt (**Minimax-Entscheidung**).

Beachte: Minimax geht davon aus, dass MIN perfekt spielt. Jede Abweichung (d.h. jeder Fehler von MIN) kann das Ergebnis für MAX nur verbessern.

Beispiel für Minimax



Minimax-Algorithmus

Berechne rekursiv den besten Zug von der Anfangssituation ausgehend:

function MINIMAX-DECISION(*game*) **returns** *an operator*

  for each *op* **in** OPERATORS[*game*] **do**
    VALUE[*op*] ← MINIMAX-VALUE(APPLY(*op*, *game*), *game*)
  end
  return the *op* with the highest VALUE[*op*]

---

function MINIMAX-VALUE(*state*, *game*) **returns** *a utility value*

  if TERMINAL-TEST[*game*](*state*) **then**
    return UTILITY[*game*](*state*)
  else if MAX is to move in *state* **then**
    return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)

Beachte: Minimax funktioniert nur, wenn der Spielbaum nicht zu tief ist. Andernfalls muss man eine *Approximation* des Minimax-Wertes bestimmen.

Evaluierungsfunktion

Bei zu großem Suchraum, kann der Spielbaum nur bis zu einer gewissen Tiefe erzeugt werden. Die Kunst besteht darin, die Güte der Spielposition der Blätter korrekt zu bewerten.

Beispiel einfacher Bewertungskriterien im Schach:

- Materialwert: Bauer = 1, Springer = 3, Läufer = 5, Dame = 9.
- andere: Sicherheit des Königs, Positionierung der Bauern.
- *Faustregel*: 3-Punkte Vorsprung = sicherer Sieg.

Die Wahl der Evaluierungsfunktion ist spielentscheidend!

Wert einer Spielposition sollte die Gewinnchancen widerspiegeln, d.h. die Gewinnchancen beim einem Vorteil von +1 sollten geringer sein als bei einem Vorteil von +3.

Evaluierungsfunktion – allgemein

Bevorzugte Evaluierungsfunktionen sind *gewichtete lineare Funktionen:*

$$w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$$

Hierbei ist f_i das i-te Kriterium und w_i: i-tes Gewicht

[Bsp: $w_1 = 3$, f_1 = Zahl der eigenen Springer auf dem Brett]

Annahme: Die Kriterien sind unabhängig voneinander.

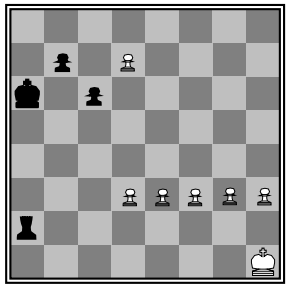
Die Gewichte können u.U. gelernt werden. Die Kriterien müssen allerdings vorgegeben werden (niemand weiß, wie man diese erlernen könnte).

Wann den Baum beschneiden?

- Tiefenbeschränkte Suche (so dass Zeitlimit nicht überschritten wird)
- Besser: Iterative Tiefensuche (mit Abbruch beim Erreichen des Zeitlimits)
- . . . aber nur „ruhige“ Positionen evaluieren, solche die nicht zu großen Schwankungen bei der Evaluierung in den folgenden Zügen führen.

$\leadsto$ D.h. u.U. noch etwas weiter suchen und einen Schlagabtausch zu Ende führen.

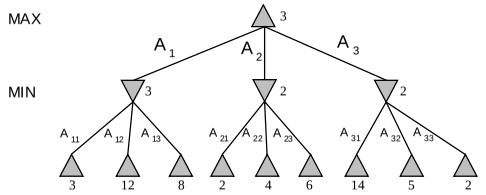
Horizontproblem



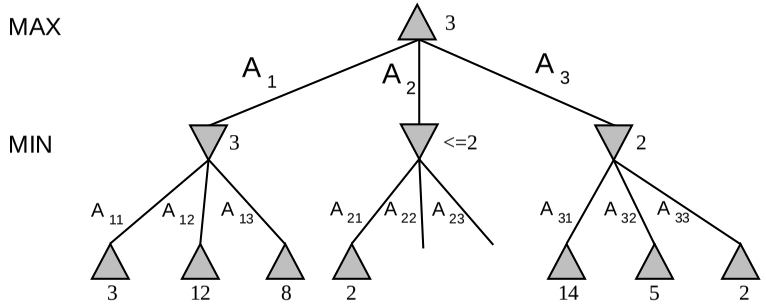
Black to move

- Schwarz hat zwar leichten Materialvorteil
- verliert aber unweigerlich (Bauer wird zur Dame)
- wird nicht erkannt bei fester Suchtiefe, da „Unglück“ herauszögerbar (jenseits des Horizonts - weil sich Schwarz auf das Schach mit dem Turm konzentriert, auf das Weiß reagieren muss)

Alpha-Beta Beschneidung: Beispiel



Wir brauchen nicht alle Knoten anschauen:



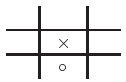
Alpha-Beta-Beschneidung in Tic-Tac-Toe (1)

zur Erinnerung: Minimax-Algorithmus mit Tiefe-Zuerst-Suche

α = bester Wert für MAX auf *einem* Pfad
β = bester Wert für MIN auf *einem* Pfad

Beispiel **Tic-Tac-Toe**:

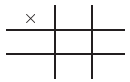
Mögliche Bewertung $e(p)$ eines Spielzustandes:
Anzahl der noch möglichen vollständigen Zeilen für MAX *minus* Anzahl der noch möglichen vollständigen Zeilen für MIN



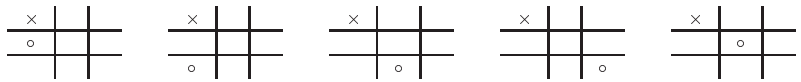
$e(p) = 6 - 4 = 2$

MAX kann noch in 2 Vertikalen, 2 Horizontalen und beiden Diagonalen einen Sieg erreichen (=6), MIN nur noch in der obersten/unteren Horizontalen und den äußeren Vertikalen (=4).

Berechnung unterer Schranken: Alpha-Wert

Erster Zug von MAX: 

Unter Ausnutzung der Symmetrie des Bretts kann MIN reagieren:

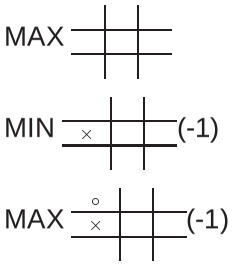


1 **0** **1** **0** **-1**

für den MAX Knoten ergibt sich eine untere Schranke, weil seine Bewertung in jedem Fall nur ≥ -1 sein wird $\rightsquigarrow \alpha = -1$.

α_{MAX} = größte aktuelle Bewertung seiner Nachfolger.

Berechnung oberer Schranken: Beta-Wert



Für den MIN Knoten ergibt sich aktuell eine beste mögliche Bewertung von -1. Da der Minimax Algorithmus für die MIN-Stufe das Minimum der Nachfolgerknoten wählt, kann jede folgende Bewertung bei Expandierung weiterer Zweige höchstens abnehmen. Es ergibt sich eine obere Schranke von $\beta = -1$.

Wann kann abgeschnitten werden?

es gilt:

α Werte von MAX Knoten können nie abnehmen

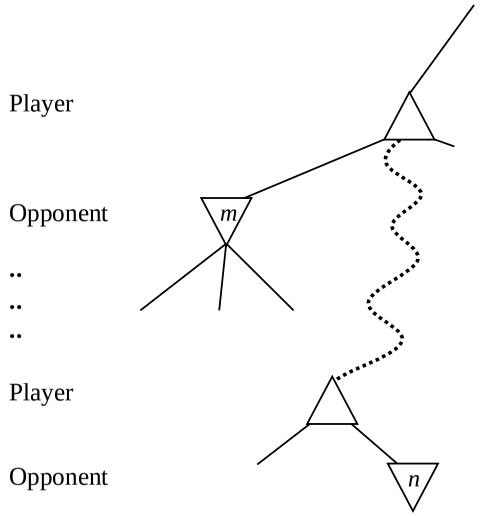
β Werte von MIN Knoten können nie zunehmen

(1) Abschneiden unterhalb des MIN Knotens dessen β-Schranke kleiner oder gleich der α-Schranke seines MAX-Vorgängerknotens ist.

(2) Abschneiden unterhalb des MAX Knotens dessen α-Schranke größer oder gleich der β-Schranke seines MIN-Vorgängerknotens ist.

$\rightsquigarrow$ liefert Ergebnisse, die genauso gut sind wie vollständige Mimimax-Suche bis zur gleichen Tiefe (weil nur irrelevante Zweige eliminiert werden).

Alpha-Beta Beschneidung: Allgemein



. . . falls $m > n$ werden wir im Spiel nie zum Knoten n gelangen

Alpha-Beta-Suchalgorithmus

function MAX-VALUE(*state*, *game*, α, β) **returns** the minimax value of *state*
  inputs: *state*, current state in game
    game, game description
    α, the best score for MAX along the path to *state*
    β, the best score for MIN along the path to *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\alpha \leftarrow$ MAX(α, MIN-VALUE(*s*, *game*, α, β))
    if $\alpha \geq \beta$ **then return** β
  end
  return α

---

function MIN-VALUE(*state*, *game*, α, β) **returns** the minimax value of *state*

  if CUTOFF-TEST(*state*) **then return** EVAL(*state*)
  for each *s* **in** SUCCESSORS(*state*) **do**
    $\beta \leftarrow$ MIN(β, MAX-VALUE(*s*, *game*, α, β))
    if $\beta \leq \alpha$ **then return** α
  end
  return β

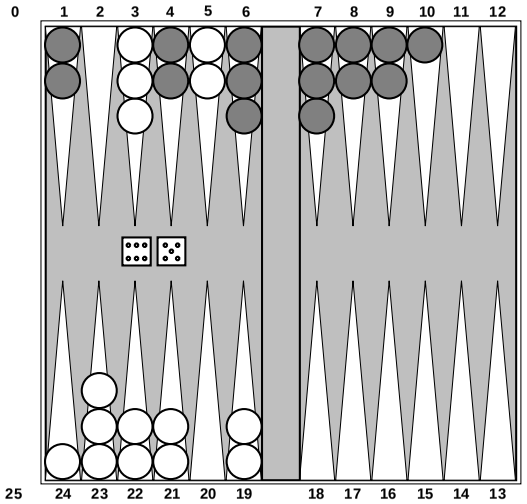
Initialer Aufruf mit MAX-VALUE(*initial-state*, $-\infty$, $+\infty$).

Effizienzgewinn

- Alpha-Beta-Suche schneidet am meisten ab, wenn jeweils der beste Zug als erstes betrachtet wird.
- **Bester Fall** (immer bester Zug zuerst) reduziert den Suchaufwand auf $O(b^{d/2})$.
- **Durchschnittlicher Fall** (zufällig verteilte Züge) reduziert den Suchaufwand auf $O((b/\log b)^d)$
- Für $b < 100$ erreichen wir $O(b^{3d/4})$.
- **Praktischer Fall**: Schon mit relativ einfachen Anordnungsheuristiken kommt man in die Nähe des besten Falls.
- D.h. wir können doppelt so tief in der gleichen Zeit suchen.

$\rightsquigarrow$ Bei Schach erreicht man damit eine Tiefe von 6-7 Halbzügen!

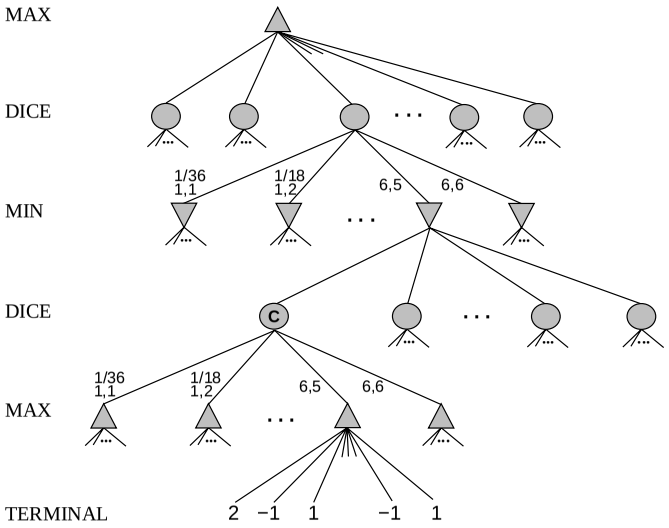
Spiele mit Zufallsereignissen



Weiß hat gerade 5/6 gewürfelt und hat 4 legale Züge.

Spielbaum für Backgammon

Zusätzlich zu MIN- und MAX-Knoten brauchen wir *Würfelknoten*.



Berechnung des Erwarteten Nutzens

Nutzenfunktion an Würfelknoten C über MAX:

d_i: mögliche Würfelergebnisse

$P(d_i)$: ihre Wahrscheinlichkeit

$S(C, d_i)$: Von C erreichbare Positionen beim Wurf d_i

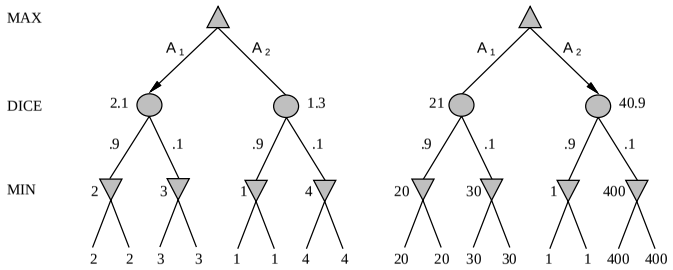
utility(s): Bewertung von s

$$\textit{expectimax}(C) = \sum_i P(d_i) \max_{s \in S(C,d_i)} (\textit{utility}(s))$$

expectimin entsprechend

Probleme

- Anordnungsinvariante Transformationen von Evaluierungsfunktionen erhalten nicht die Ordnung zwischen Zügen:



- Die Suchkosten steigen: Statt $O(b^d)$ haben wir $O(b \times n)^d$, wenn n die Anzahl möglicher Würfelergebnisse ist.

$\rightsquigarrow$ Bei Backgammon ($n = 21, b = 20$, manchmal sogar 4000) kann d maximal 2 werden.

Stand der Kunst

Dame (*Checkers, Draughts*, d.h. nach internationalen Regeln): Das Programm CHINOOK ist amtierender Weltmeister im Mensch-Maschine-Vergleich (anerkannt von ACF und EDA) und höchst bewerteter Spieler:

CHINOOK: 2712 R on King: 2632
A sa Long: 2631 D on Lafferty: 2625

Backgammon: Das Programm BKG schlug den amtierenden Weltmeister 1980. Ein neueres Programm ist unter den ersten 3 Spielern.

Reversi (*Othello*): Sehr gut auch auf normalen Computern. Programme werden bei Turnieren nicht zugelassen.

Go: Die besten Programme spielen etwas besser als Anfänger (Verzweigungsfaktor > 300). $\rightsquigarrow$ Für das erste GO Programm, das einen Großmeister schlägt, ist ein Preis von 2 Mio. US-$ ausgelobt.

Schach (1)

Schach als „Drosophila“ der KI-Forschung

- begrenzte Anzahl von Regeln bringt unbegrenzte Zahl von Spielverläufen hervor. Für eine Partie von 40 Zügen gibt es 1.5×10^{128} mögliche Spielverläufe.
- Sieg durch Logik, Intuition, Kreativität, Vorwissen
- nur spezielle Schachintelligenz, keine “Alltagsintelligenz”

Spielstärken:

G. Kasparow	2828
V. Anand	2758
A. Karpow	2710
Deep Blue	2680

Es gibt zur Zeit nur ca. 64 Super-Großmeister, die oberhalb von 2600 ELO Punkten spielen.

Schach (2)

1997 wird der amtierende Weltmeister G. Kasparow erstmals in einem Spiel aus 6 Partien von einem Computer geschlagen!

Deep Blue (IBM Thomas J. Watson Research Center)

- spezielle Hardware (32 Rechner mit 8 Chips, 2 Mio Berechnungen pro Sekunde)
- heuristische Suche
- fallbasiertes Schließen + Lerntechniken
  - 1996 Wissen aus 600 000 Schachpartien
  - 1997 Wissen aus 2 Mio Schachpartien
  - Training durch Großmeister

Duell „Maschinenmensch Kasparow - Menschmaschine Deep Blue“

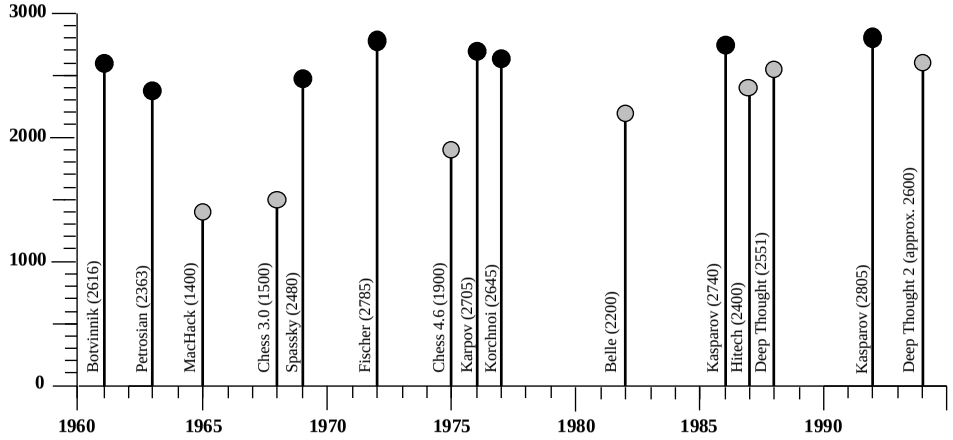
Schach (3)

Partie	1996	1997
1	DB gewinnt	DB gewinnt
2	Remis	DB gewinnt (K. gibt auf)
3	Remis	Remis
4	DB verliert	Remis
5	DB verliert	Remis
6	DB verliert	DB gewinnt (K. gibt auf)

Schach (4)

- Der Verzweigungsfaktor bei Schach ist ca. 35.
- Im Turnierschach stehen uns 150 Sekunden pro Zug zur Verfügung.
- „normale“ Schachcomputer expandieren/evaluieren ca. 1000 Positionen pro Sekunde.
  $\leadsto$ 150.000 Positionen durchsuchbar: 3 Halbzüge.
- Deep Blue 1996: Horizont von 3 Zügen / 15 Halbzügen.
- Deep Blue 1997: Horizont von 7 Zügen (50-100 Mrd. Stellungen in 3 Minuten)

Schach (5)



Schach (6)

Kasparow: *Es gab Momente, da beschlich mich ein Gefühl, dass diese Kisten womöglich näher an der Intelligenz sind, als wir wahrhaben wollen.*

Von einem bestimmten Punkt an scheint, zumindest im Schach, immense Quantität in Qualität umzuschlagen.

Doch ich sehe vielmehr die gewaltige Chance: Feine Kreativität und brutale Rechenkapazität könnten sich zu einer neuen Art der Informationsgewinnung ergänzen. Menschen und Elektronengehirn würden sich zu einer neuen Qualität von Intelligenz ergänzen, zu einer Intelligenz, die diesen Namen womöglich erst verdient.

Der Grund für die Erfolge ...

- Alpha-Beta-Suche
- ... mit dynamischer Tiefenfestlegung bei unsicheren Positionen
- gute (aber normalerweise einfache) Evaluierungsfunktionen
- große Eröffnungsdatenbanken
- sehr große Endspieldatenbanken (für Dame: alle 8-Steine Situationen)
- und sehr schnelle und parallele Rechner!

Zusammenfassung

- Ein **Spiel** kann durch Angabe des *Anfangszustands*, der *Operatoren* (legale Züge), eines *Terminaltests* und einer *Nutzenfunktion* (Ausgang des Spiels) beschrieben werden.
- In 2-Personen-Brettspielen kann der **Minimax-Algorithmus** den besten Zug bestimmen, indem er den ganzen Spielbaum aufbaut.
- Der **Alpha-Beta-Algorithmus** liefert das gleiche Ergebnis, ist jedoch effizienter, da er überflüssige Zweige abschneidet.
- Normalerweise kann nicht der ganze Spielbaum aufgebaut werden, so dass man Zwischenzustände mit einer **Evaluierungsfunktion** bewerten muss.
- Spiele mit Zufallselement kann man mit einer Erweiterung des Alpha-Beta-Algorithmus behandeln.

Grundlagen der KI

6. Aussagenlogik

Rationales Denken, Logik, Resolution

Wolfram Burgard

Inhalt

- Rational denkende Agenten
- Die Wumpus-Welt
- Aussagenlogik: Syntax & Semantik
- Logische Folgerbarkeit
- Logische Ableitungen: Resolution

Rational denkende Agenten

- Bisher lag das Schwergewicht auf *rational handelnden* Agenten.
- Oft setzt rationales Handeln aber *rationales* (logisches) *Denken* durch den Agenten selbst voraus.
- Dazu müssen Teile der Welt symbolisch in einer *Wissensbasis* (*knowledge base* oder *KB*) repräsentiert sein:
  - KB enthält Sätze in einer Sprache mit einer Wahrheitstheorie (Logik), d.h. wir (als Außenstehende) können Sätze als *Aussagen* über die Welt *interpretieren*. (**Semantik**)
  - Die Sätze haben allein durch ihre *Form* einen *kausalen Einfluss* auf das Verhalten des Agenten in einer Weise, die in Korrelation zum Inhalt der Sätze steht. (**Syntax**)
- Interaktion mit der KB durch **ASK** und **TELL** (vereinfacht):

ASK(KB,α) = JA	gdw. α folgt aus der KB
TELL(KB,α) = KB',	so dass α aus KB' folgt.
FORGET(KB,α) = KB',	nicht monoton (wird nicht behandelt)

3 Ebenen

Im Kontext der *Wissensrepräsentation* unterscheidet man 3 Ebenen [Newell 1990]:

Wissensebene: abstrakteste Ebene; umfasst das gesamte Wissen, dass in der KB steckt; z.B. automatische DB-Auskunft weiß, dass die eine Fahrt Ulm-Freiburg DM 88,– kostet.

Symbolische Ebene: Kodierung des Wissens in einer formalen Sprache: *Preis(Ulm, Freiburg, 88.00)*

Implementierungsebene: Die interne Darstellung der Sätze, z.B.:

- als String „`Preis(Ulm,Freiburg,88.00)`“
- als Wert in einer Matrix.

Wenn ASK und TELL korrekt funktionieren, reicht es, sich auf der Wissensebene zu bewegen. Vorteil: sehr komfortable Benutzerschnittstelle. Benutzer hat selbst mentales Weltmodell (Aussage über die Welt) und teilt dies einfach dem Agenten mit (TELL).

Ein wissensbasierter Agent

Ein wissensbasierter Agent benutzt seine Wissensbasis um

- sein Hintergrundwissen zu repräsentieren
- seine Beobachtungen zu speichern
- und seine durchgeführten Aktionen zu speichern
- . . . um daraus Handlungen abzuleiten

```
function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

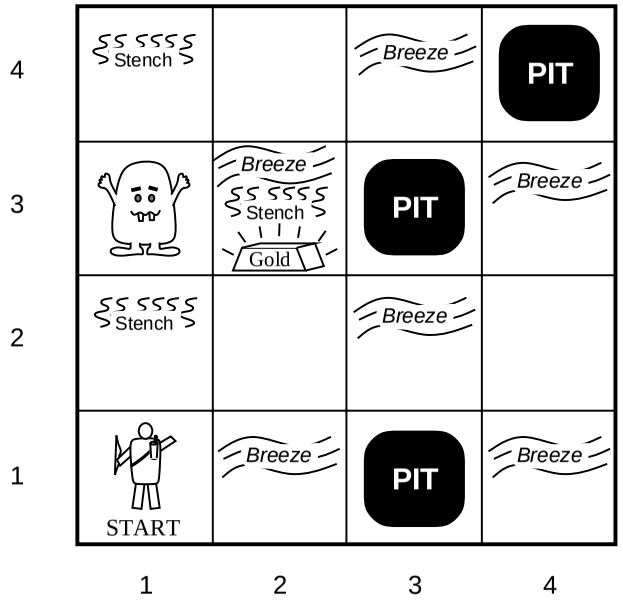

Die Wumpus-Welt (1)

- Ein 4×4 Feld.
- Im Kästchen, in dem der Wumpus sich befindet, und in den direkten Nachbarkästchen nimmt man einen üblen Geruch wahr (Stench).
- In den Kästchen neben einer Fallgrube (Pit) nimmt man einen Luftzug wahr (Breeze).
- Im Kästchen mit dem Gold glitzert es (Glitter).
- Wenn der Agent in eine Wand läuft, bekommt er einen Schlag.
- Wenn der Wumpus getötet ist, hört man es überall. (Todesschrei)
- Wahrnehmungen werden als 5-Tupel dargestellt. Z.B. [Stench, Breeze, Glitter, None, None] bedeutet, dass es stinkt, zieht und glitzert, aber es gab weder einen Schlag noch einen Todesschrei. Der Agent kann seinen Standort nicht wahrnehmen!

Die Wumpus-Welt (2)

- Aktionen: gehe vorwärts, 90^o nach rechts drehen, 90^o nach links drehen, greife ein Objekt im selben Kästchen, schieße (es gibt nur einen Pfeil), verlasse die Höhle (funktioniert nur in Kästchen [1,1]).
- Der Agent stirbt, wenn er in eine Fallgrube fällt oder dem lebenden Wumpus begegnet.
- Anfangszustand: Agent in [1,1] nach Osten schauend, irgendwo 1 Wumpus, 1 Haufen Gold und 3 Fallgruben.
- **Ziel**: Hole das Gold und verlasse die Höhle.

Die Wumpus-Welt (3): Eine Beispielkonfiguration



Die Wumpus-Welt (4)

$[1, 2]$ und $[2, 1]$ sind sicher:

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

(a)

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2 P?	3,2	4,2
1,1 V OK	2,1 A B OK	3,1 P?	4,1

(b)

Die Wumpus-Welt (5)

Der Wumpus ist in $[1, 3]$!

1,4	2,4	3,4	4,4
1,3 W!	2,3	3,3	4,3
1,2 A S OK	2,2 OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(a)

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4 P?	3,4	4,4
1,3 W!	2,3 A S G B	3,3 P?	4,3
1,2 S V OK	2,2 V OK	3,2	4,2
1,1 V OK	2,1 B V OK	3,1 P!	4,1

(b)

Deklarative Sprachen

Bevor man ein System konstruiert, das lernen, denken, planen, erklären, ... kann, muss man in der Lage, sein Wissen **auszudrücken**.

Wir brauchen eine präzise, deklarative Sprache.

- **deklarativ**: System glaubt P gdw. es P für **wahr** hält (man kann P nicht glauben ohne eine Vorstellung, was es bedeutet, dass die Welt P erfüllt).
- **präzise**: Wir müssen wissen,
  - welche Zeichenketten als *Sätze* gelten,
  - was es bedeutet, dass ein Satz *wahr* ist und
  - wann ein Satz aus anderen Sätzen *folgt*.

Eine Möglichkeit: **Aussagenlogik**

Grundideen der Aussagenlogik

Aussagen: Die Grundbausteine der Aussagenlogik sind nicht weiter zerlegbare, atomare Aussagen (atomare Propositionen), wie z.B.

- „Der Block ist rot“
- „Der Wumpus ist in [1,3]“

und logische Konnektoren „und“, „oder“, „nicht“, mit denen wir **Formeln** aufbauen können.

Uns interessiert:

- Wann ist eine Aussage **wahr**?
- Wann **folgt** eine Aussage aus einer Wissensbasis KB, symbolisch: $\boxed{KB \models \varphi}$?
- Können wir einen (syntaktischen) *Ableitungsbegriff*, symbolisch $\boxed{KB \vdash \varphi}$, so definieren, dass er mit dem *Folgerungsbegriff* übereinstimmt?

$\rightsquigarrow$ Bedeutung und Implementierung von ASK

Syntax der Aussagenlogik

Abzählbares Alphabet Σ von **atomaren Aussagen**: $P, Q, R, \ldots$.

Aussagenlogische Formeln:

$$\begin{array}{rcll} \varphi, \psi & \longrightarrow & P & \textit{atomare Formel} \\ & | & \bot & \textit{Falschheit} \\ & | & \top & \textit{Wahrheit} \\ & | & \neg\varphi & \textit{Negation} \\ & | & \varphi \wedge \psi & \textit{Konjunktion} \\ & | & \varphi \vee \psi & \textit{Disjunktion} \\ & | & \varphi \Rightarrow \psi & \textit{Implikation} \\ & | & \varphi \Leftrightarrow \psi & \textit{Äquivalenz} \end{array}$$

Operatorpräzedens: $\neg > \wedge > \vee > \Rightarrow = \Leftrightarrow$. (Ggf. muss geklammert werden.)
Atom: atomare Formel
Literal: (u.U. negierte) atomare Formel
Klausel: Disjunktion von Literalen

Semantik: Intuition

Atomare Aussagen können *wahr* (T oder *true*) oder *falsch* (F oder *false*) sein.

Der Wahrheitswert von Formeln ergibt sich aus den Wahrheitswerten der Atome (Wahrheitsbelegung oder Interpretation).

Beispiel:

$$(P \vee Q) \wedge R$$

- Wenn P und Q falsch sind und R wahr ist, ist die Formel nicht wahr.
- Wenn dagegen P und R wahr sind (Q egal), ist die Formel wahr.

Semantik – formal

Eine **Wahrheitsbelegung** der Atome in Σ oder **Interpretation** über Σ ist eine Funktion $\mathcal{I}$:

$$\mathcal{I} \colon \Sigma \Rightarrow \{\mathtt{T}, \mathtt{F}\}.$$

Interpretation $\mathcal{I}(\varphi)$ bzw. $\varphi^{\mathcal{I}}$ einer Formel φ:

$$\begin{array}{rll} \mathcal{I} \models \top & & \\ \mathcal{I} \not\models \bot & & \\ \mathcal{I} \models P & \text{gdw.} & P^{\mathcal{I}} = \mathtt{T} \\ \mathcal{I} \models \neg\varphi & \text{gdw.} & \mathcal{I} \not\models \varphi \\ \mathcal{I} \models \varphi \wedge \psi & \text{gdw.} & \mathcal{I} \models \varphi \text{ und } \mathcal{I} \models \psi \\ \mathcal{I} \models \varphi \vee \psi & \text{gdw.} & \mathcal{I} \models \varphi \text{ oder } \mathcal{I} \models \psi \\ \mathcal{I} \models \varphi \Rightarrow \psi & \text{gdw.} & \text{wenn } \mathcal{I} \models \varphi, \text{ dann } \mathcal{I} \models \psi \\ \mathcal{I} \models \varphi \Leftrightarrow \psi & \text{gdw.} & \mathcal{I} \models \varphi, \text{ genau dann, wenn } \mathcal{I} \models \psi \end{array}$$

$\mathcal{I}$ *erfüllt* φ ($\mathcal{I} \models \varphi$) oder φ ist *wahr* unter $\mathcal{I}$, falls $\mathcal{I}(\varphi) = \mathtt{T}$.

Beispiel

$$\mathcal{I} \colon \begin{cases} P & \mapsto & \mathtt{T} \\ Q & \mapsto & \mathtt{F} \\ R & \mapsto & \mathtt{F} \\ S & \mapsto & \mathtt{T} \\ & \vdots & \end{cases}$$

$$\varphi = ((P \vee Q) \Leftrightarrow (R \vee S)) \wedge (\neg(P \wedge Q) \vee (R \wedge \neg S)).$$

Frage: $\mathcal{I} \models \varphi$?

Terminologie

Eine Interpretation $\mathcal{I}$ heißt **Modell** von φ, falls $\mathcal{I} \models \varphi$

Eine Interpretation ist Modell einer *Menge von Formeln*, falls sie alle Formeln der Menge erfüllt.

Eine Formel φ heißt

- **erfüllbar**, wenn es $\mathcal{I}$ gibt, die φ erfüllt,
- **unerfüllbar**, wenn φ nicht erfüllbar ist,
- **falsifizierbar**, wenn es $\mathcal{I}$ gibt, die φ nicht erfüllt, und
- **allgemeingültig** (**Tautologie**), wenn für alle $\mathcal{I}$ gilt, dass $\mathcal{I} \models \varphi$.

Zwei Formeln heißen

- **logisch äquivalent** ($\varphi \equiv \psi$), wenn für alle $\mathcal{I}$ gilt, dass $\mathcal{I} \models \varphi$ gdw. $\mathcal{I} \models \psi$.

Die Wahrheitstabellenmethode

Wie entscheiden wir, ob eine Formel erfüllbar, allgemeingültig usw. ist?

$\rightsquigarrow$ Wahrheitstabelle aufstellen

Beispiel: Ist $\varphi = ((P \vee H) \wedge \neg H) \Rightarrow P$ allgemeingültig?

P	H	$P \vee H$	$(P \vee H) \wedge \neg H$	$((P \vee H) \wedge \neg H) \Rightarrow P$
F	F	F	F	T
F	T	T	F	T
T	F	T	T	T
T	T	T	F	T

Da die Formel unter allen möglichen Wahrheitsbelegungen wahr ist (von allen Interpretationen erfüllt wird), ist φ allgemeingültig.

Erfüllbarkeit, Falsifizierbarkeit, Unerfüllbarkeit entsprechend.

Normalformen

Eine Formel ist in **konjunktiver Normalform** (KNF), wenn die Formel aus einer Konjunktion von Disjunktionen von Literalen $l_{i,j}$ besteht, d.h. wenn sie die folgende Gestalt hat:

$$\bigwedge_{i=1}^{n} (\bigvee_{j=1}^{m_i} l_{i,j}).$$

Eine Formel ist in **disjunktiver Normalform** (DNF), wenn die Formel eine Disjunktion von Konjunktionen von Literalen ist:

$$\bigvee_{i=1}^{n} (\bigwedge_{j=1}^{m_i} l_{i,j}).$$

Zu jeder Formel existiert mindestens eine äquivalente Formel in KNF und eine in DNF.

Eine Formel in DNF ist erfüllbar gdw. ein Disjunkt erfüllbar ist.

Eine Formel in KNF ist allgemeingültig gdw. wenn jedes Konjunkt allgemeingültig ist.

Erzeugen der KNF

1. Eliminiere $\Rightarrow$ und $\Leftrightarrow$: $\alpha \Rightarrow \beta \rightsquigarrow (\neg\alpha \vee \beta)$ usw.
2. Schiebe $\neg$ nach innen: $\neg(\alpha \wedge \beta) \rightsquigarrow (\neg\alpha \vee \neg\beta)$ usw.
3. Verteile: $\vee$ über $\wedge$ $((\alpha \wedge \beta) \vee \gamma) \rightsquigarrow ((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$
4. Vereinfache: $(\alpha \vee \alpha) \rightsquigarrow \alpha$ usw.

Ergebnis ist eine Konjunktion von Disjunktionen von Literalen

- Ein analoges Verfahren überführt eine beliebige Formel in eine äquivalente Formel in DNF.
- Formeln können bei der Transformation exponentiell aufgebläht werden.

Folgerbarkeit: Intuition

Eine Menge von Formeln (eine KB) beschreibt die Welt u.U. nur unvollständig, d.h. lässt die Wahrheitswerte einiger Aussagen offen.

Beispiel: $KB = \{P \vee Q, R \vee \neg P, S\}$

Ist definitiv bzgl. S, lässt aber P, Q, R offen (allerdings nicht beliebig).

Modelle von KB:

P	Q	R	S
F	T	F	T
F	T	T	T
T	F	T	T
T	T	T	T

In allen Modellen von KB ist $Q \vee R$ wahr, d.h. $\boxed{Q \vee R}$ *folgt logisch* aus KB.

Logische Folgerbarkeit – formal

Die Formel φ folgt aus KB, wenn φ in in allen Modellen von KB wahr ist (symbolisch $KB \models \varphi$):

$$KB \models \varphi \quad \text{gdw.} \quad \mathcal{I} \models \varphi \text{ für alle Modelle } \mathcal{I} \text{ von } KB$$

Beachte: Auch hier ist $\models$ wieder ein Metasymbol; diesmal werden andere Objekte in Beziehung zueinander gesetzt!

Einige Eigenschaften der Folgerungsbeziehung:

- Deduktionssatz: $KB \cup \{\varphi\} \models \psi$ gdw. $KB \models \varphi \Rightarrow \psi$
- Kontrapositionssatz: $KB \cup \{\varphi\} \models \neg\psi$ gdw. $KB \cup \{\psi\} \models \neg\varphi$
- Widerspruchssatz: $KB \cup \{\varphi\}$ ist unerfüllbar gdw. $KB \models \neg\varphi$

Frage: Können wir $KB \models \varphi$ entscheiden, ohne alle Interpretationen betrachten zu müssen (die Wahrheitstabellenmethode)?

Beweis des Deduktionssatzes

„⇒“ Annahme: $KB \cup \{\varphi\} \models \psi$, d.h. jedes Modell von $KB \cup \{\varphi\}$ ist auch Modell von ψ.

Sei I ein beliebiges Modell von KB. Ist I auch Modell von φ dann folgt daraus, dass I auch Modell von ψ.

Damit ist aber auch I ein Modell von $\varphi \Rightarrow \psi$, d.h. $KB \models \varphi \Rightarrow \psi$.

„⇐“ Annahme $KB \models \varphi \Rightarrow \psi$. Sei I ein beliebiges Modell von KB, welches auch Modell von φ, d.h. $I \models KB \cup \{\varphi\}$.

Aufgrund der Annahme ist I auch Modell von $\varphi \Rightarrow \psi$ und damit auch von ψ, d.h. $KB \cup \{\varphi\} \models \psi$.

Beweis des Kontrapositionssatzes

$$\begin{aligned}
& KB \cup \{\varphi\} \models \neg\psi \\
\Leftrightarrow \quad & KB \models \varphi \Rightarrow \neg\psi && (1) \\
\Leftrightarrow \quad & KB \models (\neg\varphi \vee \neg\psi) \\
\Leftrightarrow \quad & KB \models (\neg\psi \vee \neg\varphi) \\
\Leftrightarrow \quad & KB \models \psi \Rightarrow \neg\varphi \\
\Leftrightarrow \quad & KB \cup \{\psi\} \models \neg\varphi && (2)
\end{aligned}$$

Hinweis: (1) und (2) sind jeweils Anwendungen des Deduktionssatzes.

Inferenzregeln, Kalküle und Beweise

Oft können wir aus Formeln in der KB weitere Formeln erzeugen (oder *ableiten*), die auf Grund der syntaktischen Struktur der KB-Formeln *logisch folgen müssen*.

Beispiel: Falls $KB = \{\ldots, (\varphi \Rightarrow \psi), \ldots, \varphi, \ldots\}$, dann gilt immer $KB \models \psi$.

$\rightsquigarrow$ **Inferenzregeln**, z.B. $\frac{\varphi, \varphi \Rightarrow \psi}{\psi}$

Kalkül: Menge von Inferenzregeln (und u.U. sogenannten logischen Axiomen)

Beweisschritt: Anwendung einer Inferenzregel auf eine Menge von Formeln.

Beweis: Sequenz von Beweisschritten, wobei die jeweils neu abgeleiteten Formeln hinzugenommen werden und im letzten Schritt die *Zielformel* erzeugt wird.

Korrektheit und Vollständigkeit

Falls es im Kalkül C einen Beweis für eine Formel $\varphi \in KB$ gibt, schreiben wir

$$KB \vdash_C \varphi$$

(u.U. ohne Subskript C).

Ein Kalkül C heißt **korrekt**, wenn alle aus einer KB ableitbaren Formeln auch tatsächlich logisch folgen:

$$KB \vdash_C \varphi \text{ impliziert } KB \models \varphi.$$

Folgt normalerweise aus der Korrektheit der Inferenzregeln und der logischen Axiome.

Ein Kalkül heißt **vollständig**, falls jede Formel, die logisch aus KB folgt, auch aus KB ableitbar ist:

$$KB \models \varphi \text{ impliziert } KB \vdash_C \varphi.$$

Resolution: Idee

Wir wollen jetzt ein *Ableitungsverfahren* studieren, das nicht darauf beruht, dass man alle Interpretationen durchprobiert.

Idee: Man versucht zu zeigen, dass eine Formelmenge unerfüllbar ist.

Allerdings: Es wird vorausgesetzt, dass alle Formeln in KNF vorliegen.

Aber: In den meisten Fällen sind die Formeln nahe an KNF (und es ex. eine schnelle erfüllbarkeitserhaltende Transformation)

Nichtsdestotrotz: Auch dieses Ableitungsverfahren benötigt im *schlechtesten Fall* exponentiell viel Zeit (das lässt sich aber vermutlich nicht vermeiden).

Resolution: Repräsentation

Annahme: Alle Formeln in der Formelmenge KB liegen in KNF vor.

Äquivalent können wir annehmen, dass KB eine Menge von Klauseln ist.

Wegen der Kommutativität, Assoziativität und Idempotenz von $\vee$ können wir Klauseln auch als **Mengen von Literalen** auffassen. Die **leere Menge von Literalen** wird als $\square$ geschrieben.

Menge von Klauseln:	Δ
Menge von Literalen:	C, D
Literale:	l
Negation des Literals:	$\overline{l}$.

Eine Interpretation $\mathcal{I}$ erfüllt C gdw. es ein $l \in C$ gibt, so dass $\mathcal{I} \models l$. $\mathcal{I}$ erfüllt Δ falls für alle $C \in \Delta$: $\mathcal{I} \models C$. D.h. $\mathcal{I} \not\models \square$, $\mathcal{I} \not\models \{\square\}$, $\mathcal{I} \models \{\,\}$, für alle $\mathcal{I}$.

Die Resolutionsregel

$$\frac{C_1 \cup \{l\}, C_2 \cup \{\bar{l}\}}{C_1 \cup C_2}$$

$C_1 \cup C_2$ wird **Resolvente** der **Elternklauseln** $C_1 \cup \{l\}$ und $C_2 \cup \{\bar{l}\}$ genannt. l und $\bar{l}$ sind die **Resolutionsliterale**.

Beispiel: $\{a, b, \neg c\}$ resolviert mit $\{a, d, c\}$ zu $\{a, b, d\}$.

Beachte: Die Resolvente ist **nicht** äquivalent mit den Elternklauseln, folgt aber aus diesen!

Notation: $R(\Delta) = \Delta \cup \{C \mid C \text{ ist Resolvente zweier Klauseln aus } \Delta\}$

Ableitungen

Wir sagen D kann aus Δ mit Hilfe von Resolution **abgeleitet** werden, d.h.

$$\Delta \vdash D,$$

wenn es $C_1, C_2, \ldots, C_n = D$ gibt, so dass

$$C_i \in R(\Delta \cup \{C_1, \ldots, C_{i-1}\}), \text{ für } 1 \leq i \leq n.$$

Lemma (*Korrektheit*) Wenn $\Delta \vdash D$, dann $\Delta \models D$.

Beweisskizze: Da alle $D \in R(\Delta)$ aus Δ logisch folgen, ergibt sich das Lemma durch Induktion über die Länge der Ableitung.

Vollständigkeit?

Ist Resolution auch vollständig? D.h. gilt

$$\Delta \models \varphi \text{ impliziert } \Delta \vdash \varphi?$$

Höchstens für Klauseln. Aber:

$$\left\{\{a, b\}, \{\neg b, c\}\right\} \models \{a, b, c\}$$
$$\not\vdash \{a, b, c\}$$

Aber man kann zeigen, dass Resolution **widerlegungsvollständig** ist:

$$\Delta \text{ unerfüllbar impliziert } \Delta \vdash \square.$$

Satz Δ ist unerfüllbar gdw. $\Delta \vdash \square$.

D.h. wir können mit Hilfe des Widerspruchtheorems zeigen, dass $KB \models \varphi$.

Resolution: Ausblick

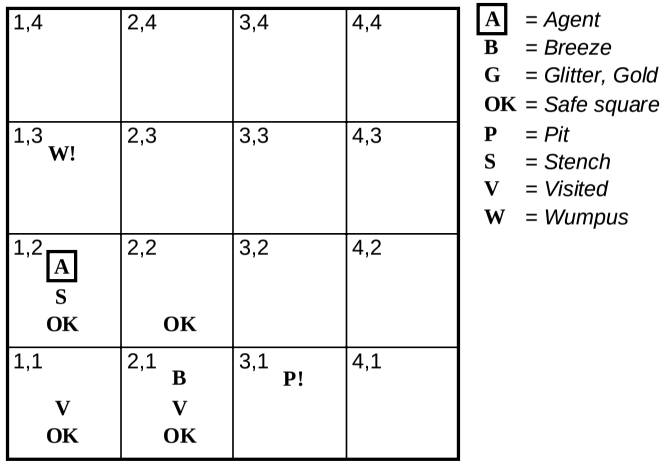
Resolution ist *ein* vollständiges Beweisverfahren. Es gibt noch andere (Davis-Putnam Prozedur, Tableaux-Verfahren, ...).

Für eine Implementation des Verfahrens muss man noch eine *Strategie* festlegen, die angibt, wann welche Resolutionsschritte ausgeführt werden.

Im schlechtesten Fall kann ein Resolutionsbeweis exponentielle Länge haben. Allerdings gilt dies höchstwahrscheinlich auch für alle anderen Beweisverfahren.

Für KNF-Formeln in propositionaler Logik ist die Davis-Putnam Prozedur (Backtracking über alle Wahrheitsbelegungen) das in der Praxis „schnellste“ vollständige Verfahren, das auch als eine Art von Resolutionsverfahren aufgefasst werden kann.

Wo ist der Wumpus: Die Situation



A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

Wo ist der Wumpus: Wissen über die Situation

B = Breeze, S = Stench, $B_{i,j}$ = es zieht im Kästchen (i, j)

$$\begin{array}{ll} \neg S_{1,1} & \neg B_{1,1} \\ \neg S_{2,1} & B_{2,1} \\ S_{1,2} & \neg B_{1,2} \end{array}$$

Invariantes Wissen über Wumpus und Geruch:

$$R_1: \quad \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}$$
$$R_2: \quad \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}$$
$$R_3: \quad \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}$$

$$R_4: \quad S_{1,2} \Rightarrow W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$$
$$\ldots$$

Zu zeigen: $KB \models W_{1,3}$

Klauseldarstellung der Wumpus-Welt

Situationswissen:
$\neg S_{1,1}, \neg S_{2,1}, S_{1,2}, \ldots$

Regelwissen:

$$R_1: \quad S_{1,1} \vee \neg W_{1,1},\ S_{1,1} \vee \neg W_{1,2},\ S_{1,1} \vee \neg W_{2,1}$$
$$R_2: \quad \ldots, S_{2,1} \vee \neg W_{2,2}, \ldots$$
$$R_3: \quad \ldots$$

$$R_4: \quad \neg S_{1,2} \vee W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1}$$
$$\ldots$$

Negierte Zielformel: $\neg W_{1,3}$

Resolutionsbeweis für die Wumpus-Welt

Resolution:

$$\begin{array}{rcl} \neg W_{1,3}, & & \neg S_{1,2} \vee W_{1,3} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1} \\ & \rightsquigarrow & \neg S_{1,2} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1} \\ S_{1,2}, & & \neg S_{1,2} \vee W_{1,2} \vee W_{2,2} \vee W_{1,1} \\ & \rightsquigarrow & W_{1,2} \vee W_{2,2} \vee W_{1,1} \\ \neg S_{1,1}, & & S_{1,1} \vee \neg W_{1,1} \\ & \rightsquigarrow & \neg W_{1,1} \\ \neg W_{1,1}, & & W_{1,2} \vee W_{2,2} \vee W_{1,1} \\ & \rightsquigarrow & W_{1,2} \vee W_{2,2} \\ \vdots & \vdots & \vdots \\ \neg W_{2,2}, & & W_{2,2} \\ & \rightsquigarrow & \square \end{array}$$

Von Wissen zu Aktionen

Wir können jetzt neue Fakten inferieren, aber wie setzen wir das Wissen in Aktionen um?

Negative Selektion: Schließe alle beweisbar gefährlichen Aktionen aus

$$A_{1,1} \wedge East_A \wedge W_{2,1} \Rightarrow \neg Forward$$

Positive Selektion: Schlage nur Aktionen vor, die beweisbar sicher sind

$$A_{1,1} \wedge East_A \wedge \neg W_{2,1} \Rightarrow Forward$$

Unterschiede?

Aus den Vorschlägen muss der Agent sich dann noch eine Aktion „aussuchen“

Probleme mit der Aussagenlogik

Obwohl Aussagenlogik für die Darstellung der WUMPUS-Welt ausreichend ist, ist sie doch ziemlich umständlich.

1. *Regeln* müssen für jedes einzelne Feld aufgestellt werden

$$\begin{aligned} R_1 &: \quad \neg S_{1,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1} \\ R_2 &: \quad \neg S_{2,1} \Rightarrow \neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1} \\ R_3 &: \quad \neg S_{1,2} \Rightarrow \neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3} \\ &\vdots \qquad \vdots \end{aligned}$$

2. Tatsächlich müssten wir ja eigentlich alle atomaren Aussagen mit einem *Zeitindex* versehen, der die Gültigkeit der Aussage in der Zeit beschreibt
   $\leadsto$ weitere Aufblähung der Regeln.

$\leadsto$ mächtigere Logik, in der wir *Objektvariablen* benutzen können

$\leadsto$ *Prädikatenlogik 1. Stufe* (first-order predicate logic)

Zusammenfassung

- Rationale Agenten benötigen *Wissen* über ihre Welt, um rationale Entscheidungen zu treffen.
- Dieses Wissen wird mit Hilfe einer *deklarativen* (Wissensrepräsentations-) Sprache dargestellt und in einer *Wissensbasis* gespeichert.
- Wir benutzen dafür (zunächst) *Aussagenlogik*.
- Aussagenlogische Formeln können *allgemeingültig, erfüllbar* oder *unerfüllbar* sein.
- Wichtig ist der Begriff der *logischen Folgerbarkeit*.
- Folgerbarkeit kann durch einen *Kalkül* mechanisiert werden kann $\leadsto$ Resolution.
- Aussagenlogik wird selbst für kleine Weltausschnitte sehr schnell unhandlich.

Grundlagen der KI

7. Erfüllbarkeit und Modellkonstruktion

Davis-Putnam, Phasentransitionen, GSAT und GWSAT

Wolfram Burgard

Inhalt

- Motivation
- Davis-Putnam Prozedur
- „Durchschnittliche“ Schwierigkeit des Erfüllbarkeitsproblems
- GSAT: Gierige SAT-Prozedur
- GWSAT: GSAT mit Random walk

Motivation (1)

Was ist die Moral aus dem Match *Kasparov* gegen *Deep Blue*? (**http://www.chess.ibm.com/**)

- *Brute-force*-Suchtechniken führen zu intelligentem Verhalten . . . (?)

  . . . aber diese Suchtechniken müssen *effizient* sein;

  . . . und man braucht auch Wissen (Eröffnungs- und Endspielbibliotheken, gute Evaluierungsfunktionen);

  . . . Imitation der menschlichen Vorgehensweise beim Schach führte bisher zu weniger eindrucksvollen Leistungen.

$\leadsto$ Thema heute: Effiziente Suchtechniken für die *Modellkonstruktion*.

Motivation (2)

Gegeben: Eine logische Theorie (Menge von Aussagen)

Frage: **Folgt** eine Aussage aus dieser Theorie?

Lösung: Reduktion auf *Unerfüllbarkeit* (bei Resolution)

Problem: UNSAT ist co-NP-vollständig
Man muss für *alle* möglichen Belegungen zeigen, dass die Formel unerfüllbar ist.

Motivation (3)

Gegeben: Eine logische Theorie (Menge von Aussagen).

Gesucht: **Modelle der Theorie**.

Beispiel: Konfigurationen, welche die durch die Theorie gegebenen *Constraints* erfüllen.

Vorgehen: Reduktion auf *Erfüllbarkeit*.
Man muss eine Belegung finden, welche die Theorie wahr macht

Aber: SAT ist NP-vollständig

UNSAT/SAT als komplementäre Probleme. Speicherbeschränkungen machen die meisten Verfahren *praktisch* unvollständig. Eine Aufgabe kann dadurch leichter lösbar sein als die andere.

Die Davis-Putnam Prozedur

Function DP
Given a set of clauses Δ defined over a set of variables Σ, return "satisfiable" if Δ is satisfiable. Otherwise return "unsatisfiable".

1. If $\Delta = \emptyset$ return "satisfiable"
2. If $\square \in \Delta$ return "unsatisfiable"
3. *Unit-Propagation Rule*: If Δ contains a *unit-clause* c, assign a truth-value to the variable in c that satisfies c, simplify Δ to Δ' and return **DP**(Δ')
4. *Splitting Rule*: Select from Σ a variable v which has not been assigned a truth-value. Assign `true` to it, simplify Δ to Δ' and call **DP**(Δ').
   (a) If the call returns "satisfiable," then return "satisfiable."
   (b) Assign the truth-value `false` to v in Δ, simplify to Δ'' and return **DP**(Δ'').

Beispiele

$$\Delta = \{\{a, b, \neg c\}, \{\neg a, \neg b\}, \{c\}, \{a, \neg b\}$$

$$\Delta = \{\{a, \neg b, \neg c, \neg d\}, \{b, \neg d\}, \{c, \neg d\}, \{d\}\}$$

Eigenschaften von DP

- DP braucht i.allg. exponentielle Laufzeit
  (Splitting rule!)
- DP ist polynomiell auf *Horn-Klauseln*, d.h. Klauseln mit maximal einem positiven Literal ($\neg A_1 \vee \cdots \vee \neg A_n \vee B \equiv \bigwedge_i A_i \rightarrow B$)
- DP ist vollständig, korrekt und terminierend.
- DP konstruiert ein Modell, falls eines vorhanden ist.

$\rightsquigarrow$ Man braucht Heuristiken für die Entscheidung, welche Variable als nächstes instantiiert werden soll, z.B. Variablen aus 2er Klauseln liefern Units - *most constrained first*.

$\rightsquigarrow$ Bei allen bisherigen Wettbewerben haben DP-basierte Verfahren am besten abgeschnitten.

Wie gut ist DP im durchschnittlichen Fall?

- Wir wissen, dass SAT NP-vollständig ist, d.h. im schlechtesten Fall brauchen wir exponentielle Zeit.
- Dies gilt ganz offensichtlich auch für die DP-Prozedur.

$\rightsquigarrow$ Könnten wir im durchschnittlichen Fall nicht besser liegen?

- Für KNF-Formeln, bei denen die Wahrscheinlichkeit 1/3 für positives Auftreten, negatives Auftreten und Nichtauftreten einer Variablen in einer Klausel ist, benötigt DP im Durchschnitt **quadratische Zeit** (Goldberg 79)!!!

$\rightsquigarrow$ Allerdings sind diese Formeln auch mit sehr großer Wahrscheinlichkeit erfüllbar!

Phasenübergänge ...

Umgekehrt kann man natürlich nach „schwierigen“ Bereichen von Problemen suchen.

Cheeseman et al. (IJCAI-91) haben folgende empirisch plausible Vermutung aufgestellt:

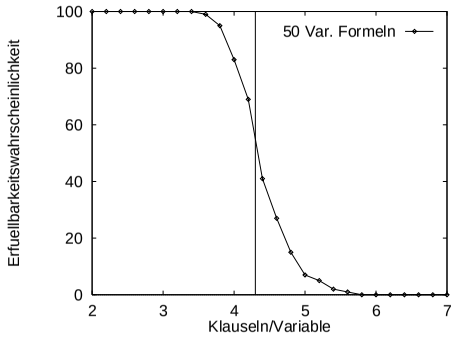
> All NP-complete problems have at least one order parameter and the hard to solve problems are around a critical value of this order parameter. This critical value (a **phase transition**) separates one region from another, such as overconstrained and underconstrained regions of the problem space.

Bestätigung für Graphfärbung und Hamiltonkreise ... später auch für andere NP-vollständige Probleme.

Der Phasenübergang bei 3-SAT

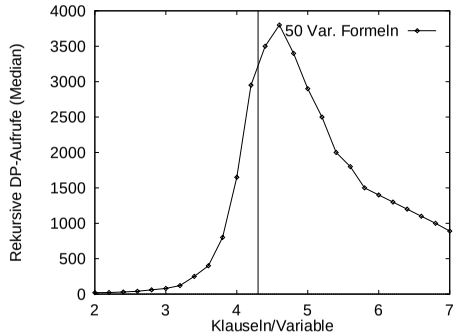
Konstantes Klausellängenmodell (Mitchell et al., AAAI-92): Klausellänge k vorgegeben. Wähle für jede Klausel k Variablen und benutze Komplement mit Wahrscheinlichkeit 0.5 für jede der Variablen.

Phasenübergang bei 3SAT bei einem Klausel/Variablen-Verhältnis bei ca. 4.3:



Empirische Schwierigkeit ...

Die Davis-Putnam (DP) Prozedur zeigt bei diesem Phasenübergang extreme Laufzeitspitzen:



Aber: Auch in der „einfachen“ Region der erfüllbaren Instanzen kann es einige sehr schwierige Instanzen geben!

Bemerkungen zum Phasenübergang

- Wenn die Lösungswahrscheinlichkeit nahe bei 1 (*under-constrained*) ist, gibt es sehr viele Lösungen, und meist ist der erste Suchpfad einer Backtracking-Suche schon erfolgreich.
- Liegt die Lösungswahrscheinlichkeit nahe bei 0 (*over-constrained*), kann dies meist sehr früh in der Suche festgestellt werden.
- An der Phasenübergangsstelle gibt es viele „fast“ Lösungen („close, but no cigar“).

$\rightsquigarrow$ (eingeschränkte) Möglichkeit der Voraussage der Lösungsschwierigkeit anhand des Parameters.

$\rightsquigarrow$ (suchintensive) Benchmark-Probleme findet man in der Phasenregion.

Lokale Suche zum Lösen logischer Probleme

In vielen Fällen ist man daran interessiert, eine erfüllende Belegung für Variablen zu finden (Beispiel: CSP) und man kann auf *Vollständigkeit* des Lösungsverfahrens verzichten, wenn man dafür „viele“ sehr große Instanzen lösen kann.

Standardverfahren für Optimierungsprobleme: **Lokale Suche**

- Ausgehend von einer (zufälligen) Konfiguration
- versucht man durch lokale Modifikationen bessere Konfigurationen zu erzeugen.

$\rightsquigarrow$ Hauptproblem: lokale Extrema

Umgang mit lokalen Extrema

Für logische Probleme könnte man als Gütemaß einer Konfiguration die Anzahl der erfüllten Constraints/Klauseln nehmen.

Aber lokale Suche scheint wenig geeignet, da man ja ein globales Extremum finden will (alle Constraints/Klauseln erfüllt).

Durch Neustart und/oder Einbringen von etwas Rauschen kann man aber oft lokale Extrema verlassen.

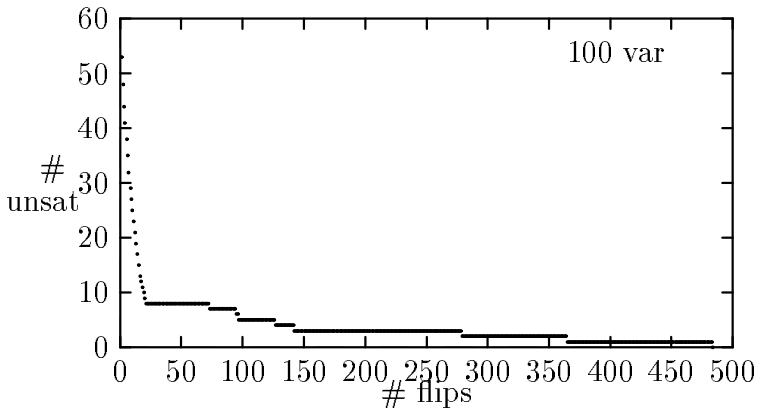
Tatsächlich: Lokale Suche funktioniert sehr gut beim Finden von erfüllenden Belegungen von KNF-Formeln (auch ohne Rauschen).

GSAT

Procedure GSAT
INPUT: a set of clauses α, MAX-FLIPS, and
MAX-TRIES
OUTPUT: a satisfying truth assignment of α, if found
begin
  for i:=1 to MAX-TRIES
    T:= a randomly generated truth assignment
    for j:= 1 to MAX-FLIPS
      if T satisfies α **then return** T
      p:= a propositional variable such that a change in its truth assignment gives
        the largest increase in the total number of clauses of α that are satisfied by T
      T:= T with the truth assignment of p reversed
    end for
  end for
  return “no satisfying assignment found”
end

Das Suchverhalten von GSAT

- Man muß im Gegensatz zu normaler lokaler Suche auch Seitwärtsbewegungen zulassen!
- Die meiste Zeit wird mit Suche auf einem *Plateau* zugebracht.



Anwendung von GSAT

formulas		GSAT			DP		
vars	clauses	M-FLIPS	tries	time	choices	depth	time
50	215	250	6.4	0.4s	77	11	1.4s
100	430	500	42.5	6s	84 x 10^3	19	2.8m
140	602	700	52.6	14s	2.2 x 10^6	27	4.7h
150	645	1500	100.5	45s	—	—	—
300	1275	6000	231.8	12m	—	—	—
500	2150	10000	995.8	1.6h	—	—	—

GSAT + Rauschen

Um ungeeignete Plateaus zu verlassen, eignet sich die Injektion von Rauschen.

1. **Simuliertes Abkühlen**
2. **Random walk**:
   Mit Wahrscheinlichkeit p, wähle eine Variable in einer noch nicht erfüllten Klausel und ändere ihre Belegung. Mit Wahrscheinlichkeit $1 - p$, benutze die GSAT-Strategie.
3. **Random noise:**
   Wie *Random walk*, aber wähle irgendeine Klausel (auch eine schon erfüllte).

Ergebnisse auf Random-3CNF Formeln

formula		GSAT						Simul. Ann.	
		basic		walk		noise			
vars	clauses	time	flips	time	flips	time	flips	time	flips
100	430	.4	7554	.2	2385	.6	9975	.6	4748
200	860	22	284693	4	27654	47	396534	21	106643
400	1700	122	2.6×10^6	7	59744	95	892048	75	552433
600	2550	1471	30×10^6	35	241561	921	7.8×10^6	427	2.7×10^6
800	3400	*	*	286	1.8×10^6	*	*	*	*
1000	4250	*	*	1095	5.8×10^6	*	*	*	*
2000	8480	*	*	3255	23×10^6	*	*	*	*

- Zeit in CPU-Sekunden auf einer SGI Challenge 100 MHz
- Jeweils beste Konfiguierung (Restarts, p), wobei *walk* und *noise* keine Restarts brauchen.
- "*" bedeutet mehr als 1000 Restarts oder mehr als 20 CPU-Stunden.

Ergebnisse auf Schaltkreissyntheseproblemen

Schaltkreissyntheseprobleme für Random-Schaltkreise zum Testen von *Integer-programming*-Ansätzen.

formula		Int.P.	GSAT						Simul. Ann.	
			basic		walk		noise			
ids	clauses	time	time	flips	time	flips	time	flips	time	flips
f16a1	1650	2039	114	709895	4.2	3371	708	1025454	25	98105
f16b1	1728	78	452	2870019	24	25529	2199	287226	22	96612
f16c1	1580	758	3.5	12178	1.6	1545	11	14614	8.4	21222
f16d1	1230	1547	174	872219	6.2	5582	371	387491	8.4	25027
f16e1	1245	2156	1.7	2090	1.8	1468	3	3130	6.3	5867

Ähnliche Ergebnisse auch auf anderen Schaltkreissyntheseproblemen und Diagnoseproblemen.

Ausblick

- Hauptproblem: Bestimmung der **Parameter** `MAX-TRIES`, p ... scheint aber für Problemklassen zuverlässig zu sein.
- Nebenproblem: Kann u.U. eine vorhandene Belegung nicht finden $\leadsto$ Unvollständigkeit bzgl. Erfüllbarkeit.

$\leadsto$ **Einsatzgebiet:** Nur für Anwendungen mit sehr großen Probleminstanzen, bei denen es sich lohnt, die Parameter anzupassen.

- GWSAT scheint besser zu sein als GSAT.
- Tabu-Suche ist vermutlich besser als GWSAT – behaupten einige Autoren.

$\leadsto$ An der Grenze des Machbaren ist ein bisschen „schwarze Magie“ gefragt.

$\leadsto$ Lokale Suche scheint so etwas wie „Intuition“ zu modellieren.

Grundlagen der KI

8. Prädikatenlogik

Syntax und Semantik, Normalformen, Herbrandexpansion

Wolfram Burgard

Inhalt

Motivation

Man kann schon eine ganze Menge mit Aussagenlogik anfangen. Allerdings ist es ärgerlich, dass es nicht möglich ist, auf die *Struktur* der atomaren Aussagen einzugehen.

Beispiel:
„Alle Blöcke sind rot“
„Es gibt einen Block A“
Daraus sollte folgen: „A ist rot“

Mit Aussagenlogik können wir dies aber nicht ausdrücken.

Idee: Wir führen Individuenvariablen, Prädikate, Funktionen . . . ein
$\rightsquigarrow$ *Prädikatenlogik 1. Stufe* (PL1)

Alphabet der Prädikatenlogik 1. Stufe

Symbole:

- Operatoren: $\neg$,$\wedge$,$\vee$,$\forall$,$\exists$, $=$
- Variablen: $x, x_1, x_2, \ldots, x', x'', \ldots, y, \ldots, z, \ldots$
- Klammersymbole: ( )
- Funktionssymbole (z.B. Gewicht, Farbe) (zur Repräsentation von Objekten)
- Prädikatensymbole (z.B. Block, Rot) (für Aussagen über Objekten)

Hinweise:

1. Prädikaten- und Funktionssymbole besitzen eine Stelligkeit (Zahl der Argumente).
   0-stellige Prädikate: aussagenlogische Atome
   0-stellige Funktionen: Konstanten
2. Wir nehmen abzählbar viele Prädikate und Funktionen jeder Stelligkeit an.
3. „=“ wird nicht als Prädikat behandelt!

Grammatik der Prädikatenlogik 1. Stufe (1)

Terme (stehen für Objekte):

1. Jede Variable ist ein Term.
2. Wenn $t_1, t_2, \ldots, t_n$ Terme sind und f ein n-stelliges Funktionssymbol, dann ist $f(t_1, t_2, \ldots, t_n)$ auch ein Term.

   Terme ohne Variablen: **Grundterme**

Atomare Formeln (stehen für Aussagen über Objekten)

1. Wenn $t_1, t_2, \ldots, t_n$ Terme sind und P ein n-stelliges Prädikat ist, dann ist $P(t_1, t_2, \ldots, t_n)$ eine atomare Formel.
2. Wenn t_1 und t_2 Terme sind, dann ist $t_1 = t_2$ eine atomare Formel.

   Atomare Formeln ohne Variablen: **Grundatome**.

Grammatik der Prädikatenlogik 1. Stufe (2)

Formeln:

1. Jede atomare Formel ist eine Formel.
2. Wenn φ und ψ Formeln sind und x eine Variable ist, dann sind $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi, \exists x\ \varphi$ und $\forall x\ \varphi$ auch Formeln.
   $\forall, \exists$
   binden so stark wie $\neg$.

Die Aussagenlogik ist eine Teilsprache der PL1:

1. Atomare Formeln: nur 0-stellige Prädikate
2. Weder Variablen noch Quantoren.

Notation

Klammern zur Strukturierung

Verwendung anderer Klammern zwecks Leserlichkeit $\{, \}, [,]$.

Prädikate: Person, Schön, älter-als

Funktionen: Vater-von, Nachfolger, a, b

Variablen: $x, y, z,$

Geschachtelte Quantoren: $\forall x \forall y \ldots$ auch $\forall x, y \ldots$

Alternative Notation:

hier	sonst
$\neg\varphi$	$\sim\varphi \quad \overline{\varphi}$
$\varphi \wedge \psi$	$\varphi\&\psi \quad \varphi \cdot \psi \quad \varphi, \psi$
$\varphi \vee \psi$	$\varphi \mid \psi \quad \varphi; \psi \quad \varphi + \psi$
$\varphi \Rightarrow \psi$	$\varphi \rightarrow \psi \quad \varphi \supset \psi$
$\varphi \Leftrightarrow \psi$	$\varphi \leftrightarrow \psi \quad \varphi \equiv \psi$
$\forall x \varphi$	$(\forall x)\varphi \quad \bigwedge x\varphi$
$\exists x \varphi$	$(\exists x)\varphi \quad \bigvee x\varphi$

Bedeutung von PL1-Formeln

Unser Beispiel: $\forall x[\texttt{Block}(x) \Rightarrow \texttt{Rot}(x)], \texttt{Block}(a)$

Für alle Objekte x gilt: Falls x ein Block ist, dann ist x rot. a ist ein Block.

Generell:

- Terme werden als Objekte interpretiert.
- Universell quantifizierte Variablen laufen über alle Objekte des Universums.
- Existentiell quantifizierte Variablen stehen für ein Objekt des Universums (das den quantifizierten Ausdruck wahr macht).
- Prädikate stehen für Teilmengen des Universums.

Ähnlich wie für Aussagenlogik definieren wir: Interpretationen, Erfüllung, Modelle, Allgemeingültigkeit, Folgerung, ...

Semantik von PL1: Interpretationen

Interpretation: $\mathcal{I} = \langle \mathbf{D}, \cdot^{\mathcal{I},\alpha} \rangle$ mit $\mathbf{D}$ eine beliebige nicht-leere Menge und $\cdot^{\mathcal{I},\alpha}$ eine Funktion, die

- n-stellige Funktionssymbole auf Funktionen über $\mathbf{D}$ abbildet: $f^{\mathcal{I}} \in [\mathbf{D}^n \to \mathbf{D}]$
- Individuenkonstanten auf Elemente aus $\mathbf{D}$ abbildet: $a^{\mathcal{I}} \in \mathbf{D}$
- n-stellige Prädikatensymbole auf Relationen über $\mathbf{D}$ abbildet: $P^{\mathcal{I}} \subseteq \mathbf{D}^n$

Interpretation von Grundtermen:

$$(f(t_1, \ldots, t_n))^{\mathcal{I}} \quad = \quad f^{\mathcal{I}}({t_1}^{\mathcal{I}}, \ldots, {t_n}^{\mathcal{I}}) \ (\in \mathbf{D})$$

Erfüllung von Grundatomen $P(t_1, \ldots, t_n)$:

$$\mathcal{I} \models P(t_1, \ldots, t_n) \quad \text{gdw.} \quad \langle {t_1}^{\mathcal{I}}, \ldots, {t_n}^{\mathcal{I}} \rangle \in P^{\mathcal{I}}$$

Beispiel 1

$$\begin{aligned}
\mathbf{D} &= \{d_1, \ldots, d_n \mid n > 1\} \\
\texttt{a}^{\mathcal{I}} &= d_1 \\
\texttt{b}^{\mathcal{I}} &= d_2 \\
\texttt{c}^{\mathcal{I}} &= \ldots \\
\texttt{Block}^{\mathcal{I}} &= \{d_1\} \\
\texttt{Rot}^{\mathcal{I}} &= \mathbf{D} \\
\mathcal{I} &\models \texttt{Rot(b)} \\
\mathcal{I} &\not\models \texttt{Block(b)}
\end{aligned}$$

Beispiel 2

$$\begin{aligned}
\mathbf{D} &= \{1, 2, 3, \ldots\} \\
1^{\mathcal{I}} &= 1 \\
2^{\mathcal{I}} &= 2 \\
&\vdots \\
\texttt{Even}^{\mathcal{I}} &= \{2, 4, 6, \ldots\} \\
\texttt{succ}^{\mathcal{I}} &= \{(1 \mapsto 2), (2 \mapsto 3), \ldots\} \\
\mathcal{I} &\models \texttt{Even}(2) \\
\mathcal{I} &\not\models \texttt{Even}(\texttt{succ}(2))
\end{aligned}$$

Semantik von PL1: Variablenbelegung

V Menge aller Variablen. Funktion $\alpha\colon V \to \mathbf{D}$.

Notation: $\alpha[x/d]$ ist identisch mit α bis auf die Stelle x. Für x gilt: $\alpha[x/d](x) = d$.

Interpretation von Termen unter $\mathcal{I}, \alpha$:

$$\begin{aligned} x^{\mathcal{I},\alpha} &= \alpha(x) \\ a^{\mathcal{I},\alpha} &= a^{\mathcal{I}} \\ (f(t_1,\ldots,t_n))^{\mathcal{I},\alpha} &= f^{\mathcal{I}}({t_1}^{\mathcal{I},\alpha},\ldots,{t_n}^{\mathcal{I},\alpha}) \end{aligned}$$

Erfüllbarkeit von atomaren Formeln:

$$\mathcal{I},\alpha \models P(t_1,\ldots,t_n) \quad \text{gdw.} \quad \langle {t_1}^{\mathcal{I},\alpha},\ldots,{t_n}^{\mathcal{I},\alpha}\rangle \in P^{\mathcal{I}}$$

Beispiel

$$\begin{aligned} \alpha &= \{(\mathtt{x} \mapsto d_1),(\mathtt{y} \mapsto d_2)\} \\ \mathcal{I},\alpha &\models \texttt{Rot(x)} \\ \mathcal{I},\alpha[\mathtt{y}/d_1] &\models \texttt{Block(y)} \end{aligned}$$

Semantik von PL1: Erfüllbarkeit

Eine Formel φ wird von einer Interpretation $\mathcal{I}$ unter einer Variablenbelegung α erfüllt, d.h. $\mathcal{I},\alpha \models \varphi$:

$$\begin{aligned} &\mathcal{I},\alpha \models \top \\ &\mathcal{I},\alpha \not\models \bot \\ &\mathcal{I},\alpha \models \neg\varphi \quad \text{gdw.} \quad \mathcal{I},\alpha \not\models \varphi \\ &\ldots \end{aligned}$$

und alle weiteren propositionalen Regeln sowie

$$\begin{aligned} \mathcal{I} \models P(t_1,\ldots,t_n) \quad &\text{gdw.} \quad \langle {t_1}^{\mathcal{I},\alpha},\ldots,{t_n}^{\mathcal{I},\alpha}\rangle \in P^{\mathcal{I}} \\ \mathcal{I},\alpha \models \forall x\,\varphi \quad &\text{gdw.} \quad \text{für alle } d \in \mathbf{D} \text{ gilt } \mathcal{I},\alpha[x/d] \models \varphi \\ \mathcal{I},\alpha \models \exists x\,\varphi \quad &\text{gdw.} \quad \text{es gibt ein } d \in \mathbf{D} \text{ mit } \mathcal{I},\alpha[x/d] \models \varphi \end{aligned}$$

Beispiel

$$\begin{aligned} \Theta &= \left\{ \begin{array}{l} \texttt{Block(a), Block(b)} \\ \forall\texttt{x (Block(x)} \Rightarrow \texttt{Rot(x))} \end{array} \right\} \\ \mathbf{D} &= \{d_1,\ldots,d_n\}\ n > 1 \\ \mathtt{a}^{\mathcal{I}} &= d_1 \\ \mathtt{b}^{\mathcal{I}} &= d_2 \\ \texttt{Block}^{\mathcal{I}} &= \{d_1\} \\ \texttt{Rot}^{\mathcal{I}} &= \mathbf{D} \\ \alpha &= \{(\mathtt{x} \mapsto d_1),(\mathtt{y} \mapsto d_2)\} \end{aligned}$$

Fragen:

1. $\mathcal{I},\alpha \models \texttt{Block(b)} \vee \neg\texttt{Block(b)}$?
2. $\mathcal{I},\alpha \models \texttt{Block(x)} \Rightarrow (\texttt{Block(x)} \vee \texttt{Block(y)})$?
3. $\mathcal{I},\alpha \models \texttt{Block(a)} \wedge \texttt{Block(b)}$?
4. $\mathcal{I},\alpha \models \forall\texttt{x (Block(x)} \Rightarrow \texttt{Rot(x))}$?
5. $\mathcal{I},\alpha \models \Theta$?

Freie und gebundene Variablen

$$\forall x[R(\boxed{y},\boxed{z}) \wedge \exists y\{\neg P(y,x) \vee R(y,\boxed{z})\}]$$

Eingekästelte Vorkommen von $\mathtt{y}$ und $\mathtt{z}$ sind **frei**. Alle anderen Vorkommen von $\mathtt{x}, \mathtt{y}, \mathtt{z}$ sind **gebunden**.

Formeln, in denen keine freien Variablen vorkommen, heißen **geschlossene** Formeln oder auch **Sätze**. Bei der Formulierung von Theorien benutzen wir nur geschlossene Formeln.

Beachte: Die Begriffe *logische Äquivalenz, Erfüllbarkeit, Folgerbarkeit* usw. sind bei geschlossenen Formeln unabhängig von der Variablenbelegung α (d.h. man kann immer über alle Variablenbelegungen gehen).

Bei geschlossenen Formeln wird dann α auf der linken Seite des Modellbeziehungszeichens weggelassen:

$$\mathcal{I} \models \varphi$$

Terminologie

Eine Interpretation $\mathcal{I}$ heißt **Modell** von φ *unter* α, wenn

$$\mathcal{I}, \alpha \models \varphi.$$

Eine Formel φ der PL1 kann, ebenso wie in der Aussagenlogik) **erfüllbar**, **unerfüllbar**, **falsifizierbar** oder **allgemeingültig** sein.

Analog sind zwei Formeln **logisch äquivalent** ($\varphi \equiv \psi$), wenn für alle $\mathcal{I}, \alpha$ gilt:

$$\mathcal{I}, \alpha \models \varphi \quad \text{gdw.} \quad \mathcal{I}, \alpha \models \psi$$

Beachte: $\mathtt{P(x)} \not\equiv \mathtt{P(y)}$!

Auch **logische Folgerbarkeit** ist analog zu propositionaler Logik.

Frage: Wie können wir einen Ableitungsbegriff definieren?

Pränex-Normalform

Wegen der Quantoren können wir nicht direkt die KNF einer Formel bilden.

Erster Schritt: Bilden der *Pränex-Normalform*

Quantorenpräfix + (quantorenfreie) Matrix φ:

$$\forall x_1 \, \forall x_2 \, \exists x_3 \, \ldots \forall x_n \, \varphi$$

Äquivalenzen für die Erzeugung der Pränex-Normalform

$$\begin{aligned}
(\forall x\varphi) \wedge \psi &\equiv \forall x(\varphi \wedge \psi) \quad x \text{ nicht frei in } \psi \\
(\forall x\varphi) \vee \psi &\equiv \forall x(\varphi \vee \psi) \quad x \text{ nicht frei in } \psi \\
(\exists x\varphi) \wedge \psi &\equiv \exists x(\varphi \wedge \psi) \quad x \text{ nicht frei in } \psi \\
(\exists x\varphi) \vee \psi &\equiv \exists x(\varphi \vee \psi) \quad x \text{ nicht frei in } \psi \\
\forall x\varphi \wedge \forall x\psi &\equiv \forall x(\varphi \wedge \psi) \\
\exists x\varphi \vee \exists x\psi &\equiv \exists x(\varphi \vee \psi)
\end{aligned}$$

$$\begin{aligned}
\neg\forall x\varphi &\equiv \exists x\neg\varphi \\
\neg\exists x\varphi &\equiv \forall x\neg\varphi
\end{aligned}$$

. . . und aussagenlogische Äquivalenzen

Erzeugen der Pränex-Normalform

1. Eliminierung von $\Rightarrow$ und $\Leftrightarrow$
2. $\neg$ nach innen
3. Quantoren nach außen

Beispiel:
$\neg\forall x\,[(\forall x\,P(x)) \Rightarrow Q(x)] \rightsquigarrow \neg\forall x\,[\neg(\forall x\,P(x)) \vee Q(x)] \rightsquigarrow \exists x\,[(\forall x\,P(x)) \wedge \neg Q(x)]$
Und nun?

Lösung: Variablenumbenennung
$\varphi[x/t]$ entsteht aus φ, indem alle **freien** Vorkommen von x in φ durch den Term t ersetzt werden.

Lemma: Sei y eine Variable, die nicht in φ vorkommt. Dann gilt $\forall x\,\varphi \equiv \forall y\,\varphi[x/y]$ und $\exists x\,\varphi \equiv \exists y\,\varphi[x/y]$.

Satz: Es existiert ein Algorithmus, der zu jeder Formel ihre Pränex-Normalform berechnet.

Ableitungen in PL1

Was nutzt uns die Pränex-Normalform?

Leider gibt es nicht wie für die propositionalen Logik einfache Gesetze, die uns erlauben, Erfüllbarkeit oder Allgemeingültigkeit zu bestimmen (durch Umformung in DNF oder KNF).

Aber: Wir können das Erfüllbarkeitsproblem der Prädikatenlogik auf Erfüllbarkeit in propositionaler Logik *reduzieren*. I.allg. entstehen dabei allerdings unendliche Mengen von propositionalen Formeln.

Dann: Anwenden von Resolution.

Skolemisierung

Idee: Eliminierung der Existenzquantoren durch Funktionen, die uns das „richtige“ Element liefern.

Satz (*Skolem-Normalform*): Sei φ eine geschlossene Formel in Pränex-Normalform, so dass alle quantifizierten Variablen paarweise verschieden sind und die Funktionssymbole $g_1, g_2, \ldots$ nicht in φ auftreten. Sei

$$\varphi = \forall x_1 \ldots \forall x_i\,\exists y\,\psi,$$

dann ist φ erfüllbar gdw.

$$\varphi' = \forall x_1 \ldots \forall x_i\,\psi[y/g_i(x_1, \ldots, x_i)]$$

erfüllbar ist.

Beispiel: $\forall x\,\exists y\,[P(x) \Rightarrow Q(y)] \rightsquigarrow \forall x\,[P(x) \Rightarrow Q(g(x))]$

Skolem-Normalform

Skolem-Normalform: Pränex-Normalform ohne Existenzquantoren.
Schreibweise: φ^* ist SNF von φ.

Satz: Zu jeder geschlossen Formel φ kann ihre SNF φ^* effektiv berechnet werden.

Beispiel:
$\exists x\,((\forall x\,P(x)) \wedge \neg Q(x))$ weiter geht's so:
$\exists y\,((\forall x\,P(x)) \wedge \neg Q(y))$
$\exists y\,(\forall x\,(P(x) \wedge \neg Q(y)))$
$\forall x\,(P(x) \wedge \neg Q(g_0))$

Beachte: Diese Transformation ist *keine Äquivalenztransformation*, sie erhält nur Erfüllbarkeit!

Beachte: . . . und sie ist nicht eindeutig.
Beispiel: $\exists x\,(p(x)) \wedge \forall y\,(q(y))$

Grundterme, Herbrandexpansion

Die **Grundtermmenge** (oder das *Herbranduniversum*) über einer Menge von SNF-Formeln Θ^* ist die (abzählbare) Menge aller Grundterme, die sich mit Symbolen aus Θ^* bilden lassen (falls es kein Konstantensymbol gibt, wird eines hinzugefügt). Diese Menge wird mit $D(\Theta^*)$ bezeichnet $\leadsto$ repräsentative Trägermenge.

Die **Herbrandexpansion** $E(\Theta^*)$ ist die Instantiierung der Matrizen ψ_i aller Formeln in Θ^* durch alle Terme $t \in D(\Theta^*)$:

$$E(\Theta^*) = \{\psi_i[x_1/t_1, \ldots, x_n/t_n] \mid (\forall x_1, \ldots, x_n\ \psi_i) \in \Theta^*, t_j \in D(\Theta^*)\}$$

Satz (*Herbrand*): Sei Θ^* eine Menge von Formeln in SNF. Dann ist Θ^* erfüllbar gdw. $E(\Theta^*)$ erfüllbar ist.

Beachte: Falls $D(\Theta^*)$ und Θ^* endlich, dann ist die Herbrandexpansion endlich $\leadsto$ endliche aussagenlogische Theorie

Unendliche aussagenlogische Theorien ...

Gibt es bei unendlichen Formelmengen endliche Beweise?

Satz (*Kompaktheit der Aussagenlogik*): Jede (höchstens abzählbare) Menge von Formeln der Aussagenlogik ist erfüllbar genau dann, wenn *jede* endliche Teilmenge erfüllbar ist.

Korollar: Eine (höchstens abzählbare) Menge von Formeln der Aussagenlogik ist unerfüllbar genau dann wenn bereits *eine* endliche Teilmenge unerfüllbar ist.

Korollar (*Kompaktheit der PL1*): Jede (höchstens abzählbare) Menge von Formeln der Prädikatenlogik ist erfüllbar genau dann, wenn jede endliche Teilmenge erfüllbar ist.

Rekursive Aufzählbarkeit und Entscheidbarkeit

Es lässt sich ein *Semi-Entscheidungsverfahren* für Allgemeingültigkeit konstruieren, d.h. wir könnten einen (ziemlich ineffizienten) Algorithmus angeben, der Schritt für Schritt alle allgemeingültigen Formeln aufzählt.

Satz: Die Menge der allgemeingültigen (und unerfüllbaren) Formeln in PL1 ist rekursiv aufzählbar.

Wie sieht es mit den *erfüllbaren* Formeln aus?

Satz (*Unentscheidbarkeit von PL1*): Es ist unentscheidbar ob eine Formel der PL1 allgemeingültig ist.
(Beweis durch Reduktion des Postschen Korrespondenzproblems.)

Korollar: Die Menge der erfüllbaren Formeln in PL1 ist nicht rekursiv aufzählbar.

Mit anderen Worten: Falls eine Formel allgemeingültig ist, können wir dafür auch effektiv eine Bestätigung finden. Ansonsten können wir u.U. in eine Endlosschleife geraten.

Ausblick: Mögliche Erweiterungen

PL1 ist zwar sehr ausdrucksstark, aber manchmal möchte man u.U. mehr ...

- Logik 2. Stufe: auch über Prädikate quantifizieren

$$\forall x, y\,[(x = y) \Leftrightarrow \{\forall p\,[p(x) \Leftrightarrow p(y)]\}]$$

$\leadsto$ Allgemeingültigkeit ist nicht mehr semi-entscheidbar (keine Kompaktheit mehr)

- Lambda-Ausdrücke: Definition von Prädikaten, z.B. $\lambda x, y[\exists z\ P(x, z) \wedge Q(z, y)]$ definiert neues 2-stelliges Prädikat.

$\leadsto$ lässt sich auf PL1 reduzieren durch Lambda-Reduktion

- Eindeutigkeitsquantor: $\exists! x\ \varphi(x)$ – es existiert *genau ein* x ...

$\leadsto$ Reduktion auf PL1:

$$\exists x\,[\varphi(x) \wedge \forall y\,\{\varphi(y) \Rightarrow x = y\}]$$

Ausblick: Verarbeitung

- **PL1-Resolution:** Statt Resolution auf der Herbrandexpansion wird Resolution über Klauseln mit Variablen durchgeführt. $\leadsto$ **Unifikation**, $\leadsto$ Resolution über Klassen von Grundinstanzen
- **Einschränkung der syntaktischen Form:** Nur Horn-Klauseln $\leadsto$ PROLOG $\leadsto$ erheblich effizientere Methoden
- **Endliche Theorien:** In Anwendungen hat man oft eine endliche Menge von Objekten vorgegeben. *Domain closure axiom*:

$$\forall x[x = c_1 \vee x = c_2 \vee \ldots \vee x = c_n]$$

Übersetzung in endliche aussagenlogische Form möglich.

Zusammenfassung

- PL1 erlaubt es, Aussagen zu strukturieren und gibt uns damit eine erheblich größere Aussagekraft als Aussagenlogik.
- Formeln bestehen aus *Termen* und *atomaren Formeln*, die mit Hilfe von *Konnektoren* und *Quantoren* zu Formeln zusammengesetzt werden können.
- Interpretationen in PL1 bestehen aus einem *Universum* und der *Interpretationsfunktion*.
- Der Satz von Herbrand zeigt, dass Erfüllbarkeit in PL1 auf Erfüllbarkeit in Aussagenlogik reduziert werden kann (allerdings entstehen dabei u.U. unendliche Formelmengen).
- Aber: Allgemeingültigkeit in PL1 ist nicht entscheidbar!

Grundlagen der KI

9. Modellierung mit Logik

Zeit, Raum & der ganze Rest

Wolfram Burgard

Inhalt

- Logische Agenten für die WUMPUS-Welt
- Situationskalkül
- Das Rahmenproblem
- Raummodellierung
- Kausale & diagnostische Regeln
- Aktionsauswahl

Logikbasierte Agenten

function KB-AGENT(*percept*) **returns** an *action*
  static: *KB*, a knowledge base
    t, a counter, initially 0, indicating time

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  action ← ASK(*KB*, MAKE-ACTION-QUERY(*t*))
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  t ← *t* + 1
  return *action*

Anfrage (Make-Action-Query): $\exists a\, Action(a, \mathtt{t})$

Variablenbindung an a sollte z.B. in der WUMPUS-Welt folgende Antworten geben:

turn(right), turn(left), forward, shoot, grab, release, climb.

Reflex-Agenten

. . . reagieren nur auf Perzepte

Beispiel für eine Perzept-Aussage (zum Zeitpunkt 5):

$$Percept(stench, breeze, glitter, none, none, 5)$$

1. Schritt: Abstraktion der wesentlichen Eigenschaften

$$\begin{aligned}
\forall b, g, u, c, t\ [Percept(stench, b, g, u, c, t) &\Rightarrow Stench(t)] \\
\forall s, g, u, c, t\ [Percept(s, breeze, g, u, c, t) &\Rightarrow Breeze(t)] \\
\forall s, b, u, c, t\ [Percept(s, b, glitter, u, c, t) &\Rightarrow AtGold(t)] \\
&\ldots
\end{aligned}$$

2. Schritt: Aktionsauswahl

$$\begin{aligned}
\forall t\ [AtGold(t) &\Rightarrow Action(grab, t)] \\
&\ldots
\end{aligned}$$

Aber: Unser Reflex-Agent weiß nicht, wann er aus der Höhle klettern soll, und kann keine Endlosschleifen verhindern.

Modellbasierte Agenten

. . . haben ein internes Modell

- aller wesentlichen Aspekte ihrer Umgebung,
- der Ausführbarkeit und Wirkung von Aktionen,
- von weiteren wesentlichen Gesetzmäßigkeiten der Welt,
- und den eigenen Zielen.

Wichtigster Aspekt: Wie ändert sich die Welt?

$\rightsquigarrow$ **Situationskalkül** (McCarthy 63).

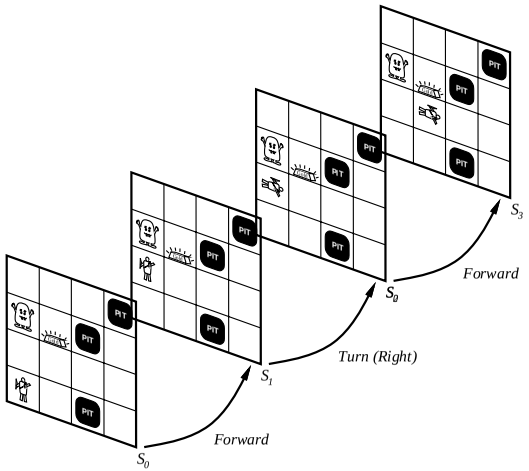
Situationskalkül

- Ein Weg, um mit PL1 **dynamische Welten** zu beschreiben.
- **Zustände** werden durch Terme repräsentiert.
- Die Welt ist in einem Zustand s und kann nur durch Ausführung einer **Aktion** geändert werden: $do(a, s)$ ist der **Nachfolgezustand**, wenn a ausgeführt wurde.
- Aktionen haben **Vorbedingungen** und werden durch ihre **Effekte** beschrieben.
- Relationen, die ihren Wahrheitswert über die Zeit ändern, heißen **Fluents**. Darstellung durch zusätzliches Argument, das ein Zustandsterm ist. $Holding(x, s)$ bedeutet z.B., dass in der Situation s das Objekt x gehalten wird.

Beispiel: WUMPUS-Welt

Sei s_0 die Ausgangssituation und

$s_1 = do(forward, s_0), \qquad s_2 = do(turn(right), s_1), \qquad s_3 = do(forward, s_2)$



Beschreibung von Aktionen

Vorbedingungen: Um etwas aufzuheben, muss es da sein und tragbar sein

$$\forall x, s\ [Poss(grab(x), s) \Leftrightarrow Present(x, s) \wedge Portable(x)]$$

In der WUMPUS-Welt:

$$Portable(gold),\ \forall s\ [AtGold(s) \Rightarrow Present(gold, s)]$$

Positives Effektaxiom:

$$\forall x, s\ [Poss(grab(x), s) \Rightarrow Holding(x, do(grab(x), s))]$$

Negatives Effektaxiom:

$$\forall x, s\ \neg Holding(x, do(release(x), s))$$

Das Rahmenproblem

Es gelte: $Holding(gold, s_0)$.
Folgt: $\neg Holding(gold, do(release(gold), s_0))$?

Es gelte: $\neg Holding(gold, s_0)$.
Folgt: $\neg Holding(gold, do(turn(right), s_0))$?

- Man muss auch spezifizieren, welche *Fluents* nicht geändert werden!
- Das Rahmen- (oder *Frame*-) Problem: Spezifikation der Eigenschaften, die sich bei Aktionen *nicht* ändern.

$\rightsquigarrow$ Es müssen auch **Rahmenaxiome** spezifiziert werden.

Anzahl der Rahmenaxiome

$$\forall a, x, s\ [Holding(x, s) \wedge (a \neq release(x)) \Rightarrow Holding(x, do(a, s))]$$

$$\forall a, x, s\ [\neg Holding(x, s) \wedge \{(a \neq grab(x)) \vee \neg Poss(grab(x), s)\} \Rightarrow \neg Holding(x, do(a, s))]$$

Kann u.U. sehr aufwendig werden, da ($O(|F| \times |A|)$) Axiome spezifiziert werden müssen.

Nachfolgezustandsaxiome

Eine elegantere Art, das Rahmenproblem zu lösen, ist, den Nachfolgezustand vollständig zu beschreiben:

wahr nach Aktion $\Leftrightarrow$ [Aktion hat es *wahr* gemacht $\vee$ bereits *wahr* und durch Aktion nicht *falsch* geworden]

Beispiel für *Holding*:

$$\forall a, x, s\ [Holding(x, do(a, s)) \Leftrightarrow \{(a = grab(x) \wedge Poss(a, s)) \vee (Holding(x, s) \wedge a \neq release(x))\}]$$

Kann auch automatisch aus der Angabe nur der Effektaxiome kompiliert werden („Erklärungsabschluss“). Dabei wird die Annahme gemacht, dass nur die spezifizierten Effekte auftreten können.

Grenzen dieser Version des Situationskalküls

- Keine explizite **Zeit**. Man kann nicht darüber reden, wie lange eine Aktion benötigt, wenn Sie ausgeführt wird.
- **Nur ein Agent**. Im Prinzip können aber auch mehrere Agenten modelliert werden.
- **Keine parallele** Ausführung von Aktionen.
- **Diskrete Zustände**. Keine kontinuierlichen Aktionen wie das Schieben eines Objekts von A nach B.
- **Abgeschlossene Welt**. Nur der Agent verändert den Zustand.
- **Determinismus**. Aktionen werden immer mit absoluter Sicherheit ausgeführt.

$\rightsquigarrow$ Trotzdem für viele Fälle ausreichend.

Raum und Bewegung

- Der Agent hat eine Orientierung: 0 (nach Osten), 90 (nach Norden), usw:

$$\mathit{orientation}(s_0) = 0$$

- Mit der Orientierung und der aktuellen Position können wir die Zelle bestimmen, zu der wir als nächstes kommen:

$$\forall x, y\ \mathit{locationToward}([x, y], 0) = [x + 1, y]$$
$$\ldots$$

- Wände sind auch da:

$$\forall x, y\ [\mathit{Wall}(x, y) \Leftrightarrow (x = 0 \vee x = 5 \vee y = 0 \vee y = 5)]$$

$\rightsquigarrow$ Damit kann man dann die entsprechenden Nachfolgezustands-Axiome für *forward* und *turn* angeben.

Explorieren

Am wichtigsten ist natürlich das Entdecken nicht offensichtlicher Eigenschaften (*wo ist der WUMPUS?*).

1. Sich merken, wo es stinkt und wo es weht: **Diagnostische Regeln**, die aus Perzepten direkt Eigenschaften ableiten

$$\begin{aligned} \forall l, s\ [\mathit{At}(\mathit{Agent}, l, s) \wedge \mathit{Breeze}(s) &\Rightarrow \mathit{Breezy}(l)] \\ \forall l, s\ [\mathit{At}(\mathit{Agent}, l, s) \wedge \mathit{Stench}(s) &\Rightarrow \mathit{Smelly}(l)] \end{aligned}$$

2. Auf den WUMPUS und die Pits schließen: **Kausale Regeln**, die Perzepte „erklären“

$$\begin{aligned} \forall l_1, l_2, s\ [\mathit{At}(\mathit{Wumpus}, l_1, s) \wedge \mathit{Adjacent}(l_1, l_2) &\Rightarrow \mathit{Smelly}(l_2)] \\ \forall l_1, l_2, s\ [\mathit{At}(\mathit{Pit}, l_1, s) \wedge \mathit{Adjacent}(l_1, l_2) &\Rightarrow \mathit{Breezy}(l_2)] \end{aligned}$$

Diagnostische vs. Kausale Regeln (1)

Kausal: *Ursache* $\Rightarrow$ *Symptom*
Einfach zu formulieren (wenn man die Domäne kennt), aber nicht so einfach zu behandeln.

Beispiel: Um aus der Regel

$$\forall l_1, l_2, s\ [\mathit{At}(\mathit{Wumpus}, l_1, s) \wedge \mathit{Adjacent}(l_1, l_2) \Rightarrow \mathit{Smelly}(l_2)]$$

auf den Ort des WUMPUS zu schließen, müssen wir für alle bis auf ein angrenzendes Feld den WUMPUS ausschließen (durch die Kontraposition der Regel).

Diagnostische vs. Kausale Regeln (2)

Diagnostisch: *Symptom* $\Rightarrow$ *Diagnose*
Einfach zu behandeln, aber oft sehr schwierig zu formulieren!

Beispiel: Jedes Feld, in dem es weder stinkt noch zieht, hat nur benachbarte Felder, die sicher sind:

$$\begin{aligned} \forall l_1, l_2, s, g, u, c\ [&\mathit{Percept}(\mathit{none}, \mathit{none}, g, u, c, s) \wedge \\ &\mathit{At}(\mathit{Agent}, l_1, s) \wedge \\ &\mathit{Adjacent}(l_1, l_2) \\ \Rightarrow\ &OK(l_2)] \end{aligned}$$

Allerdings ist dies zu schwach! Manchmal können auch Felder sicher sein, in deren Nachbarfelder es stinkt oder zieht. Besser sind deshalb kausale Regeln wie:

$$\forall l, s\ [\neg \mathit{At}(\mathit{Wumpus}, l, s) \wedge \neg \mathit{At}(\mathit{Pit}, l, s) \Leftrightarrow OK(l)]$$

Aktionsauswahl

Je nach Zustand können wir die möglichen Aktionen (deren Vorbedingungen erfüllt sind) nach ihrer Wünschbarkeit klassifizieren.

Beispielskala: *Great*, *Good*, *Medium*, *Risky*.

Beispiel: Wenn wir Gold nehmen können oder mit dem Gold die Höhle verlassen können ist das *great*, usw.

$$\forall a, s\ [\mathit{Poss}(\mathit{grab(gold)}, s) \Rightarrow \mathit{Great}(\mathit{grab(gold)}, s)]$$

Wir können dann an Hand der Klassifizierung mögliche Aktionen auswählen (durch die Anfrage $\exists a\, \mathit{Action}(a, \mathtt{t})$).

$$\begin{aligned} \forall a, s\ [\mathit{Great}(a, s) &\Rightarrow \mathit{Action}(a, s)] \\ \forall a, s\ [\mathit{Good}(a, s) \wedge \neg\exists b\, \mathit{Great}(b, s) &\Rightarrow \mathit{Action}(a, s)] \\ &\ldots \end{aligned}$$

Man könnte die Aktionsauswahl natürlich auch direkt in die Aktionen hineincodieren, aber das ist wenig *modular*.

Ausblick: Einsatz logischer Formalismen

- In vielen (aber nicht allen) Fällen ist volle Inferenz in PL1 einfach zu langsam (und deshalb zu unzuverlässig).
- Oft werden für spezielle Anwendungen spezielle (*logikbasierte*) Repräsentationsformalismen entworfen, für die dann spezielle, effiziente Inferenzverfahren benutzt werden können. Beispiele:
  - Terminologische Logiken (auch Beschreibungslogiken) zur Repräsentation von begrifflichem Wissen.
  - Das Allen'sche Intervallkalkül zur Repräsentation qualitativen temporalen Wissens.
  - Handlungsplanung: Statt Situationskalkül, spezialisiertes Kalkül (STRIPS), das die Behandlung des Rahmenproblems einfach ermöglicht.

$\rightsquigarrow$ ... Allgemeinheit vs. Effizienz

$\rightsquigarrow$ Wichtig aber in jedem Fall: logische Semantik!

Grundlagen der KI

10. Handlungsplanung

Planen im Situationskalkül, STRIPS-Formalismus, Nicht-lineares Planen

Wolfram Burgard

Inhalt

- Handlungsplanung vs. Problemlösen
- Planen im Situationskalkül
- Der STRIPS-Formalismus
- Nicht-lineares Planen
- Der POP-Algorithmus
- Variablenbindungen

Handlungsplanung

1. Gegeben eine *initiale Situation*,
2. eine Beschreibung der *Zielbedingungen* und
3. eine Menge von *möglichen Aktionen*,

$\rightarrow$ finde eine Sequenz von Aktionen (einen Handlungsplan), der die initiale Situation in eine Situation überführt, in der die Zielbedingungen gelten.

$\rightsquigarrow$ Unterschied zu Problemlösen und Suche?

$\rightsquigarrow$ Unterschied zu (automatischem) Programmieren?

Ein einfacher, planender Agent

function SIMPLE-PLANNING-AGENT(*percept*) **returns** an *action*
  static: *KB*, a knowledge base (includes action descriptions)
    p, a plan, initially *NoPlan*
    t, a counter, initially 0, indicating time
  local variables: *G*, a goal
    current, a current state description

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept*, *t*))
  current $\leftarrow$ STATE-DESCRIPTION(*KB*, *t*)
  if *p* = *NoPlan* **then**
    G $\leftarrow$ ASK(*KB*, MAKE-GOAL-QUERY(*t*))
    p $\leftarrow$ IDEAL-PLANNER(*current*, *G*, *KB*)
  if *p* = *NoPlan* **or** *p* is empty **then** *action* $\leftarrow$ *NoOp*
  else
    action $\leftarrow$ FIRST(*p*)
    p $\leftarrow$ REST(*p*)
  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  t $\leftarrow$ *t* + 1
  return *action*

Vorgehensweise des Agenten

1. Bestimmung eines Ziels.
2. Berechnung eines Plans, um dieses Ziel vom aktuellen Zustand aus zu erreichen.
3. Ausführen des Plans, bis der Zielzustand erreicht ist.
4. Weiter bei 1.).

Handlungsplanung vs. Problemlösen

Wesentlicher Unterschied: **Explizite, logikbasierte Repräsentation**

- Zustände/Situationen: Durch logische Formeln beschriebene Weltzustände vs. Datenstrukturen
  Der Agent kann damit explizit über die Welt nachdenken und kommunizieren
- Zielbedingungen als logische Formeln vs. Zieltest (Black Box)
  Der Agent kann über seine Ziele auch reflektieren
- Operatoren: Axiome oder Transformation von Formeln vs. Modifikation von Datenstrukturen durch Programme
  Der Agent kann durch Inspektion der Operatoren Information über die Effekte von Aktionen gewinnen.

Unterschied zu Programmierung:
- Logikbasierte Weltbeschreibung.
- Pläne sind meist nur lineare Programme (keine Kontrollstrukturen).

Planen als logische Inferenz (1)

Handlungsplanung kann elegant mit Hilfe des Situationskalküls formalisiert werden.

Initialer Zustand:

$$At(home, s_0) \wedge \neg Have(milk, s_0) \wedge \neg Have(banana, s_0) \wedge \neg Have(drill, s_0)$$

Operatoren (Nachfolgezustandsaxiome):

$$\forall a, s\ Have(milk, do(a, s)) \Leftrightarrow$$
$$\{a = buy(milk) \wedge Poss(buy(milk), s) \vee Have(milk, s) \wedge a \neq drop(milk)\}$$

Zielbedingungen (Anfrage)

$$\exists s\ At(home, s) \wedge Have(milk, s) \wedge Have(banana, s) \wedge Have(drill, s)$$

Wenn der Initialzustand, alle Vorbedingungen und alle Nachfolgezustandsaxiome gegeben sind, liefert der *konstruktive* Beweis der existentiellen Anfrage einen Plan, der das Gewünschte macht.

Planen als logische Inferenz (2)

Die Variablenbindung für s könnte z.B. lauten:

$$do(go(home), do(buy(drill), do(go(hardware\_store), do(buy(banana),$$
$$do(buy(milk), do(go(super\_market), s_0))))))$$

D.h. der Plan (-term) würde dann lauten:

$$\langle go(super\_market), buy(milk), \ldots \rangle$$

Allerdings wäre auch z.B. der folgende Plan korrekt:

$$\langle go(super\_market), buy(milk), drop(milk), buy(milk), \ldots \rangle$$

Deshalb i.allg. zu ineffizient, zu großer Suchraum für den Beweiser!

$\rightsquigarrow$ Spezialisiertes Inferenzsystem für eingeschränkte Repräsentation.
$\rightsquigarrow$ Planungsalgorithmus

Der STRIPS-Formalismus

STRIPS: STanford Research Institute Problem Solver (Planer der frühen 70-er Jahre)

System ist zwar obsolet, der Formalismus wird aber immer noch benutzt.

Weltzustand (inkl. initialer Zustand): Menge von Grundatomen, keine Funktionssymbole außer Konstanten, interpretiert unter CWA (manchmal auch Standardinterpretation, d.h. negative Fakten müssen angegeben werden)

Zielbedingungen: Menge von Grundatomen

Beachte: Keine explizite Zustandsvariable wie im Situationskalkül. Es ist immer nur der aktuelle Weltzustand zugreifbar.

STRIPS-Operatoren

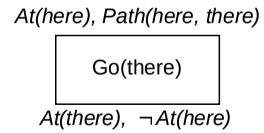
Aktionen sind Tripel, bestehend aus

Aktionsnamen: Funktionsname mit Parametern (wie im Situationenkalkül)

Vorbedingungen: Konjunktion positiver Literale; müssen gelten, damit Aktion ausführbar ist

Effekte: Konjunktion positiver und negativer Literale; positive Literale werden hinzugenommen (ADD Liste), negative gelöscht (DEL Liste) (kein *Frame*-Problem!).

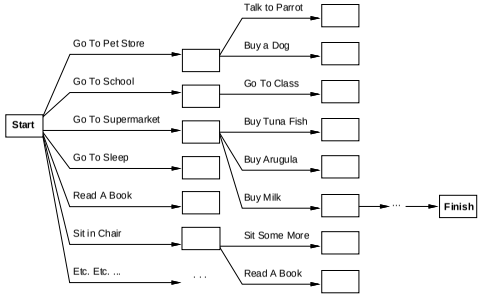
$$Op(\text{Action}: Go(there),\ \text{Precond}: At(here) \wedge Path(here, there),\ \text{Effect}: At(there) \wedge \neg At(here))$$



Suchen im Zustandsraum

Wir könnten jetzt durch den Zustandsraum (die Menge aller Zustände) suchen – und damit Planen auf Suche reduzieren.

Dabei können wir vorwärts suchen (Progressionsplanung):



Alternativ könnten wir ausgehend vom Ziel die Suche rückwärts durchführen (Regressionsplanung).

Möglich, da die Operatoren genug Information zur Verfügung stellen, und *meist effizienter*, da der Verzweigungsfaktor geringer.

Suchen im Planraum

Statt im Zustandraum zu suchen, können wir im Raum aller Pläne suchen.

Der Ausgangszustand ist dann ein **partieller Plan**, der nur einen Start- und einen Zielzustand enthält:

Der Zielzustand ist ein **vollständiger Plan**, der das gegebene Problem löst:

Operatoren im Planraum:

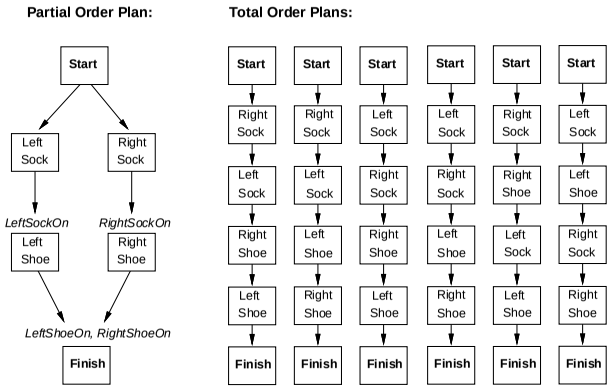
Verfeinerungsoperatoren machen den Plan konkreter (mehr Schritte usw.)

Modifikationsoperatoren modifizieren den Plan (im folgenden nur V.-Op.)

Plan = Sequenz von Aktionen?

Oft ist es aber nicht sinnvoll oder möglich, sich bei der Reihenfolge früh festzulegen (Socken und Schuhe anziehen).

$\rightsquigarrow$ **nicht-lineare** oder **partiell geordnete Pläne** (*least-commitment planning*)



Repräsentation nicht-linearer Pläne

Ein Planschritt = STRIPS-Operator

Ein **Plan** besteht aus

- Menge von **Planschritten** mit partieller Ordnung ($\prec$), wobei $S_i \prec S_j$ gdw. S_i muss vor S_j ausgeführt werden.
- Menge von **Variablenbelegungen** $x = t$, wobei x eine Variable und t eine Konstante oder Variable ist.
- Menge **kausaler Beziehungen**: $S_i \xrightarrow{c} S_j$ bedeutet „S_i erzeugt die Vorbedingung c für S_j“ (impliziert $S_i \prec S_j$)

Lösungen für Planungsprobleme müssen **vollständig** und **konsistent** sein.

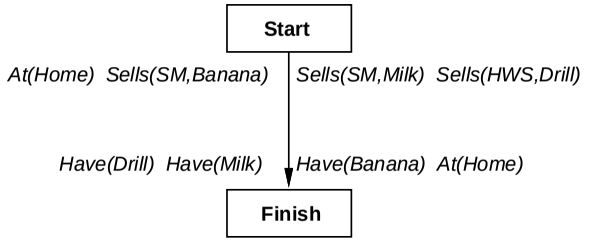
Vollständigkeit und Konsistenz

Vollständiger Plan:
Jede Vorbedingung eines Schritts wird erfüllt:
$\forall S_j$ mit $c \in$ Precond(S_j) und
$\exists S_i$ mit $S_i \prec S_j$ und $c \in$ Effects(S_i) und
für jede Linearisierung des Plans gilt:
$\forall S_k$ mit $S_i \prec S_k \prec S_j$, $\neg c \notin \mathrm{Effect}(S_k)$.

Konsistenter Plan:
wenn $S_i \prec S_j$, dann $S_j \not\prec S_i$ und
wenn $x = A$, dann $x \neq B$ für verschiedene A und B für eine Variable x
(Unique Name Assumption!)

Ein *vollständiger, konsistenter Plan* heißt **Lösung** eines Planungsproblems.

Beispiel



Aktionen:

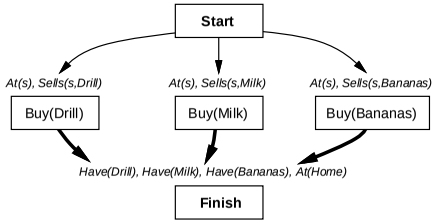
Op (Action: go(there),
Precond: At(here),
Effect: At(there) $\wedge$ $\neg$At(here))

Op (Action: buy(x),
Precond: At(store) $\wedge$ Sells(store,x),
Effect: Have(x))

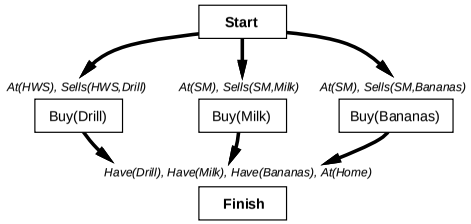
there, here, x, store sind Variablen.

Planverfeinerung (1)

Regressionsplanung: Erfülle die *Have*-Prädikate:



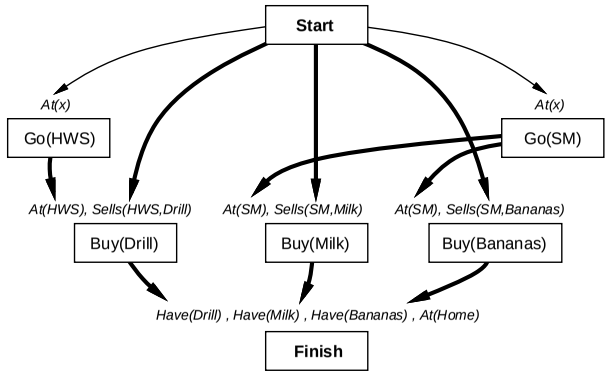
. . . nach Instantiierung der Variablen:



Dünne Pfeile = $\prec$, Fette Pfeile = kausale Beziehungen $+ \prec$

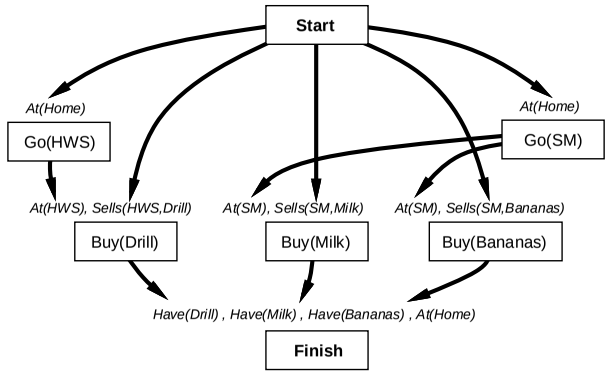
Planverfeinerung (2)

Im richtigen Geschäft kaufen . . .



Planverfeinerung (3)

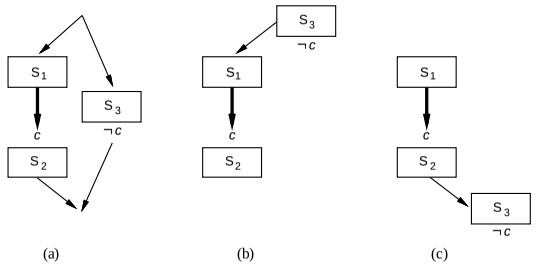
Da muss man erst einmal hin . . .



Beachte: Bisher keine Suche, sondern einfache Rückwärtsverkettung.

Jetzt: *Konflikt!* Wenn wir *go(hws)* gemacht haben, sind wir nicht mehr *At(home)*. Das gleiche gilt für *go(sm)*.

Schutz kausaler Beziehungen



(a) Konflikt: S_3 „bedroht“ die kausale Beziehung zwischen S_1 und S_2

Konfliktlösungen:

(b) Demotion: Den „Bedroher“ vor die kausale Beziehung legen.

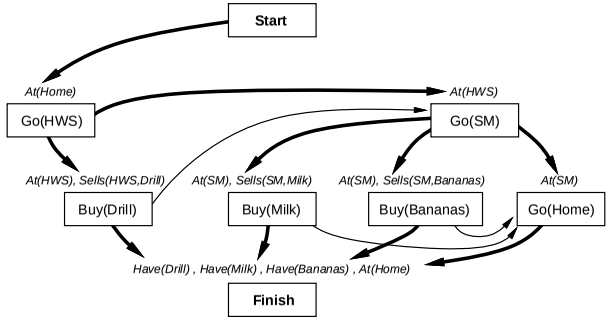
(c) Promotion: Den „Bedroher“ hinter die kausale Beziehung legen.

Eine andere Planverfeinerung ...

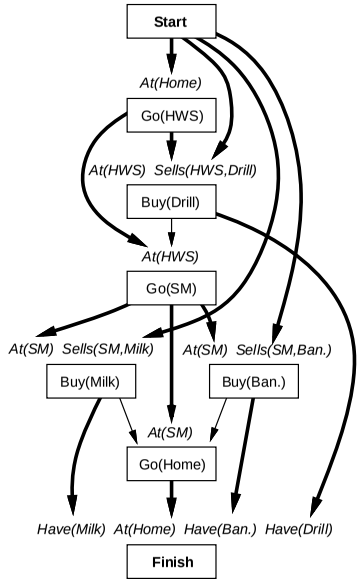
- Wir können den Konflikt nicht durch „Schutz“ auflösen.

$\rightsquigarrow$ Wir haben uns bei einer Planverfeinerung **falsch** entschieden.

- **Alternative:** Bei der Instantiierung von *At(*x*)* in *go(sm)* wählen wir $x = hws$ (mit *kausaler Beziehung*)
- **Beachte:** Dies bedroht das Kaufen des *drills* $\rightsquigarrow$ Promotion von *go(sm)*.



Die vollständige Lösung



Der POP-Algorithmus

function POP(*initial*, *goal*, *operators*) **returns** *plan*

plan $\leftarrow$ MAKE-MINIMAL-PLAN(*initial*, *goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  S_{need}, $c \leftarrow$ SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

---

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
  add S_{add} to STEPS(*plan*)
  add *Start* $\prec S_{add} \prec$ *Finish* to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

Eigenschaften des POP-Algorithmus

Korrektheit: Jedes Resultat des POP-Algorithmus ist ein vollständiger, korrekter Plan.

Vollständigkeit: Falls Breitensuche oder iterative Tiefensuche benutzt wird, findet der Algorithmus eine Lösung, falls eine existiert.

Systematizität: Zwei verschiedene partielle Pläne haben nicht die gleichen *total geordneten Pläne* als Verfeinerungen, falls die partiellen Pläne keine Verfeinerungen voneinander sind (und total geordnete Pläne kausale Beziehungen enthalten).

$\rightsquigarrow$ Instantiierung von Variablen werden nicht behandelt

Variablen

Falls eine Variable in einem Literal eines „Effect“ auftaucht, z.B. $\neg At(x)$, kann dieses Literal eine *potentielle Bedrohung* für eine kausale Beziehung sein. Konfliktauflösung

- durch *Gleichheits-Constraint*, z.B. $x = hws$, um nicht *At(sm)* zu bedrohen;
- durch *Ungleichheits-Constraint* (Spracherweiterung), z.B. $x \neq sm$ (ist aber trickreich);
- später durchführen, wenn Variable instantiiert (macht es schwieriger festzustellen, ob ein Plan eine Lösung ist).

Wir wählen letzteres.

Behandlung von Variablen

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c_{add} as an effect
such that u = UNIFY(c, c_{add}, BINDINGS(*plan*))
if there is no such step
then fail
add u to BINDINGS(*plan*)
add $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
add S_{add} to STEPS(*plan*)
add *Start* $\prec S_{add} \prec$ *Finish* to ORDERINGS(*plan*)

procedure RESOLVE-THREATS(*plan*)

for each $S_i \xrightarrow{c} S_j$ **in** LINKS(*plan*) **do**
for each S_{threat} **in** STEPS(*plan*) **do**
for each c' **in** EFFECT(S_{threat}) **do**
if SUBST(BINDINGS(*plan*), c) = SUBST(BINDINGS(*plan*), $\neg c'$) **then**
choose either
Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
if not CONSISTENT(*plan*)
then fail
end
end
end

Funktioniert, falls Initialzustand variablenfrei und alle Operatoren alle ihre Variablen in den Vorbedingungen benutzen. Ansonsten muss *Solution?* geändert werden.

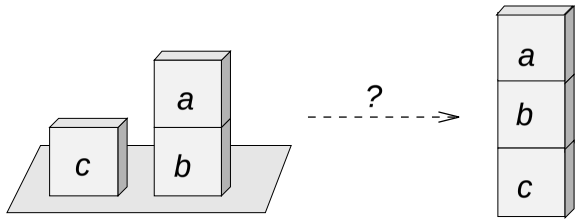
Modellierung in STRIPS

Ähnlich wie bisher auch schon (Problemlösen, PL1), muss man bei der Modellierung von Aufgaben die folgenden Schritte durchführen:

- Entscheiden, worüber man sprechen will
- Ein Vokabular für Bedingungen, Operatoren und Objekte festlegen
- Operatoren kodieren
- Probleminstanzen kodieren

Dann kann ein Planer Lösungen produzieren.

Beispiel: Blockwelt



- Es gibt benannte Blöcke und eine Tischplatte in der Welt.
- Auf der Tischplatte können beliebig viele Blöcke stehen, auf einem Block kann nur ein anderer Block stehen.
- Ein Block kann nur bewegt werden, wenn kein anderer Block auf ihm steht.
- Umstürzen von Blöcken usw. ist nicht erlaubt.

Modellierung der Blockwelt

- Nur die Blöcke als Objekte modellieren, die Tischplatte ist implizit.

$\rightsquigarrow$ Objekte: a, b, c

- Man repräsentiert explizit, ob ein Block bewegt werden kann und ob er auf dem Tisch liegt.

$\rightsquigarrow$ Prädikate:

  - *On*(x, y): x liegt auf y
  - *OnTable*(x): x liegt auf der Tischplatte
  - *Clear*(x): Es liegt nichts auf x

- Es gibt Operatoren, um Blöcke von Blöcken auf andere Blöcke zu bewegen.
- Es gibt Operatoren, um Blöcke vom Tisch auf einen anderen Block zu bewegen und umgekehrt.

$\rightsquigarrow$ Operatoren: *move*(x, y, z), *stack*(x, y), *unstack*(x, y) ...

Zusammenfassung

- Handlungsplanung unterscheidet sich vom Problemlösen dadurch, dass die **Repräsentation flexibler** ist.
- Im Prinzip kann man Planung auf **logische Inferenz** (= *Situationskalkül*) reduzieren; das ist aber sehr ineffizient.
- Statt im Zustandsraum kann man im **Planraum** suchen.
- Das Prinzip der **geringsten Festlegung** (*least commitment*) besagt, dass man während der Suche Entscheidungen nur dann treffen soll, wenn es unbedingt notwendig ist.
- **Nicht-lineares** Planen ist eine Instanz dieses Prinzips.
- Der POP-Algorithmus realisiert nicht-lineares Planen und ist **vollständig** und **korrekt**.

Selber planen lassen ...

Wir haben 4 verschiedene Planungssysteme zur Verfügung:

1. UCPOP (Achtung läuft nicht auf Ultras)
2. Prodigy (sucht im Zustandsraum und generiert lineare Pläne)
3. Graphplan (lernen wir als nächstes kennen)
4. IPP (unser eigenes System)

Hinweise zum Umgang mit den Systemen finden sich unter

http://www.informatik.uni-freiburg.de/~ki/lehre/ws9798/planer.html

Grundlagen der KI

Planen mit Planungsgraphen

Suchraumstrukturierung, Terminierung auf unlösbaren Problemen

Wolfram Burgard

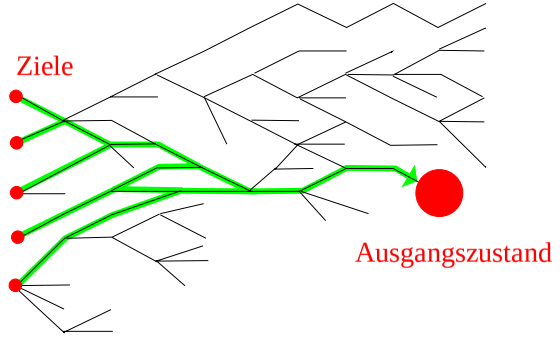
Inhalt

- Suchverfahren im Zustandsraum
- Interferenz von Operatoren
- Planungsgraphen als kompakte Datenstrukturen
- Terminierung auf unlösbaren Problemen
- Extraktion von Plänen in Form von Teilgraphen
- Warum ist Graphplan so schnell?

Strategie des POP-Algorithmus

POP-Planer

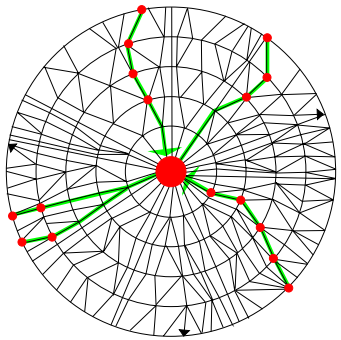
- Suche im Raum der partiellen Pläne
- vom Zielzustand zum Ausgangszustand



Rückwärtssuche im unstrukturierten Suchraum

Graphplan

- Suche im Raum der möglichen Zustände
- in 2 Phasen: vom Ausgangszustand zum Ziel und wieder zurück



Suchraumstrukturierung vom Ausgangszustand zum Ziel
Rückwärtssuche im strukturierten Suchraum

Abhängigkeiten von Operatoren

Operatoren können auf verschiedene Art und Weise miteinander in Beziehung stehen.

```
GEH ?ort1 ?ort2 :ORT
pre: in(?ort1)
eff: ADD in(?ort2) DEL in(?ort1)

GIB ?b :BRIEF ?ort :ORT
pre: habe(?b) empfänger(?b ?ort) in(?ort)
eff: ADD zugestellt(?b ?ort) DEL habe(?b)

NIMM ?b :BRIEF ?ort :ORT
pre: sender(?b ?ort) in(?ort)
eff: ADD habe(?b)
```

GEH erzeugt und zerstört Vorbedingungen für GIB und NIMM

GIB zerstört Effekte von NIMM

Aktionen = Grundinstanzen von Operatoren

Ob eine Abhängigkeit tatsächlich auftritt, kann nur für die konkreten Aktionen entschieden werden.

bn jk prak sind vom Typ ORT und *paket* ist vom Typ BRIEF

GEH(bn jk)
pre: in(bn)
eff: ADD in(jk) DEL in(bn)
▶ GEH(jk bn), GEH(bn prak), GEH(prak bn),
▶ GEH(prak jk), GEH(jk prak)

GIB(paket bn)
pre: habe(paket) empfänger(paket bn) in(bn)
eff: ADD zugestellt(paket bn) DEL habe(paket)
▶ GIB(paket jk), GIB(paket prak)

NIMM(paket bn)
pre: sender(paket bn) in(bn)
eff: ADD habe(paket)
▶ NIMM(paket jk), NIMM(paket prak)

Parallele Ausführung von Aktionen

Aktionen können auch gleichzeitig (im gleichen Zustand = *parallel*) ausgeführt werden, solange sie sich nicht "stören".

NIMM(brief bn) & GIB(paket bn)

Der Robby ist für beide Aktionen im gleichen Raum und eine Tragekapazität oder die Anzahl der freien Hände haben wir nicht modelliert.

NIMM(brief bn) & GIB(paket jk)

Das geht schief! Hier muss der Robby zum gleichen Zeitpunkt an zwei verschiedenen Orten sein.

$\rightsquigarrow$ formale Definition von "sich stören"

Interferenz zwischen Aktionen

Definition 1 *Zwei Aktionen interferieren miteinander gdw. die eine einen Effekt oder eine Vorbedingung der anderen löscht.*

$\rightsquigarrow$ solche Aktionen können niemals im gleichen Zustand ausgeführt werden!

▶ GEH(bn jk), GEH(bn prak)

▶ GEH(jk bn), GEH(jk prak)

▶ GEH(prak jk), GEH(prak bn)

▶ GIB(paket jk), GIB(paket prak)

▶ NIMM(paket bn), NIMM(paket jk)

Pläne, die solche Aktionen parallel ausführen wollen, sind garantiert keine Lösung des Problems.

$\rightsquigarrow$ sehr wirksame Beschränkung des Suchraums

Planungsgraphen

Definition 2 *Ein Planungsgraph $\Pi(N, E)$ zu einem gegebenen Planungsproblem $P(D, O, I, G)$ ist ein gerichteter, in alternierende Ebenen geteilter Graph mit zwei Arten von Knoten*

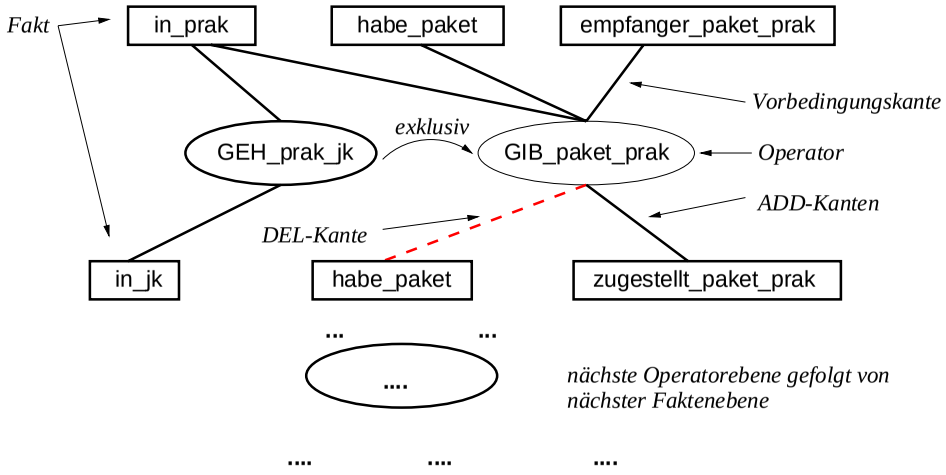
1. *Aktionsknoten A*
2. *Faktenknoten F*

und zwei Arten von Kanten

1. *Vorbedingungskanten $F \times A$*
2. *Effektkanten $A \times F$.*

Jede Ebene enthält nur eine Art von Knoten. Der Graph beginnt und endet mit einer Faktenebene. Kanten verbinden nur Knoten benachbarter Ebenen.

Repräsentation von Planungsgraphen

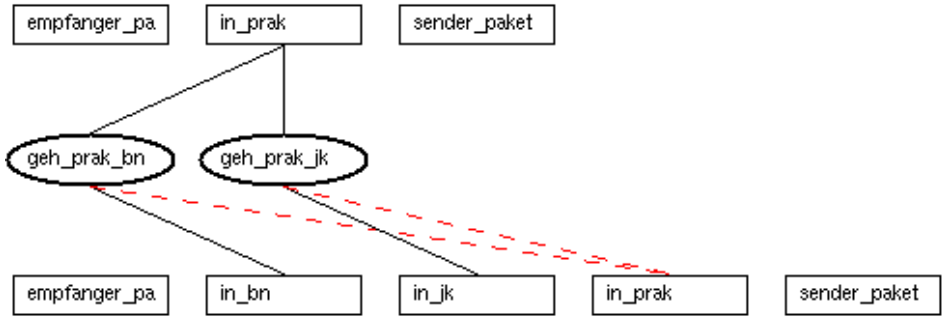


Hinweis: Knotennamen werden oft auf 12 Zeichen verkürzt.

Anwendbarkeit von Aktionen im Planungsgraphen

I: sender(paket bn) in(prak) empfänger(paket jk)

G: zugestellt(paket jk)



Definition 3 *Eine Aktion ist* anwendbar *auf Ebene Q_O gdw. ihre Vorbedingungen in Q_O enthalten sind.*

Definition 4 *Zwei Aktionen sind* exklusiv *voneinander auf Ebene S_0 gdw. sie interferieren.*

Exklusivität von Aktionen und Fakten

Definition 5 *Zwei Fakten p, q sind* exklusiv *voneinander auf Ebene $Q_{i\geq 1}$ gdw. es kein Paar nicht-exklusiver Aktionen s, t auf Ebene S_{i-1} (oder keine einzelne Aktion gibt), die p und q als Effekt haben.*

Kein möglicher Weltzustand kann exklusive Fakten wahr machen.

Definition 6 *Zwei Aktionen s, t haben* konkurrierende Vorbedingungen *gdw. es Fakten $p \in pre(s)$ und $q \in pre(t)$ gibt, die exklusiv sind.*

Definition 7 *Zwei Aktionen s, t sind* exklusiv *voneinander*

- *auf Ebene S_0: gdw. sie interferieren;*
- *auf Ebene $S_{i\geq 1}$: gdw. sie interferieren oder konkurrierende Vorbedingungen haben.*

In keinem möglichen Weltzustand sind exklusive Aktionen gleichzeitig ausführbar.

NO-OPs

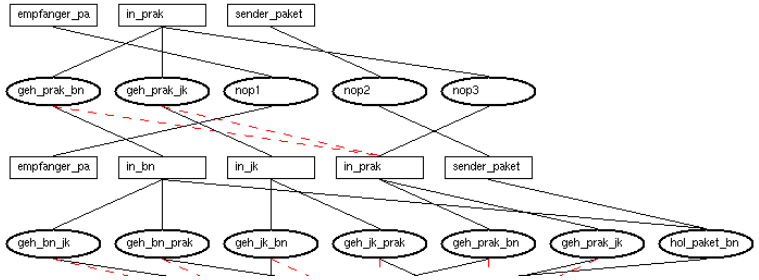
Das Rahmenproblem: Welche Fakten gelten nach Ausführung einer Aktion unverändert weiter?

Annahme: Zunächst erst mal alles! Ob es tatsächlich gilt, lässt sich anhand der Exklusivitätseigenschaft entscheiden.

Wir definieren eine spezielle Art von Operatoren für jeden Fakt f einer Faktenebene:

no-op$_f$
pre: f
eff:f
NO-OPs kopieren alle Fakten einer Ebene auf die folgende Ebene. Sie werden ansonsten genauso behandelt wie andere Aktionen. Es folgt, dass jeder Fakt, der einmal in einer Ebene auftritt, auch auf allen Folge-Ebenen auftreten muss.

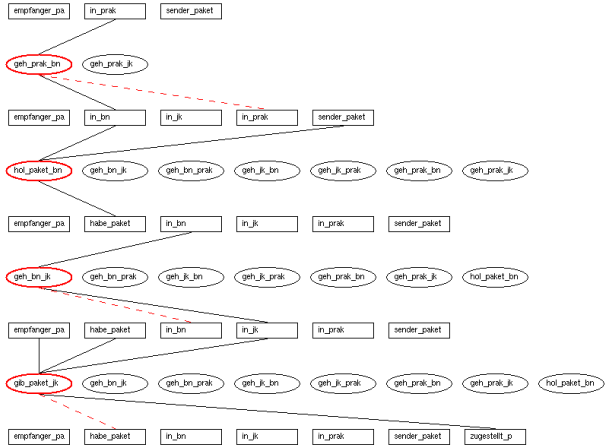
Vollständiger Planungsgraph - 1. Ebene



exklusive Operatoren: geh(prak bn) $\leftrightsquigarrow$ geh(prak jk) nop3
geh(prak jk) $\leftrightsquigarrow$ geh(prak bn) nop3
exklusive Fakten: in(bn) $\leftrightsquigarrow$ in(jk) in(prak)
in(jk) $\leftrightsquigarrow$ in(bn) in(prak)
in(prak) $\leftrightsquigarrow$ in(jk) in(bn)

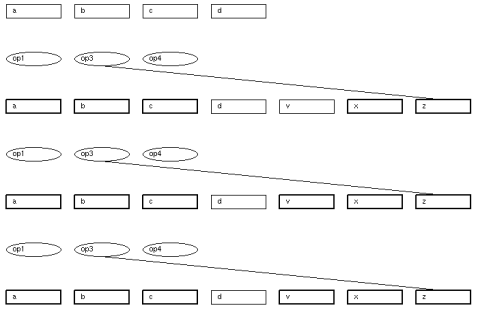
Terminierung

im positiven Fall:
alle Ziele sind erreicht und sie sind nicht exklusiv



Fixpunkt eines Planungsgraphen

Definition 8 *Ein Planungsgraph hat einen Fixpunkt erreicht gdw. 2 Faktenebenen und alle ihre Exklusivitätsbeziehungen identisch sind.*



Das Ziel y fehlt!

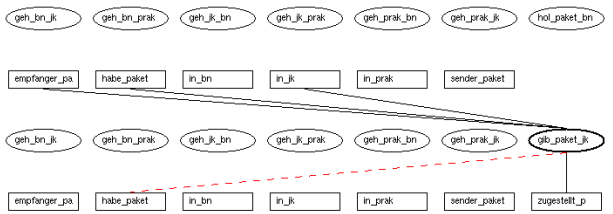
Theorem 1 *Hat ein Graph den Fixpunkt erreicht und ist ein Teilziel nicht in der letzten Faktenebene enthalten oder sind 2 Teilziele exklusiv voneinander, dann ist das Planungsproblem unlösbar.* ▶ *hinreichend, nicht notwendig*

Rückwärtssuche

Starten bei der maximalen Faktenebene. Die Ziele auf dieser Ebene sind gegeben durch unser ursprüngliches Ziel, hier *zugestellt(paket jk)*.

Auswahlpunkt: *minimale* Menge von nicht-exklusiven Operatoren auf Ebene i, die Ziel auf Ebene $i+1$ erreichen

Minimal heißt, dass jeder Operator mindestens ein Teilziel erreicht, das von keinem anderen Operator erreicht wird.



Die Vorbedingungen der Operatoren bilden die Ziele auf der Ebene i.
Backtracking: alle möglichen Operatoren oder eine konstruierte Zielmenge sind exklusiv

Iteratives Expandieren und Suchen

Es konnte kein gültiger Plan aus dem Graphen extrahiert werden.
▶ Solange der Fixpunkt noch nicht erreicht ist, wird eine weitere Ebene erzeugt und die Suche erneut gestartet.

Kann es sein, dass die Ziele nicht-exklusiv sind, aber trotzdem kein Plan gefunden werden kann?
Ja! Nämlich dann, wenn auf einer Zwischenebene ein unmöglicher Zustand durchquert werden müsste.

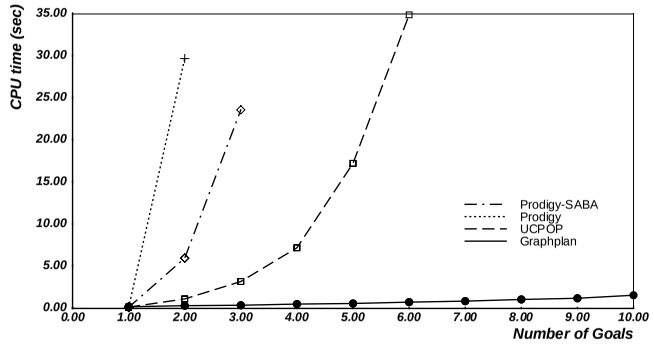
Theorem 2 *Sei i die Ebene, auf der der Graph seinen Fixpunkt erreicht hat. Sei $|G_i^t|$ die Anzahl aller unlösbaren Ziele, die der Planer auf i für einen Graphen der Tiefe t erzeugt hat. Ein Planungsproblem besitzt keine Lösung gdw.*

$$|G_i^t| = |G_i^{t+1}|$$

▶ hinreichend + notwendig

Praktisches Verhalten

Graphplan ist das zur Zeit schnellste Planungssystem auf allen Testbeispielen.



Problem	Graph	Planning	Total
rocket-8	0.16	0	0.16
rocket-10	0.20	0	0.20

Warum?

- Exklusivitätsbeziehungen treten in allen diesen Beispielen hinreichend oft auf.
- Sie können propagiert werden und schränken den Suchraum sehr stark ein.
- Viele Planungsprobleme können mit parallelen Plänen gelöst werden, d.h. Planungsgraphen werden nicht sehr tief.
- Die Beschreibungen der zu lösenden Planungsprobleme enthalten kaum irrelevante Informationen.
  - Objekte: Briefe, die nicht transportiert werden sollen.
  - Fakten: Räume, die während der Zustellung nie durchquert werden müssen.
  - Operatoren: Sortieren oder Umtüten von Post spielen bei der Zustellung niemals eine Rolle.

Grundlagen der KI

11. Verarbeitung unsicheren Wissens

Wahrscheinlichkeitstheorie, bayessche Netze, andere Ansätze

Wolfram Burgard

Inhalt

- Motivation
- Grundlagen der Wahrscheinlichkeitstheorie
- Probabilistische Inferenzen
- bayessche Netze
- Alternative Ansätze

Motivation

- In vielen Fällen ist unser Wissen über die Welt unvollständig (nicht genug Information) oder unsicher (Sensoren sind unzuverlässig).
- Oft sind Gesetzmäßigkeiten nur unvollständig bekannt oder sogar inkorrekt – z.B. das *Qualifikationsproblem*: was sind die Vorbedingungen einer Aktion?
- Wir müssen trotzdem agieren!

$\rightsquigarrow$ Schließen unter Unsicherheit

$\rightarrow$ Nicht-monotones Schließen

$\rightarrow$ Schließen über Eintrittswahrscheinlichkeiten

$\ldots$ und Kosten/Nutzen

Beispiel

- **Ziel**: Um 9:15 Uhr in Freiburg sein, um eine Vorlesung zu halten.
- Es gibt mehrere **Pläne**, um das Ziel zu erreichen:
  - P_1: 7:00 aufstehen, 8:15 den Bus nehmen, 8:30 den Zug $\ldots$
  - P_2: 6:00 aufstehen, 7:15 den Bus nehmen, 7:30 den Zug $\ldots$
  - $\ldots$
- Alle Pläne sind *korrekt*, aber

$\rightsquigarrow$ sie implizieren verschiedene **Kosten** und verschiedene **Wahrscheinlichkeiten**, das Ziel *tatsächlich* zu erreichen.

$\rightarrow$ P_2 wäre der Plan der Wahl, da Vorlesungen halten (verglichen mit z.B. Gremiensitzungen) sehr wichtig ist, und die Erfolgsrate bei P_1 nur bei ca. 90–95% liegt.

Unsicherheiten bei Regeln (1)

Beispiel: Diagnose-Expertensystem für Zahnärzte.

$$\forall p\,[Symptom(p, toothache) \Rightarrow Disease(p, cavity)]$$

$\rightsquigarrow$ Diese Regel ist *inkorrekt*! Besser:

$$\forall p\,[Symptom(p, toothache) \Rightarrow Disease(p, cavity) \vee Disease(p, gum\_disease) \vee \ldots]$$

. . . aber wir kennen gar nicht alle Ursachen

Vielleicht besser die *kausale* Regel?

$$\forall p\,[Disease(p, cavity) \Rightarrow Symptom(p, toothache)]$$

$\rightsquigarrow$ Auch nicht korrekt!

Unsicherheiten bei Regeln (2)

Probleme mit der Logik:

- Wir können nicht alle möglichen Ursachen aufzählen, und selbst wenn . . .
- Wir kennen nicht die Gesetzmäßigkeiten (in der Medizin)
- . . . und selbst wenn, bleibt Unsicherheit über den Patienten bestehen (Karies und Zahnschmerzen zufällig gleichzeitig, nicht alle Untersuchungen)

$\rightsquigarrow$ ohne perfektes Wissen keine korrekten logischen Regeln!

Unsicherheit bei Fakten

Nehmen wir an, wir wollten die Lokalisation eines Roboters durch (unveränderliche) Landmarken unterstützen. Aus dem Vorhandensein von Landmarken können wir auf den Raum schließen.

Problem: Sensoren kann sind ungenau.

$\rightarrow$ Aus der Tatsache, dass eine Landmarke erkannt wurde, kann man nicht mit Sicherheit schließen, dass der Roboter sich in dem entsprechenden Raum befindet.

$\rightarrow$ Gleiches gilt, falls eine Landmarke nicht wahrgenommen wird.

$\rightarrow$ Es wird lediglich die Wahrscheinlichkeit erhöht oder erniedrigt.

Grade der Überzeugung und Wahrscheinlichkeitstheorie (1)

- Wir (oder andere Agenten) sind von Regeln und Fakten nur bis zu einem gewissen Grad überzeugt.
- Eine Möglichkeit, den **Grad der Überzeugung** auszudrücken ist, *Wahrscheinlichkeiten* zu benutzen.
- Der Agent ist von der Sensorinformation zu 0.9 überzeugt $=$ in 9 von 10 Fällen ist die Information richtig (glaubt der Agent).
- Wahrscheinlichkeiten fassen die „Unsicherheit“ bedingt durch Unwissen zusammen.
- Wahrscheinlichkeiten sind nicht mit *Vagheit* zu verwechseln. Das Prädikat *groß* ist **vage**; die Aussage *„ein Mann hat eine Größe von 1.75–1.85m“* ist **unsicher**.

Rationale Entscheidungen unter Unsicherheit

- Wir haben verschiedene **Aktionen** (oder *Pläne*) zur Auswahl
- Diese können zu verschiedenen Ergebnissen führen mit verschiedenen **Wahrscheinlichkeiten**
- Die *Aktionen* verursachen verschiedene (subjektive) **Kosten**
- Die *Ergebnisse* haben verschiedenen (subjektiven) **Nutzen**
- Rational wäre es, die Aktion zu wählen, die den größten **zu erwartenden Gesamtnutzen** hat!

$\rightsquigarrow$ **Entscheidungstheorie** = **Nutzentheorie** + **Wahrscheinlichkeitstheorie**

Entscheidungstheoretischer Agent

function DT-AGENT(*percept*) **returns** an *action*
static: a set probabilistic beliefs about the state of the world

calculate updated probabilities for current state based on
available evidence including current percept and previous action
calculate outcome probabilities for actions,
given action descriptions and probabilities of current states
select *action* with highest expected utility
given probabilities of outcomes and utility information
return *action*

Entscheidungstheorie: Ein Agent ist rational genau dann, wenn er die Aktion wählt, die den größten erwarteten Nutzen gemittelt über alle möglichen Ergebnisse von Aktionen hat.

Unbedingte Wahrscheinlichkeiten (1)

$P(A)$ bezeichnet die **unbedingte** oder **a priori** Wahrscheinlichkeit, dass A eintreten wird im Fall, dass *keine* zusätzliche Information verfügbar ist, z.B.

$$P(Cavity) = 0.1$$

Cavity ist eine Proposition. A priori Wahrscheinlichkeiten gewinnt man durch statistische Analyse oder aus allgemeinen Regeln.

Unbedingte Wahrscheinlichkeiten (2)

Im allgemeinen kann eine **Zufallsvariable** nicht nur die Werte *wahr* und *falsch* sondern mehrere Werte annehmen:

$$\begin{aligned} P(\mathit{Weather} = \mathit{Sunny}) &= 0.7 \\ P(\mathit{Weather} = \mathit{Rain}) &= 0.2 \\ P(\mathit{Weather} = \mathit{Cloudy}) &= 0.08 \\ P(\mathit{Weather} = \mathit{Snow}) &= 0.02 \\ P(\mathit{Headache} = \mathit{TRUE}) &= 0.1 \end{aligned}$$

- Propositionen können auch Gleichungen über Zufallsvariablen enthalten.
- Logische Konnektoren können zur Bildung von Propositionen verwendet werden, z.B. $P(\mathit{Cavity} \wedge \neg \mathit{Insured}) = 0.06$.

Unbedingte Wahrscheinlichkeiten (3)

$\boxed{\mathbf{P}(X)}$ bezeichnet den Vektor der Wahrscheinlichkeiten für den (geordneten) Wertebereich der Zufallsvariable X:

$$\begin{aligned}\mathbf{P}(\textit{Headache}) &= \langle 0.1, 0.9\rangle \\ \mathbf{P}(\textit{Weather}) &= \langle 0.7, 0.2, 0.08, 0.02\rangle\end{aligned}$$

definieren die Wahrscheinlichkeitsverteilung der Zufallsvariablen *Headache* und *Weather*.

$\mathbf{P}(\textit{Headache}, \textit{Weather})$ ist eine 4×2 Tabelle von Wahrscheinlichkeiten aller Kombinationen der Werte einer Zufallsvariablen.

	Headache = TRUE	Headache = FALSE
Weather = Sunny	$P(W = \textit{Sunny} \wedge \textit{Headache})$	$P(W = \textit{Sunny} \wedge \neg\textit{Headache})$
Weather = Rain		
Weather = Cloudy		
Weather = Snow		

Bedingte Wahrscheinlichkeiten (1)

Neue Information kann die Wahrscheinlichkeit ändern.

Beispiel: Die Wahrscheinlichkeit von Zahnlöchern erhöht sich, wenn man weiß, dass der Patient von Zahnschmerzen hat.

Liegt Zusatzinformation vor, darf nicht mehr mit a priori Wahrscheinlichkeiten gerechnet werden!

$\boxed{P(A \mid B)}$ bezeichnet die **bedingte** oder **a posteriori** Wahrscheinlichkeit von A *gegeben* die *alleinige* Beobachtung (die *Evidenz*) B:

$$P(\textit{Cavity} \mid \textit{Toothache}) = 0.8$$

$\boxed{\mathbf{P}(X \mid Y)}$ ist die Tabelle aller bedingter Wahrscheinlichkeiten über alle Werte von X und Y.

Bedingte Wahrscheinlichkeiten (2)

$\mathbf{P}(\textit{Weather} \mid \textit{Headache})$ ist eine 4×2 Tabelle von bedingten Wahrscheinlichkeiten aller Kombinationen der Werte einer Zufallsvariablen.

	Headache = TRUE	Headache = FALSE
Weather = Sunny	$P(W = \textit{Sunny} \mid \textit{Headache})$	$P(W = \textit{Sunny} \mid \neg\textit{Headache})$
Weather = Rain		
Weather = Cloudy		
Weather = Snow		

Bedingte Wahrscheinlichkeiten ergeben sich aus unbedingten Wahrscheinlichkeiten (falls $P(B) > 0$) (**per Definition**):

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$

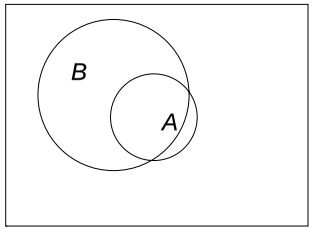
Bedingte Wahrscheinlichkeiten (3)

$\mathbf{P}(X, Y) = \mathbf{P}(X \mid Y)\mathbf{P}(Y)$ entspricht einem Gleichungssystem:

$$\begin{aligned}P(W = \textit{Sunny} \wedge \textit{Headache}) &= P(W = \textit{Sunny} \mid \textit{Headache})P(\textit{Headache}) \\ P(W = \textit{Rain} \wedge \textit{Headache}) &= P(W = \textit{Rain} \mid \textit{Headache})P(\textit{Headache}) \\ \vdots &= \vdots \\ P(W = \textit{Snow} \wedge \neg\textit{Headache}) &= P(W = \textit{Snow} \mid \neg\textit{Headache})P(\neg\textit{Headache})\end{aligned}$$

Bedingte Wahrscheinlichkeiten (4)

$$P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$$



- Produktregel: $P(A \wedge B) = P(A \mid B)P(B)$
- Analog: $P(A \wedge B) = P(B \mid A)P(A)$
- A und B heißen **unabhängig** voneinander, falls $P(A \mid B) = P(A)$ und $P(B \mid A) = P(B)$. Dann (und nur dann) gilt $P(A \wedge B) = P(A)P(B)$

Axiomatische Wahrscheinlichkeitstheorie

Eine Funktion P von aussagenlogischen Formeln in die Menge $[0, 1]$ ist ein **Wahrscheinlichkeitsmaß**, falls für alle Aussagen A, B gilt:

1. $0 \leq P(A) \leq 1$
2. $P(True) = 1$
3. $P(False) = 0$
4. $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

Alle anderen Eigenschaften lassen sich aus diesen Axiomen ableiten, z.B.

$$P(\neg A) = 1 - P(A)$$

folgt aus $P(A \vee \neg A) = 1$ und $P(A \wedge \neg A) = 0$.

Wieso sind die Axiome sinnvoll?

- Wenn P eine *objektiv* beobachtbare Wahrscheinlichkeit bezeichnet, machen die Axiome natürlich Sinn.
- Aber wieso sollte ein Agent diese Axiome beachten, wenn er den *Grad seiner Überzeugung* modelliert?

$\rightsquigarrow$ *Objektive* vs. *subjektive* Wahrscheinlichkeiten

Die Axiome schränken die Menge der Überzeugungen ein, die ein Agent aufrechterhalten kann.

Eines der überzeugendsten Argumente, warum subjektive Überzeugungen die Axiome respektieren sollten, wurde 1931 von de Finetti gegeben. Es basiert auf dem Zusammenhang zwischen Aktionen und dem Grad der Überzeugung.

$\rightsquigarrow$ Sind die Überzeugungen widersprüchlich, dann wird der Agent auf lange Sicht in seiner Umwelt scheitern!

Das Wettmodell (1)

Agent 1 hat die Überzeugung $P(A) = 0.4$.
Agent 2 kann für oder gegen A wetten, sein Einsatz muss jedoch konsistent mit der Überzeugung von Agent 1 sein.

Beispiel: Agent 2 setzt 4 zu 6 auf A, d.h. tritt A auf, muss Agent 1 den Betrag von 6 Pf. an Agent 2 zahlen, sonst zahlt Agent 2 den Betrag von 4 Pf. an Agent 1.

$\rightsquigarrow$ Agent 1 muss diese Wette akzeptieren (fair)

Eine Wettstrategie ist eine Menge von Wetten auf Ereignisse.

Annahme: Agent 1 habe die folgenden Grade von Überzeugungen:

$$\begin{aligned} P(A) &= 0.4 \\ P(B) &= 0.3 \\ P(A \wedge B) &= 0.0 \\ P(A \vee B) &= 0.8 \\ \Longrightarrow P(A \vee B) &\neq P(A) + P(B) - P(A \wedge B) \end{aligned}$$

Das Wettmodell (2)

Agent 2 hat Wettstrategie $A, B, \neg(A \vee B)$ und setzt 4 zu 6 auf A, 3 zu 7 auf B und 2 zu 8 auf $\neg(A \vee B)$:

Agent 1		Agent 2		Outcome for Agent 1			
Proposition	Belief	Bet	Stakes	$A \wedge B$	$A \wedge \neg B$	$\neg A \wedge B$	$\neg A \wedge \neg B$
A	0.4	A	4 to 6	-6	-6	4	4
B	0.3	B	3 to 7	-7	3	-7	3
$A \vee B$	0.8	$\neg(A \vee B)$	2 to 8	2	2	2	-8
				-11	-1	-1	-1

Wegen des inkonsistenten Beliefs verliert Agent 1 in allen möglichen Situationen.

Bei der Wettstrategie $A, B, (A \vee B)$ gewinnt Agent 1 in der Situation $\neg A \wedge \neg B$.

Verbundwahrscheinlichkeit

Wahrscheinlichkeit, die ein Agent jeder Proposition in der Domäne zuordnet.

Ein **atomares Ereignis** ist eine Zuweisung von Werten an alle Zufallsvariablen $X_1, \ldots, X_n$ (= vollständige Spezifikation eines Zustands).

Beispiel: Seien X, Y boolesche Variablen. Dann gibt es die folgenden 4 atomaren Ereignisse: $X \wedge Y$, $X \wedge \neg Y$, $\neg X \wedge Y$, $\neg X \wedge \neg Y$.

Die **Verbundwahrscheinlichkeitsverteilung** $\mathbf{P}(X_1, \ldots, X_n)$ weist jedem *atomaren Ereignis* eine Wahrscheinlichkeit zu:

	Toothache	¬Toothache
Cavity	0.04	0.06
¬ Cavity	0.01	0.89

Da alle atomaren Ereignisse disjunkt sind, ist die Summe über alle Felder 1 (Disjunktion der Ereignisse). Die Konjunktion ist notwendigerweise falsch.

Rechnen mit der Verbundwahrscheinlichkeit

Alle interessanten Wahrscheinlichkeiten lassen sich aus der Verbundwahrscheinlichkeit errechnen, indem wir sie als Disjunktion von atomaren Ereignissen formulieren.

Beispiele:

$$\begin{aligned} P(Cavity \vee Toothache) &= P(Cavity \wedge Toothache) \\ &+ P(\neg Cavity \wedge Toothache) \\ &+ P(Cavity \wedge \neg Toothache) \end{aligned}$$

Unbedingte Wahrscheinlichkeiten erhält man durch Aufsummieren von Zeile oder Spalte:

$$P(Cavity) = P(Cavity \wedge Toothache) + P(Cavity \wedge \neg Toothache)$$

$$P(Cavity \mid Toothache) = \frac{P(Cavity \wedge Toothache)}{P(Toothache)} = \frac{0.04}{0.04 + 0.01} = 0.80$$

Probleme mit der Verbundwahrscheinlichkeit

Aus der Verbundwahrscheinlichkeit lassen sich alle Wahrscheinlichkeiten einfach ermitteln.

Allerdings umfasst die Verbundwahrscheinlichkeit k^n Werte, wenn es n Zufallsvariablen mit k Werten gibt.

$\rightsquigarrow$ Schwierig darzustellen

$\rightsquigarrow$ Schwierig zu ermitteln

Fragen:

1. Gibt es eine **dichtere** Darstellung von Verbundwahrscheinlichkeiten?
2. Gibt es eine **effiziente** Methode, diese Darstellung zu verarbeiten?

I.allg. nicht, aber in vielen Fällen geht es. Moderne Systeme arbeiten direkt mit bedingten Wahrscheinlichkeiten (Diagnose-Kausalität) und machen Annahmen über die Unabhängigkeit von Variablen, um Rechnungen zu vereinfachen.

Die Bayessche Regel

Wir wissen (Produktregel):

$$P(A \wedge B) = P(A \mid B)P(B) \qquad \text{und} \qquad P(A \wedge B) = P(B \mid A)P(A)$$

Durch Gleichsetzen der rechten Seiten folgt:

$$P(A \mid B)P(B) = P(B \mid A)P(A)$$

$$\Longrightarrow \boxed{P(A \mid B) = \frac{P(B|A)P(A)}{P(B)}}$$

Für mehrwertige Variablen (Menge von Gleichungen):

$$\mathbf{P}(Y \mid X) = \frac{\mathbf{P}(X \mid Y)\mathbf{P}(Y)}{\mathbf{P}(X)}$$

Verallgemeinerung (bzgl. Hintergrundevidenzen):

$$\mathbf{P}(Y \mid X, E) = \frac{\mathbf{P}(X \mid Y, E)\mathbf{P}(Y \mid E)}{\mathbf{P}(X \mid E)}$$

Anwendung der Bayesschen Regel

$$\begin{aligned} P(\textit{Toothache} \mid \textit{Cavity}) &= 0.4 \\ P(\textit{Cavity}) &= 0.1 \\ P(\textit{Toothache}) &= 0.05 \\ P(\textit{Cavity} \mid \textit{Toothache}) &= \frac{0.4 \times 0.1}{0.05} = 0.8 \end{aligned}$$

Warum nicht gleich $P(Cavity \mid Toothache)$ schätzen?

$P(Toothache \mid Cavity)$ (kausal) ist robuster als $P(Cavity \mid Toothache)$ (diagnostisch):

- $P(\textit{Toothache} \mid \textit{Cavity})$ unabhängig von den a priori Wahrscheinlichkeiten $P(\textit{Toothache})$ und $P(\textit{Cavity})$.
- Nimmt $P(Cavity)$ bei einer Karies-Epidemie zu, so bleibt $P(Toothache \mid Cavity)$ unverändert, während sich $P(Toothache)$ und $P(Cavity \mid Toothache)$ proportional ändern werden.

Relative Wahrscheinlichkeit

Annahme: Wir wollen auch die Wahrscheinlichkeit der Diagnose *GumDisease* betrachten.

$$\begin{aligned} P(\textit{Toothache} \mid \textit{GumDisease}) &= 0.7 \\ P(\textit{GumDisease}) &= 0.02 \end{aligned}$$

Welche Diagnose ist wahrscheinlicher?

$$P(C \mid T) = \frac{P(T|C)P(C)}{P(T)} \qquad \text{oder} \qquad P(G \mid T) = \frac{P(T|G)P(G)}{P(T)}$$

Wenn uns nur die **relative Wahrscheinlichkeit** interessiert, brauchen wir $P(T)$ nicht zu schätzen:

$$\begin{aligned} \frac{P(C \mid T)}{P(G \mid T)} &= \frac{P(T \mid C)P(C)}{P(T)} \times \frac{P(T)}{P(T \mid G)P(G)} = \frac{P(T \mid C)P(C)}{P(T \mid G)P(G)} \\ &= \frac{0.4 \times 0.1}{0.7 \times 0.02} = 28.57 \end{aligned}$$

$\rightsquigarrow$ Wichtig, um mögliche Diagnosen auszuschließen.

Normalisierung (1)

Wenn wir die absolute Wahrscheinlichkeit von $P(C \mid T)$ bestimmen wollen und $P(T)$ nicht kennen, können wir auch eine *vollständige* Fallanalyse durchführen (z.B. für C und $\neg C$) und den Zusammenhang $P(C \mid T) + P(\neg C \mid T) = 1$ (hier boolesche Variable) ausnutzen:

$$\begin{aligned} P(C \mid T) &= \frac{P(T \mid C)P(C)}{P(T)} \\ P(\neg C \mid T) &= \frac{P(T \mid \neg C)P(\neg C)}{P(T)} \\ P(C \mid T) + P(\neg C \mid T) &= \frac{P(T \mid C)P(C)}{P(T)} + \frac{P(T \mid \neg C)P(\neg C)}{P(T)} \\ P(T) &= P(T \mid C)P(C) + P(T \mid \neg C)P(\neg C) \end{aligned}$$

Normalisierung (2)

Durch Einsetzen in die oberste Gleichung:

$$P(C \mid T) = \frac{P(T \mid C)P(C)}{P(T \mid C)P(C) + P(T \mid \neg C)P(\neg C)}$$

Für mehrwertige Zufallsvariablen:

$$\mathbf{P}(Y \mid X) = \alpha \mathbf{P}(X \mid Y)\mathbf{P}(Y)$$

wobei α eine **Normalisierungskonstante** ist, welche die Werte in $\mathbf{P}(Y \mid X)$ zu 1 aufsummieren lässt, z.B. $\alpha(1,1,3) = (\frac{1}{5}, \frac{1}{5}, \frac{3}{5})$.

Beispiel

Ihr Arzt hat einen Test T mit Ihnen durchgeführt, der eine sehr seltene Krankheit D (1 in 10000) zu 99% korrekt diagnostiziert (1% falsche positive & 1% falsche negative Ergebnisse). Der Test war positiv.

Was bedeutet das für Sie?

$$P(D \mid T) = \frac{P(T|D)P(D)}{P(T)} = \frac{P(T|D)P(D)}{P(T|D)P(D)+P(T|\neg D)P(\neg D)}$$

$$P(D) = 0.0001 \qquad P(T \mid D) = 0.99 \qquad P(T \mid \neg D) = 0.01$$

$$P(D \mid T) = \frac{0.99 \times 0.0001}{0.99 \times 0.0001 + 0.01 \times 0.9999} = \frac{0.000099}{0.000099 + 0.009999} = \frac{0.000099}{0.010088} \approx 0.01$$

Moral: Wenn die Testungenauigkeit sehr viel größer als die Häufigkeit der Krankheit ist, ist ein positives Ergebnis nicht sehr bedrohlich.

Multiple Evidenzen (1)

Nach der Frage nach den Zahnschmerzen hat der Zahnarzt etwas aus den Zähnen herausgeholt (*Catch*) und hat mit der Bayesschen Regel berechnet:

$$P(\mathit{Cavity} \mid \mathit{Catch}) = 0.95$$

Aber was bringt die kombinierte Evidenz?

Mit der Bayesschen Regel könnte er ermitteln:

$$P(\mathit{Cav} \mid \mathit{Tooth} \wedge \mathit{Catch}) = \frac{P(\mathit{Tooth} \wedge \mathit{Catch} \mid \mathit{Cav}) \times P(\mathit{Cav})}{P(\mathit{Tooth} \wedge \mathit{Catch})}$$

Multiple Evidenzen (2)

Problem: Er braucht $P(\mathit{Tooth} \wedge \mathit{Catch} \mid \mathit{Cav})$, d.h. Diagnosewissen für alle Kombinationen von Symptomen im allgemeinen Fall.

Besser ist es, Evidenzen mit Hilfe der Evidenzenregel schrittweise hinzuzunehmen.

$$\mathbf{P}(Y \mid X, E) = \frac{\mathbf{P}(X \mid Y, E)\mathbf{P}(Y \mid E)}{\mathbf{P}(X \mid E)}$$

Mit einer bestimmten a priori Wahrscheinlichkeit hat der Patient ein Loch: $P(\mathit{Cav})$. Er berichtet von Zahnschmerzen (Bayessche Regel):

$$P(\mathit{Cav} \mid \mathit{Tooth}) = P(\mathit{Cav}) \times \frac{P(\mathit{Tooth} \mid \mathit{Cav})}{P(\mathit{Tooth})} \qquad (1)$$

Multiple Evidenzen (3)

$$\boxed{P(\mathit{Cav} \mid \mathit{Tooth})} \quad = \quad P(\mathit{Cav}) \times \frac{P(\mathit{Tooth} \mid \mathit{Cav})}{P(\mathit{Tooth})} \qquad (2)$$

Die Untersuchung ergibt *Catch*, also

$$P(\mathit{Cav} \mid \mathit{Catch}, \mathit{Tooth}) = \frac{P(\mathit{Catch} \mid \mathit{Cav}, \mathit{Tooth})}{P(\mathit{Catch} \mid \mathit{Tooth})} \times \boxed{P(\mathit{Cav} \mid \mathit{Tooth})} \qquad (3)$$

(2) in (3) einsetzen ergibt

$$P(\mathit{Cav} \mid \mathit{Catch}, \mathit{Tooth}) = P(\mathit{Cav}) \times \frac{P(\mathit{Tooth} \mid \mathit{Cav})}{P(\mathit{Tooth})} \times \frac{\boxed{P(\mathit{Catch} \mid \mathit{Cav}, \mathit{Tooth})}}{P(\mathit{Catch} \mid \mathit{Tooth})}$$

Multiple Evidenzen (4)

Annahme **bedingter Unabhängigkeit** von *Toothache* und *Catch* gegeben *Cavity* (vereinfachtes Diagnosewissen):

$$\begin{aligned} \boxed{P(\mathit{Catch} \mid \mathit{Cav}, \mathit{Tooth})} &= P(\mathit{Catch} \mid \mathit{Cav}) \\ P(\mathit{Tooth} \mid \mathit{Cav}, \mathit{Catch}) &= P(\mathit{Tooth} \mid \mathit{Cav}) \end{aligned}$$

$$P(\mathit{Cav} \mid \mathit{Catch}, \mathit{Tooth}) = P(\mathit{Cav}) \times \frac{P(\mathit{Tooth} \mid \mathit{Cav})}{P(\mathit{Tooth})} \times \frac{P(\mathit{Catch} \mid \mathit{Cav})}{\boxed{P(\mathit{Catch} \mid \mathit{Tooth})}}$$

Multiple Evidenzen (5)

$$P(\mathit{Cav} \mid \mathit{Catch}, \mathit{Tooth}) = P(\mathit{Cav}) \times \frac{P(\mathit{Tooth} \mid \mathit{Cav})}{P(\mathit{Tooth})} \times \frac{P(\mathit{Catch} \mid \mathit{Cav})}{\boxed{P(\mathit{Catch} \mid \mathit{Tooth})}}$$

Wie sollen wir $P(\mathit{Catch} \mid \mathit{Tooth})$ bestimmen?

Beachte die Nenner in den Brüchen (Produktregel!):

$$P(\mathit{Tooth}) \times P(\mathit{Catch} \mid \mathit{Tooth}) = P(\mathit{Tooth} \wedge \mathit{Catch})$$

ist ein Normalisierungsfaktor, wenn wir $P(\mathit{Cav} \mid \mathit{Tooth} \wedge \mathit{Catch})$ bestimmen wollen.

Diesen können wir eliminieren, sofern wir ebenfalls $P(\mathit{Catch} \mid \neg \mathit{Cav})$ und $P(\mathit{Tooth} \mid \neg \mathit{Cav})$ kennen.

Zusammenfassung Multiple Evidenzen

Mehrfache Evidenzen können durch Reduktion auf a priori Wahrscheinlichkeiten und bedingte Wahrscheinlichkeiten für eine Evidenz berechnet werden (unter Annahme der Unabhängigkeit).

Allgemeine Kombinationsregel, falls X und Y gegeben Z bedingt unabhängig sind:

$$\mathbf{P}(Z \mid X, Y) = \alpha \mathbf{P}(Z) \mathbf{P}(X \mid Z) \mathbf{P}(Y \mid Z)$$

wobei α Normalisierungskonstante ist.

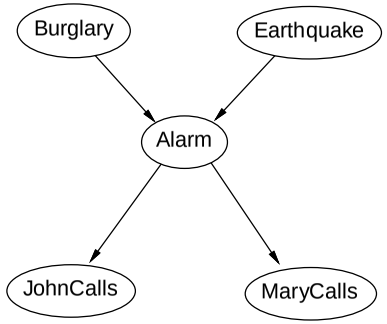
Zusammenfassung

- **Unsicherheit** ist unvermeidbar in komplexen und dynamischen Welten, in denen Agenten zur Ignoranz gezwungen sind.
- **Wahrscheinlichkeiten** formulieren die Unfähigkeit eines Agenten, eine definitive Entscheidung zu fällen. Sie drücken den Grad seiner Überzeugung aus.
- **Bedingte** und **unbedingte** Wahrscheinlichkeiten können über Propositionen formuliert werden.
- Verletzt ein Agent die wahrscheinlichkeitstheoretischen **Axiome**, so wird er unter bestimmten Umständen irrationales Verhalten zeigen.
- Die **Bayessche Regel** ermöglicht es, unbekannte Wahrscheinlichkeiten aus bekannten Wahrscheinlichkeiten zu berechnen.
- **Multiple Evidenzen** können bei bedingter Unabhängigkeit effektiv in die Berechnung einbezogen werden.

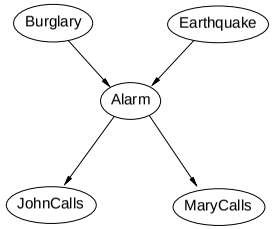
Bayessche Netzwerke

(auch *belief networks, probabilistic networks, causal networks*)

1. Die *Zufallsvariablen* bilden die **Knoten**.
2. **Gerichtete Kanten** zwischen Knoten symbolisieren *direkten Einfluss*.
3. Mit jedem Knoten ist eine **Tabelle der bedingten Wahrscheinlichkeiten** (CPT) assoziiert, die den Effekt der **Eltern** auf den Knoten quantifiziert.
4. Der Graph ist azyklisch (ein DAG).



Die Bedeutung bayesscher Netze



- *Alarm* hängt von *Burglary* und *Earthquake* ab.
- *MaryCalls* hängt nur von *Alarm* ab.

$$\mathbf{P}(\mathit{MarryCalls} \mid \mathit{Alarm}, \mathit{Burglary}) = \mathbf{P}(\mathit{MarryCalls} \mid \mathit{Alarm})$$

$\rightsquigarrow$ Bayessche Netze können als Menge von Unabhängigkeitsannahmen aufgefasst werden.

Bayessche Netzwerke und die Verbundwahrscheinlichkeit

Bayessche Netzwerke können auch als dichte Repräsentation der Verbundwahrscheinlichkeit aufgefasst werden.

Seien alle Knoten angeordnet (so dass die Ordnung die Pfeile im Netz nicht verletzt wird): $X_1, \ldots X_n$. Seien $x_1, \ldots, x_n$ Werte der Variablen. Mit der Produktregel gilt:

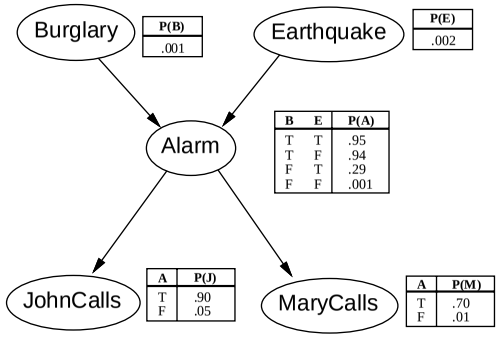
$$\begin{aligned} P(x_1, \ldots, x_n) &= P(x_n \mid x_{n-1}, \ldots, x_1) \cdot \ldots \cdot P(x_2 \mid x_1) P(x_1) \\ &= \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots, x_1) \end{aligned}$$

Wegen der Unabhängigkeitsannahmen ist dies äquivalent zu:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \mathit{parents}(x_i))$$

D.h. mit der Netztopologie und den CPTs können wir die Verbundwahrscheinlichkeit berechnen!

Beispiel



Es sind nur die Wahrscheinlichkeiten für die positiven Ereignisse angegeben. Die negativen ergeben sich als $P(\neg X) = 1 - P(X)$.

$$
\begin{aligned}
& P(J, M, A, \neg B, \neg E) = \\
&= P(J \mid A) P(M \mid A) P(A \mid \neg B, \neg E) P(\neg B) P(\neg E) \\
&= 0.9 \times 0.7 \times 0.001 \times 0.999 \times 0.998 \\
&= 0.00062
\end{aligned}
$$

Kompaktheit bayesscher Netze

- Zur expliziten Repräsentation der Verbundwahrscheinlichkeit brauchen wir eine Tabelle der Größe 2^n bei n Variablen.
- Falls in einem Netz jeder Knoten max. k Eltern hat, brauchen wir nur n Tabellen der Größe 2^k bei booleschen Variablen.
- *Beispiel:* $n = 20$ und $k = 5$

$\rightsquigarrow$ $2^{20} = 1.048.576$ und $20 \times 2^5 = 640$ verschiedene explizit repräsentierte Wahrscheinlichkeiten!

$\rightarrow$ Im schlechtesten Fall kann natürlich auch ein bayessches Netz exponentiell groß werden, z.B. wenn jede Variable von jeder anderen direkt beeinflusst wird.

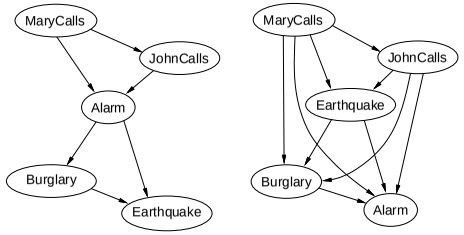
$\rightarrow$ abhängig von der Anwendungsdomäne (lokale vs. globale Interaktion) und dem Geschick des Designers.

Das Design eines Netzes

1. Ordne alle Variablen.
2. Nimm die erste von den übriggebliebenen.
3. Gib alle direkten Einflüsse von Knoten, die schon im Netz sind, auf den neuen Knoten an (Kanten + CPT).
4. Falls noch Variablen in der Liste, mache bei Schritt 2 weiter.

Beispiel

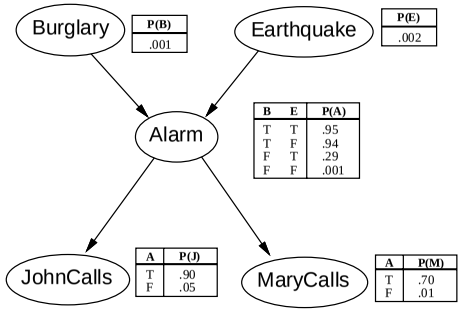
Links = *M,J,A,B,E*, rechts = *M,J,E,B,A*



$\rightsquigarrow$ Versuch, ein diagnostisches Modell von Symptomen zu Ursachen zu bauen, der immer zu Abhängigkeiten zwischen eigentlich unabhängigen Ursachen und separat auftretenden Symptomen führt.

Inferenz in bayesschen Netzen (1)

Instantiieren einiger Variablen (Evidenzen) und Abfragen von anderen Knoten.



Inferenz in bayesschen Netzen (2)

- $P(\textit{Burglary} \mid \textit{JohnCalls})$?

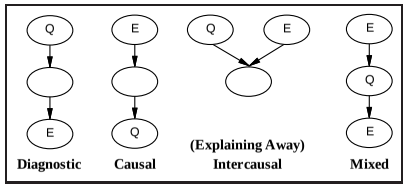
$\rightsquigarrow$ Der Alarm ist ziemlich zuverlässig und John ruft in 9 von 10 Fällen an, wenn ein Alarm vorliegt.

- Tatsächlich findet ein Einbruch nur alle 1000 Tage statt, aber John ruft 50 mal in 1000 Tagen an, d.h. auf einen Einbruch kommen 50 Fehlalarme.

$\rightsquigarrow$ $P(\textit{Burglary} \mid \textit{JohnCalls}) = 0.016$!

- $P(\textit{Burglary} \mid \textit{JohnCalls}, \textit{MaryCalls}) = 0.29$.

Typen von Inferenzen



1. **Diagnostisch**: Von Effekten zu Ursachen
   $P(\textit{Burglary} \mid \textit{JohnCalls}) = 0.016$
2. **Kausal**: Von Ursachen zu Effekten
   $P(\textit{JohnCalls} \mid \textit{Burglary}) = 0.86$
3. **Interkausal**: Zwischen Ursachen eines gemeinsamen Effekts
   $P(\textit{Burglary} \mid \textit{Alarm}) = 0.376$, aber
   $P(\textit{Burglary} \mid \textit{Alarm}, \textit{Earthquake}) = 0.003$.
4. **Gemischt:** Kombination von 1.-3.
   $P(\textit{Alarm} \mid \textit{JohnCalls}, \neg\textit{Earthquake}) = 0.03$

Unabhängigkeiten: D-Separierung (1)

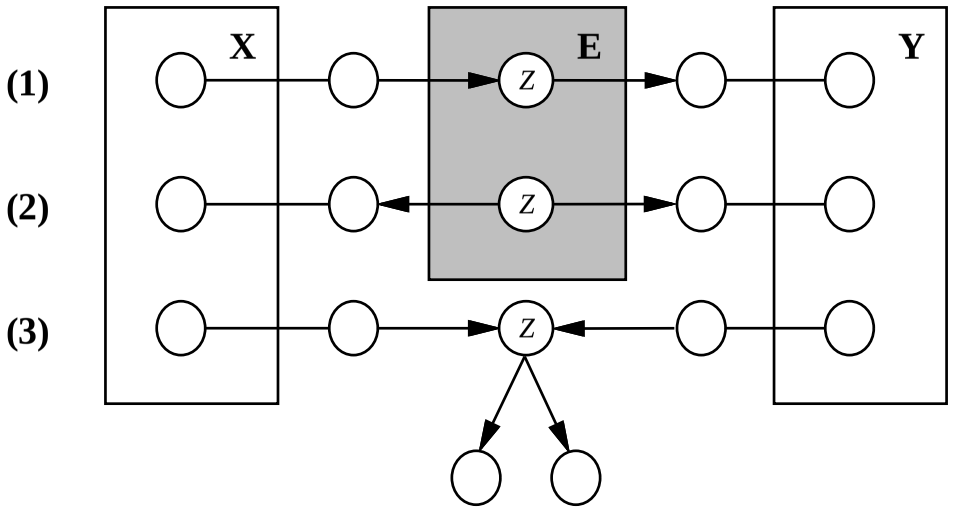
Eine Menge von Knoten E d-separiert die Mengen X und Y, falls jeder (ungerichtete) Pfad von X nach Y durch E blockiert ist.

$\rightsquigarrow$ X und Y sind bedingt unabhängig, gegeben E.

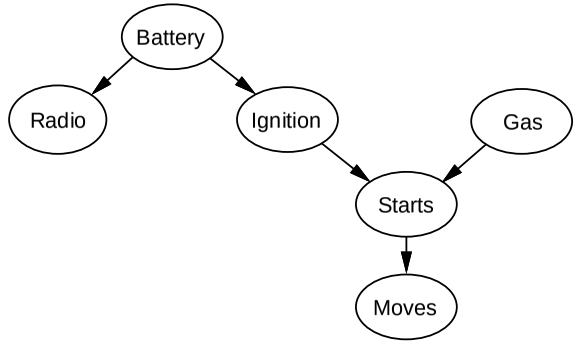
Ein Pfad von X nach Y ist durch E blockiert, falls es einen Knoten Z auf dem Pfad gibt, so dass

1. Z in E liegt und Z einen eingehenden und einen ausgehenden Teilpfad besitzt, oder
2. Z in E liegt und beide Teilpfade sind ausgehend, oder
3. Z nicht in E liegt, beide Teilpfade eingehend sind und kein Nachfolger von Z in E liegt.

Unabhängigkeiten: d-Separation (2)



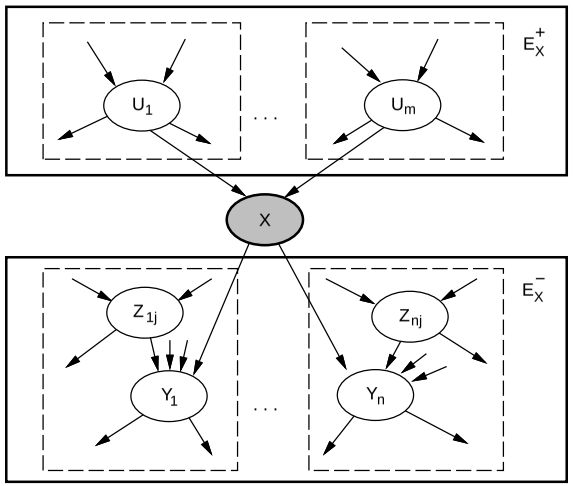
Beispiele für d-Separierung



1. $E =$ *Ignition* d-separiert *Gas* und *Radio*
2. $E =$ *Battery* d-separiert *Gas* und *Radio*
3. *Gas* und *Radio* sind (ohne Evidenz) unabhängig, aber nicht mehr, falls $E =$ *Starts* oder $E =$ *Moves*.

Inferenzmechanismen in bayesschen Netzen (1)

Grundannahme: Das Netz ist ein **Polytree**, d.h., falls man die Kantenrichtung ignoriert, bildet das Netz einen ungerichteten Baum.



Inferenzmechanismen in bayesschen Netzen (2)

- Wir können $P(X \mid E)$ berechnen, in dem wir das E aufsplitten in „untere“ und „obere“ Variablen (die bedingt unabhängig, gegeben X, sind!). Das machen wir rekursiv!

$\rightsquigarrow$ Polynomieller Algorithmus mit Bayesscher Regel

$\rightarrow$ Für Netze, die keine *Polytrees* sind, muss man Transformationen vornehmen, die exponentiell werden können. I.allg. ist die Inferenz in bayesschen Netzen NP-vollständig.

Der Algorithmus

function BELIEF-NET-ASK(X) **returns** a probability distribution over the values of X
  inputs: X, a random variable

  SUPPORT-EXCEPT(X, *null*)

---

function SUPPORT-EXCEPT(X, V) **returns** $\mathbf{P}(X|E_{X\setminus V})$

  if EVIDENCE?(X) **then return** observed point distribution for X
  else
    calculate $\mathbf{P}(E^-_{X\setminus V}|X)$ = EVIDENCE-EXCEPT(X, V)
    $\boldsymbol{U} \leftarrow$ PARENTS[X]
    if U is empty
      then return $\alpha\ \mathbf{P}(E^-_{X\setminus V}|X)\ \mathbf{P}(X)$
    else
      for each U_i **in** U
        calculate and store $\mathbf{P}(U_i|E_{U_i\setminus X})$ = SUPPORT-EXCEPT(U_i, X)
      return $\alpha\ \mathbf{P}(E^-_{X\setminus V}|X) \sum_{\mathbf{u}} \mathbf{P}(X|\mathbf{u}) \prod_i \mathbf{P}(U_i|E_{u_i\setminus X})$

---

function EVIDENCE-EXCEPT(X, V) **returns** $\mathbf{P}(E^-_{X\setminus V}|X)$

  $\mathbf{Y} \leftarrow$ CHILDREN[X] $-$ V
  if Y is empty
    then return a uniform distribution
  else
    for each Y_i **in Y do**
      calculate $\mathbf{P}(E^-_{Y_i}|y_i)$ = EVIDENCE-EXCEPT(Y_i, null)
      $\mathbf{Z}_i \leftarrow$ PARENTS[Y_i] $-$ X
      for each Z_{ij} **in** $\mathbf{Z}_i$
        calculate $\mathbf{P}(Z_{ij}|E_{Z_{ij}\setminus Y_i})$ = SUPPORT-EXCEPT(Z_{ij}, Y_i)
    return $\beta \prod_i \sum_{y_i} P(E^-_{Y_i}|y_i) \sum_{\mathbf{z}_i} \mathbf{P}(y_i|X, \mathbf{z}_i) \prod_j P(z_{ij}|E_{Z_{ij}\setminus Y_i})$

Systeme

- Das bekannteste medizinische Expertensystem, das bayessche Netze einsetzt, ist PATHFINDER IV.
- Deckt ca. 60 Lymphknotenkrankheiten und 100 Symptome und Testergebnisse ab.
- Es waren 14000 Schätzungen von Wahrscheinlichkeiten erforderlich, die in 40 Stunden Arbeit erstellt wurden

$\rightsquigarrow$ Besser als Weltklasse-Experten.

$\rightarrow$ Viele kommerzielle und PD-Tools für bayessche Netze und Erweiterungen erhältlich:

`http://bayes.stat.washington.edu/almond/belief.html`

Andere Ansätze (1)

- **Nicht-monotone Logik**
  - kann als qualitative Variante aufgefasst werden.
  - Tatsächlich sind einige NM-Logiken (die Ordnungen auf den Modellen betrachten) in einer Nicht-Standard Wahrscheinlichkeitstheorie rekonstruierbar (ϵ-Semantik mit verschwindend kleinen Wahrscheinlichkeiten).

Andere Ansätze (2)

- **Regelbasierte Systeme** mit „certainty factors“.
  - Logikbasierte Systeme mit Regelgewichten, die bei der Inferenz kombiniert werden.
  - Sind vom Berechenbarkeitsaufwand einfacher, können aber entweder nur kausale oder nur diagnostische Regeln verarbeiten, akzeptieren Evidenzen nur an den „Wurzeln“.
  - Liefern *inkorrekte* Ergebnisse, falls die Regelmenge „mehrfach verbunden“ ist.
  - Der Einsatz wird heute nicht mehr empfohlen.

Andere Ansätze (3)

- **Dempster-Shafer Theorie**
  - erlaubt neben der Repräsentation von Unsicherheit auch die Repräsentation von *Ignoranz*.
  - Beispiel: Bei einer fairen Münze würden wir von 0.5 für $P(\mathit{Kopf})$ ausgehen. Wenn wir aber nicht wissen, ob die Münze fair ist? $\leadsto$ $\mathit{Bel}(\mathit{Kopf}) = 0$, $\mathit{Bel}(\mathit{Zahl}) = 0$. Ist die Münze 90% fair, 0.5×0.9, d.h. $\mathit{Bel}(\mathit{Kopf}) = 0.45$

$\leadsto$ Intervall von Wahrscheinlichkeiten $[0, 1]$ ohne Wissen, $[0.45, 0.55]$ mit.

Andere Ansätze (4)

- **Fuzzy-Logik** und **Fuzzy-Mengen**
  - Dient zur Repräsentation und Verarbeitung von **Vagheit**, nicht Unsicherheit.
  - Beispiel: das Auto fährt *schnell*.
  - Einsatz insbesondere im Bereich Steuerung und Regelung.
  - Dort interpretierbar als *Interpolationstechnik*.

Zusammenfassung bayessche Netze

- Bayessche Netze erlauben eine kompakte Repräsentation der Verbundwahrscheinlichkeit.
- Dies wird erreicht durch Unabhängigkeitsannahmen.
- Sie unterstützen verschiedene Formen des Schließens gegeben Evidenzen: kausal, diagnostisch, interkausal, gemischt.
- Inferenz bedeutet dabei die Berechnung der Verteilung einer Menge von Variablen gegeben die Evidenzen.
- Die Komplexität der Inferenz in bayesschen Netzen hängt von der Struktur des Netzwerkes ab.
- I.allg. ist die Inferenz in bayesschen Netzen NP-vollständig.
- Für Polytrees ist die Komplexität polynomiell in der Größe des Netwerks.

Grundlagen der KI

12. Handeln unter Unsicherheit

Maximieren des erwarteten Nutzens

Wolfram Burgard

Inhalt

- Einführung in Nutzentheorie
- Auswahl einzelner Aktionen
- Sequentielle Entscheidungsprobleme
- Markov Entscheidungsprozesse
- Value iteration

Grundlagen der Nutzentheorie

Die Nutzenfunktion (*Utility function*) bewertet Zustände und formalisiert so das Bevorzugen bestimmter Zustände durch den Agenten.

$U(S)$ Nutzen des Zustandes S für den Agenten

Eine nichtdeterministische Aktion A kann zu den Folgezuständen $\mathit{Result}_i(A)$ führen. Wie hoch ist die Wahrscheinlichkeit, dass der Folgezustand $\mathit{Result}_i(A)$ erreicht wird, wenn A unter Evidenz E im aktuellen Zustand ausgeführt wird?

$$\leadsto P(\mathit{Result}_i(A) \mid \mathit{Do}(A), E)$$

erwarteter Nutzen:

$$\boxed{EU(A \mid E) = \sum_i P(\mathit{Result}_i(A) \mid \mathit{Do}(A), E) \times U(\mathit{Result}_i(A))}$$

Mit dem *Prinzip des maximalen erwarteten Nutzens MEU* sollte ein rationaler Agent diejenige Aktion auswählen, die $EU(A \mid E)$ maximiert.

Probleme mit dem *MEU*-Prinzip

$$P(\mathit{Result}_i(A) \mid \mathit{Do}(A), E)$$

erfordert ein vollständiges kausales Modell der Welt.

$\leadsto$ permanent an Veränderungen anpassen
$\leadsto$ NP-vollständig für bayessche Netze

$$U(\mathit{Result}_i(A))$$

erfordert Suche oder Planen um einen Zustand zu bewerten muss man die möglichen Folgezustände kennen („Wirkung des Zustands auf die Zukunft“).

Die Axiome der Nutzentheorie (1)

Rechtfertigung des *MEU*-Prinzips d.h. der Maximierung des Durchschnittsnutzens.

Szenario = *Lotterie* L

- mögliche Ergebnisse = mögliche Gewinne
- das Ergebnis wird vom Zufall bestimmt
- $L[1, A]$ = Zustand A

Beispiel:

Lotterie L mit 2 Ergebnissen A und B:

$$L = [p, A\ ;\ 1-p, B]$$

Präferenz von Lotterien:

$L_1 \succ L_2$	Agent L_1 bevorzugt gegenüber L_2
$L_1 \sim L_2$	Agent ist indifferent zu L_1 und L_2
$L_1 \succsim L_2$	L_1 bevorzugt oder indifferent zu L_2

Die Axiome der Nutzentheorie (2)

Gegeben Zustände A, B, C

- **Orderability**
  $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
  Ein Agent muss wissen, was er will: entweder eine von 2 Lotterien bevorzugen oder indifferent zu ihnen sein.
- **Transitivität**
  $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
  Verletzen der Transitivität verursacht irrationales Verhalten: $A \succ B \succ C \succ A$. Agent hat A und würde es mit Aufpreis für C eintauschen. C würde er wiederum für A und Geldzahlung eintauschen. $\rightsquigarrow$ Agent hat dadurch Geld verloren.

Die Axiome der Nutzentheorie (3)

- **Kontinuität**
  $A \succ B \succ C \Rightarrow \exists p\ [p, A\ ;\ 1-p, C] \sim B$
  Bei gegebenen Präferenzen lässt sich eine Lotterie konstruieren, mit A, C als Ergebnis, so dass der Agent indifferent zu dieser Lotterie und B „for sure“ ist.
- **Substituierbarkeit**
  $A \sim B \Rightarrow [p, A\ ;\ 1-p, C] \sim [p, B\ ;\ 1-p, C]$
  Einfachere Lotterien können gegen kompliziertere ersetzt werden, ohne dass sich Indifferenz ändert.

Die Axiome der Nutzentheorie (4)

- **Monotonie**
  $A \succ B \Rightarrow$
  $(p \geq q \Leftrightarrow [p, A\ ;\ 1-p, B] \succsim [q, A\ ;\ 1-q, B]$
  Bevorzugt ein Agent das Ergebnis A, dann muss er auch die Lotterie bevorzugen, die A mit höherer Wahrscheinlichkeit als Ergebnis liefert.
- **Dekomponierbarkeit**
  $[p, A\ ;\ 1-p, [q, B\ ;\ 1-q, C]] \sim$
  $[p, A\ ;\ (1-p)q, B\ ;\ (1-p)(1-q), C]$
  Lotterien mit mehr möglichen Gewinnen sollte ein Agent nicht automatisch bevorzugen („no fun in gambling“).

Nutzenfunktion und Axiome

Die Axiome sagen nur etwas über Präferenzen.

Die Existenz einer Nutzenfunktion folgt aus den Axiomen!

1. *Utility principle*

   Beachtet ein Agent in seinen Präferenzen die Axiome, dann existiert eine Funktion $U : S \to R$ mit

$$U(A) > U(B) \quad \Leftrightarrow A \succ B$$
$$U(A) = U(B) \quad \Leftrightarrow A \sim B$$

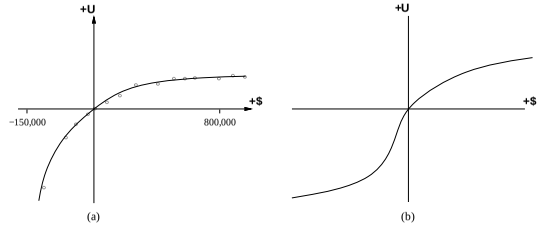
2. *Maximum expected utility principle*

$$U([p_1, S_1 ; \ldots ; p_n, S_n]) = \sum_i p_i \times U(S_i)$$

Wie entwirft man Nutzenfunktionen, die einen Agenten zum intendierten Verhalten verhelfen?

Mögliche Nutzenfunktionen

aus Marktmodellen:

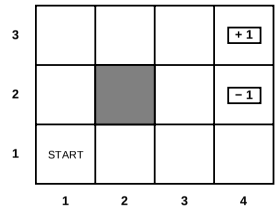


Skalierung und Normierung:

- bester Preis $U(u_\top) = 1$
- schlimmste Katastrophe $U(u_\bot) = 0$

Zwischenwerte erhält man durch Variieren der Wahrscheinlichkeit p in einer Standardlotterie und Abgleichen mit möglichem Ergebniszustand S, bis der Agent zu S und Lotterie indifferent ist.

Sequentielle Entscheidungsprobleme (1)



Der Agent soll den Zustand $(4, 3)$ erreichen (Belohnung $+1$) und den Zustand $(4, 2)$ vermeiden (Bestrafung -1). Aktionen sind: nord, süd, west, ost.

deterministische Variante: alle Aktionen führen immer zum nächsten Kästchen in der gewählten Richtung, beim Erreichen der Wand bleibt die Position des Agenten unverändert.

stochastische Variante: intendierter Effekt tritt mit 0.8 ein, mit 0.2 bewegt sich der Agent rechtwinklig zur gewünschten Richtung.

Beispiel:

$P((1, 2) \mid \mathit{Do}(\mathit{nord}, (1, 1)) = 0.8, P((2, 1) \mid \mathit{Do}(\mathit{nord}, (1, 1)) = P((1, 1) \mid \mathit{Do}(\mathit{nord}, (1, 1)) = 0.1$

Markov Entscheidungsproblem (MDP)

Gegeben:

- Menge von Zuständen in einer zugänglichen, stochastischen Umgebung
- Menge von Zielzuständen
- Menge von Aktionen
- Transitionsmodell M^a_{ij}
- Nutzenfunktion

Transitionsmodell: M^a_{ij} ist die Wahrscheinlichkeit, das Zustand j erreicht wird, wenn Aktion a in Zustand i ausgeführt wird.

Policy: Vollständige Abbildung von allen möglichen Zuständen auf die möglichen Aktionen.

Gesucht: Optimale Strategie (Policy), die den erwarteten Nutzen maximiert.

MDP-Basierter Agent

function SIMPLE-POLICY-AGENT(*percept*) **returns** an *action*
static: *M*, a transition model
U, a utility function on environment histories
P, a policy, initially unknown

if *P* is unknown **then** *P* ← the optimal policy given *U*, *M*
return *P*[*percept*]

Beispiel:

Nutzen einer Aktionsfolge
= Wert des Endzustandes $- \frac{\#Aktionsanzahl}{25}$

Bei 6 Aktionen zu Endzustand mit $+1$: $1 - \frac{6}{25} = 0.76$

Value iteration (1)

Ein Algorithmus zur Berechnung einer optimalen Strategie.

Grundidee: für jeden Zustand wird sein Nutzen berechnet. Ausgehend vom Nutzen kann eine optimale Aktion für jeden Zustand ausgewählt werden.

Eine Aktionsfolge generiert einen Baum möglicher Zustände (*Histories*). Eine Nutzenfunktion U_h auf Histories heißt *separierbar* gdw. es eine Funktion f derart gibt, dass

$$U_h([s_0, s_1, \ldots, s_n]) = f(s_0, U_h([s_1, \ldots, s_n]))$$

In der einfachsten Form wird eine additive Belohnungsfunktion R (*Reward*) verwendet:

$$U_h([s_0, s_1, \ldots, s_n]) = R(s_0) + U_h([s_1, \ldots, s_n]))$$

Im Beispiel $R((4,3)) = +1$, $R((4,2)) = -1$, $R(sonstige) = -\frac{1}{25}$.

Value iteration (2)

Der Nutzen eines Zustandes i ist durch den erwarteten Nutzen der optimalen Strategie unter Transitionsmodell M in i bestimmt:

$$\begin{aligned} U(i) &= EU(H(i, policy*) \mid M) \\ &= \sum P(H(i, policy*) \mid M) \\ &\quad \cdot U_h(H(i, policy*)) \\ policy^*(i) &= \operatorname*{argmax}_a \sum_j M^a_{ij}\, U(j) \\ U(i) &= R(i) + \operatorname*{argmax}_a \sum_j M^a_{ij}\, U(j) \end{aligned}$$

$\rightsquigarrow$ **Basis für Dynamisches Programmieren**

Value iteration (3)

Sind die Nutzen der Endzustände bekannt, dann lässt sich in bestimmten Fällen ein n-Schritt-Entscheidungsproblem auf das Berechnen der Nutzen der Endzustände des $(n-1)$-Schritt-Entscheidungsproblems reduzieren.

$\rightsquigarrow$ **Iteratives und effizientes Verfahren**

Problem: Typische Probleme enthalten Zyklen, so daß Histories eine potentiell unendliche Länge haben

Lösung: Anwendung von

$$U_{t+1}(i) \quad \leftarrow \quad R(i) + \max_a \sum_j M^a_{ij}\, U_t(j)$$

wobei $U_t(i)$ die Utility des i-ten Zustands nach t Iterationen ist.

Beobachtung: Mit $t \to \infty$ konvergieren die Utilities der einzelnen Zustände.

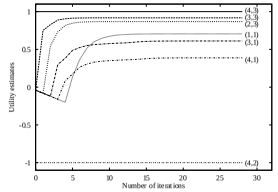
Value iteration (4)

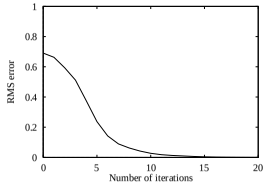
function VALUE-ITERATION(M, R) **returns** a utility function
  inputs: M, a transition model
    R, a reward function on states
  local variables: U, utility function, initially identical to R
    U', utility function, initially identical to R

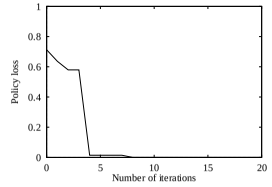
  repeat
    $U \leftarrow U'$
    for each state i **do**
      $U'[i] \leftarrow R[i] + \max_a \sum_j M_{ij}^a U[j]$
    end
  until CLOSE-ENOUGH(U, U')
  return U

Anwendung der Value Iteration

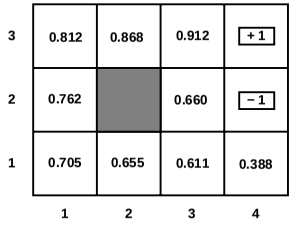
Entwicklung des Nutzens einzelner Zustände und der Fehler im Nutzen einzelner Zustände und der Policy :

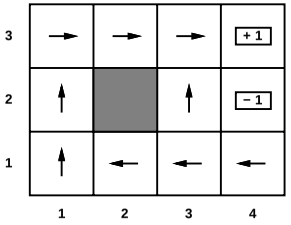






Nutzen der einzelnen Zustände nach Konvergenz und resultierende Policy:





Zusammenfassung

- Rationale Agenten können auf der Basis einer **Wahrscheinlichkeitstheorie** und einer **Nutzentheorie** entwickelt werden.
- Agenten, die Ihre Entscheidungen entsprechend den Axiomen der Nutzentheorie fällen, besitzen eine **Nutzenfunktion**.
- Sequentielle Probleme in unsicheren Umgebungen (MDP's) können durch Berechnen einer **Policy** gelöst werden.
- **Value iteration** ist ein Verfahren zur Berechnung optimaler Policies.

Grundlagen der KI

13. Maschinelles Lernen

Lernen durch Beobachtung

Wolfram Burgard

Inhalt

- Der lernende Agent
- Induktives Lernen
- Lernen von Entscheidungsbäumen
- Lernen von allgemeinen Hypothesen
- Warum Lernen funktioniert

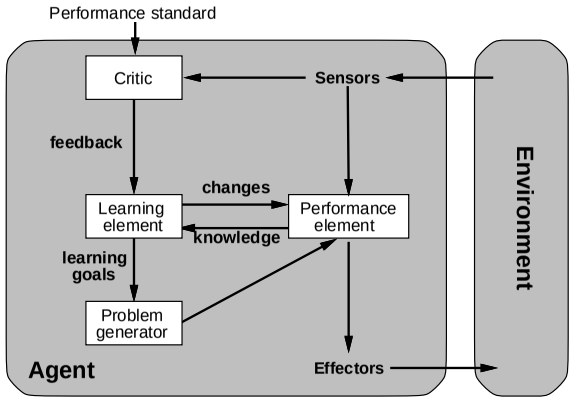
Lernen

- Was ist lernen?
  Ein Agent lernt, wenn er durch Erfahrung seine Fähigkeit, eine Aufgabe zu lösen, verbessern kann.
  $\rightarrow$ z.B. Spielprogramme
- Warum lernen?
  $\rightarrow$ Engineering, Philosophie, Kognitionswissenschaften
  $\rightarrow$ Data Mining (Entdeckung von neuem Wissen durch Datenanalyse)

Keine Intelligenz ohne Lernen!

Der lernende Agent

Bisher dienten die Wahrnehmungen des Agenten (Perzepte) nur dem Handeln. Jetzt sollen sie auch der Verbesserung zukünftiger Verhaltensweisen dienen.



Bausteine des lernenden Agenten

Performance-Element: Verarbeitet Wahrnehmungen und wählt Aktionen aus $\leadsto$ entspricht dem bisherigen Agentenmodell.

Learning-Element: Durchführen von Verbesserungen $\leadsto$ braucht Wissen über sich selbst und wie sich der Agent in der Umwelt bewährt.

Critic: Bewertung des Agentenverhaltens auf der Grundlage eines gegebenen externen Verhaltensmaßstabs $\leadsto$ Rückkopplung (feedback).

Problem-Generator: Vorschlagen von explorativen Aktionen, die den Agenten zu neuen Erfahrungen führen.

Das Learning-Element

Seine Funktionsweise wird von 4 entscheidenden Fragen beeinflusst:

1. Welche Teile des Performance-Elements sollen verbessert werden?
2. Welche Repräsentation wird gewählt?
3. Welche Form von Rückkopplung ist verfügbar?
4. Welche Ausgangsinformation steht für den Lernprozess zur Verfügung?

Arten der Rückkopplung beim Lernen

Eingabe: Information aus der Umwelt
Ausgabe: die Effekte der Aktionen des Agenten

Effekte, die der Agent durch sein Handeln erzielen will (Ideal) und Effekte, die dann tatsächlich in der Welt eintreten(Tatsache), unterscheiden sich oft erheblich.

Ziel des Lernens: Annähern der tatsächlichen an die ideale Funktion.

Supervised Learning: Eingabe/Ausgabe sind verfügbar. Ein Lehrer teilt dem System den Effekt auf die Umwelt und damit die korrekte Aktion mit.

Reinforcement Learning: Je nach Erfolg seiner Aktionen wird der Agent bestraft oder belohnt.

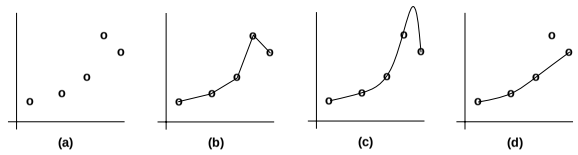
Unsupervised Learning: Der Agent kann nur Modelle für das Auftreten von Regelmäßigkeiten seiner Beobachtungen lernen, aber *nicht* was er richtigerweise tun müsste.

Induktives Lernen

Jede Art von Lernen kann als das **Lernen der Repräsentation einer Funktion** verstanden werden.

Ein **Beispiel** ist ein Paar $(x, f(x))$.

Induktive Inferenz: Für eine Menge von Beispielen für f ist eine Funktion h (**Hypothese**) gesucht, die f approximiert.



Bias: Tendenz, eine bestimmte Hypothese zu bevorzugen

Entscheidungsbäume

Eingabe: Beschreibung einer Situation durch eine Menge von Eigenschaften (entspricht Grundliteralen in FOL).

Ausgabe: Ja/Nein Entscheidung bezüglich eines Zielprädikats.

Entscheidungsbäume stellen boolesche Funktionen dar.

Ein interner Knoten im Entscheidungsbaum repräsentiert einen Test einer Eigenschaft. Zweige sind mit den möglichen Werten des Tests markiert. Jeder Blattknoten trägt den booleschen Wert, der bei Erreichen des Blattes zurückgegeben werden soll.

Ziel des Lernprozesses: Definition eines Zielprädikates in Form eines Entscheidungsbaums

Restaurantbeispiel (Tests)

Patrons: wieviele Gäste sind da? (none, some, full)

WaitEstimate: Wie lange warten? (0-10, 10-30, 30-60, >60)

Alternate: Gibt es eine Alternative? (T/F)

Hungry: Bin ich hungrig? (T/F)

Reservation: Habe ich reserviert? (T/F)

Bar: Hat das Restaurant eine Bar zum Warten? (T/F)

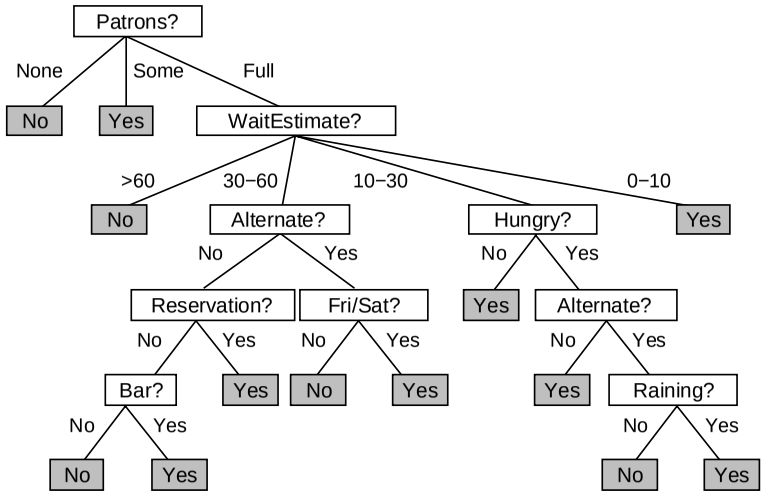
Fri/Sat: Ist es Freitag oder Samstag (T/F)

Raining: Regnet es draußen? (T/F)

Price: Wie teuer ist das Essen? ($, $$, $$$)

Type: Art des Restaurants? (french, italian, Thai, Burger)

Restaurantbeispiel (Entscheidungsbaum)



Repräsentation des Zielprädikats

Ein Entscheidungsbaum kann als Konjunktion von Implikationen repräsentiert werden. Eine Implikation entspricht den Pfaden, die in einem YES Knoten enden.

$\forall r$ [

Patrons(r,Full) $\wedge$ *WaitEstimate(r,10-30)* $\wedge$ *Hungry(r,No)* $\Rightarrow$ *WillWait(r)*

$\wedge$

Patrons*(r,*Some*)* $\Rightarrow$ WillWait*(r)*

$\wedge$ ...]

Ausdruckskraft von Entscheidungsbäumen

Theorem 1 *Alle aussagenlogischen Formeln (boolesche Fakten) sind mit Entscheidungsbäumen darstellbar.*

Kann ein Entscheidungsbaum eine beliebige Menge von Objekten darstellen?

$\leadsto$ Mit traditionellen Entscheidungsbäumen **nicht**. Alle Tests beziehen sich immer nur auf ein Objekt (hier: Restaurant r) und die Sprache von traditionellen Entscheidungsbäumen ist inhärent propositional.

Zum Beispiel ist

$\exists\, r_2$ *Near*$(r_2, r) \wedge$ *Price*$(r, p) \wedge$ *Price*$(r_2, p_2) \wedge$ *Cheaper*(p_2, p)

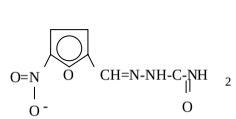
als Test nicht darstellbar.

Zwar könnten wir einen Test *CheaperRestaurantNearby* hinzufügen, aber ein Entscheidungsbaum mit *allen* solchen Attributen würde exponentiell wachsen.

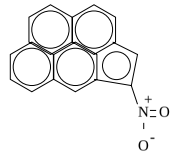
$\leadsto$ Erweiterungen existieren, z.B. (Blockeel und De Raedt, *Artificial Intelligence*, 1998)

Beispiel: Mutagenizität

Active



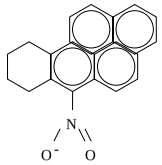
nitrofurazone



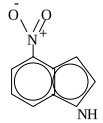
4-nitropenta[cd]pyrene

Structural alert:

Inactive



6-nitro-7,8,9,10-tetrahydrobenzo[a]pyrene



4-nitroindole

Kompakte Repräsentationen

Theoretisch kann jede Zeile einer Wahrheitswerttabelle in einen Pfad eines Entscheidungsbaums übertragen werden. Allerdings ist die Größe der Tabelle und damit des Baums exponentiell in der Anzahl der Attribute.

Funktionen, die einen Baum exponentieller Größe erfordern:

Parity Funktion:

$$p(x) = \begin{cases} 1 & \text{gerade Anzahl von Eingaben} \\ 0 & \text{sonst} \end{cases}$$

Majority Funktion:

$$m(x) = \begin{cases} 1 & \text{Hälfte der Eingaben ist 1} \\ 0 & \text{sonst} \end{cases}$$

Aber: Es gibt keine kompakte Repräsentation für alle möglichen booleschen Funktionen.

Die Trainingsmenge

Klassifizierung eines Beispiels = Wert des Zielprädikats

TRUE $\leadsto$ positives Beispiel

FALSE $\leadsto$ negatives Beispiel

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Ockham's Rasiermesser

Den trivialen Entscheidungsbaum erhält man, indem jedes Beispiel in einem Pfad repräsentiert wird.

⇝ repräsentiert nur die Erfahrung des Agenten

⇝ keine Extraktion eines allgemeineren Musters

⇝ besitzt keine Vorhersagekraft

> **Ockham's razor:** „Die wahrscheinlichste Hypothese ist die **einfachste**, die alle Beispiele umfasst."

⇝ Baum mit der *minimalen* Anzahl von Tests

Leider ist das Erzeugen des kleinsten Entscheidungsbaums nicht handhabbar.

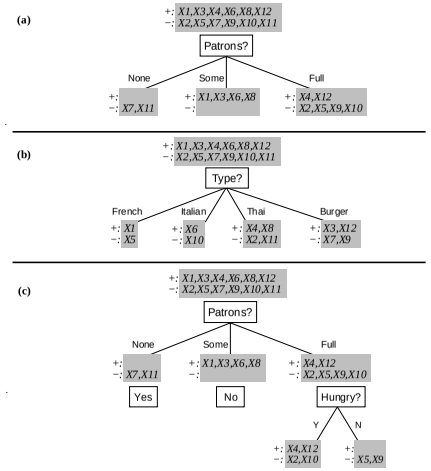
⇝ Heuristiken, die zu einer *kleinen* Menge von Tests führen

Gewichtung von Attributen

Wähle das Attribut aus, das die größtmögliche Unterscheidung der Beispiele erlaubt.

Beispiel: *Patrons?* ist günstig, weil im NONE/SOME-Fall keine weiteren Tests benötigt werden.

Type? hingegen ist ungünstig.



Rekursives Lernverfahren

Nach jedem Test liegt wieder ein Entscheidungsbaum-Lernproblem vor, wobei wir weniger Beispiele haben und die Anzahl der Attribute um eins reduziert ist.

Für den rekursiven Fall müssen 4 Fälle betrachtet werden:

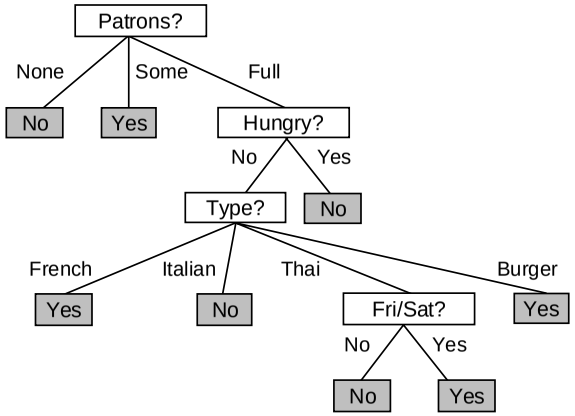
1. Positive und Negative Beispiele: Wähle neues Attribut.
2. Nur positive (oder nur negative) Beispiele: fertig.
3. Keine Beispiele: es gab kein Beispiel mit dieser Eigenschaft. Antworte JA, wenn Mehrzahl der Beispiele des Vaterknotens positiv ist, sonst NEIN.
4. Keine Attribute mehr, aber noch Beispiele mit unterschiedlicher Klassifikation: es lagen Fehler in den Daten vor (⇝ **NOISE**) oder die Attribute sind unzureichend. Reagiere wie im vorherigen Fall.

Der Algorithmus

function DECISION-TREE-LEARNING(*examples*, *attributes*, *default*) **returns** a decision tree
  inputs: *examples*, set of examples
    attributes, set of attributes
    default, default value for the goal predicate

  if *examples* is empty **then return** *default*
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** MAJORITY-VALUE(*examples*)
  else
    best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
    tree ← a new decision tree with root test *best*
    for each value v_i of *best* **do**
      $examples_i$ ← {elements of *examples* with *best* = v_i}
      subtree ← DECISION-TREE-LEARNING($examples_i$, *attributes* − *best*,
                MAJORITY-VALUE(*examples*))
      add a branch to *tree* with label v_i and subtree *subtree*
    end
    return *tree*

Anwendung auf die Restaurant-Daten



Bewertung eines Lernalgorithmus

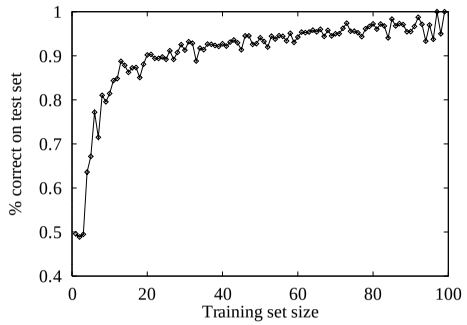
Methodologie zur Beurteilung der Vorhersagekraft:

- Sammle eine große Zahl von Beispielen.
- Unterteile die Beispiele in zwei *disjunkte* Mengen: **Trainings-** und **Testmenge**.
- Benutze Trainingsmenge, um H zu erstellen.
- Benutze Testmenge, um den Anteil korrekt klassifizierter Beispiele zu messen.
- Wiederhole das Verfahren für zufällig gewählte Trainingsmengen unterschiedlicher Größe.

Wichtige Strategien bei der Realisierung von Lernalgorithmen

- Trainings- und Testmenge müssen unbedingt getrennt gehalten werden.
- Beliebter Fehler: Nach Testläufen wird der Lernalgorithmus verändert und danach mit Trainings- und Testmengen aus derselben Grundmenge der Beispiele getestet. Dadurch wird Wissen über die Testmenge in den Algorithmus gesteckt und es besteht keine Unabhängigkeit zwischen Trainings- und Testmenge mehr.

Lernkurve des Retaurantbeispiels



Mit wachsender Größe der Trainingsmenge, nimmt die Vorhersagekraft zu.
Anwendungsbeispiele:

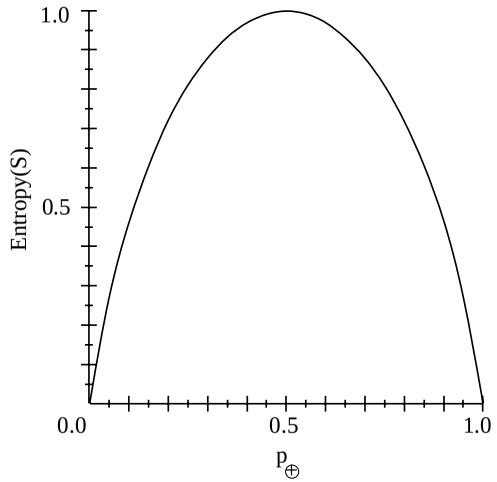
- Training von Steuerungssystemen (Flugsimulator)
- Entscheidungsunterstützungssysteme
- Data Mining in Datenbanken (C 4.5)

Bewertung von Attributen

Beispiel Münzwurf: Welchen Wert hat die Vorabinformation über den Ausgang eines Münzwurfs mit Einsatz 1$ und Gewinn 1$?

- Gefälschte Münze mit 99% Kopf und 1% Zahl.
  ($\Longrightarrow$ Durchschnittlicher Gewinn pro Wurf ist 0.98)
  $\leadsto$ Wert der Information über den Ausgang ist kleiner als 0.02$.
- Faire Münze.
  $\leadsto$ Wert der Information über den Ausgang ist kleiner als 1$.
  $\leadsto$ Je weniger man über den Ausgang weiß, desto wertvoller ist die Vorabinformation darüber.

Entropie



Informationstheorie

Messung des Werts einer Information in Bit.

Gegeben: Mögliche Werte v_i mit entsprechenden Wahrscheinlichkeiten $P(v_i)$

$$I(P(v_1),\ldots,P(v_n)) \quad = \quad \sum_{i=1}^{n} P(i)\ ld\Big(\frac{1}{P(i)}\Big)$$

Beispiel Münze:

$$I(\frac{1}{2},\frac{1}{2}) \quad = \quad \frac{1}{2}\times 1+\frac{1}{2}\times 1 = 1$$

$$I(0.99,0.01) \quad = \quad 0.08$$

Attributselektion (1)

Attribut A teile Beispielmenge E in p positive und n negative Beispiele:

$$I\left(\frac{p}{p+n},\frac{n}{p+n}\right) = \frac{p}{p+n}ld\left(\frac{p+n}{p}\right)+\frac{n}{p+n}ld\Big(\frac{p+n}{n}\Big)$$

Güte von A hängt zusätzlich von der anschließend noch benötigten Information ab.

Annahme: A teilt Trainingsmenge E in Teilmengen E_i, $i=1,\ldots,v$.

Jedes E_i hat wiederum $I\left(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}\right)$

Zufälliges Beispiel hat Wert i mit Wahrscheinlichkeit $\frac{p_i+n_i}{p+n}$

Attributselektion (2)

$\rightsquigarrow$ Durchschnittlicher Informationsgehalt nach Auswahl von A ist

$$R(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \times I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)$$

$\rightsquigarrow$ Informationsgewinn durch Auswahl von A ist

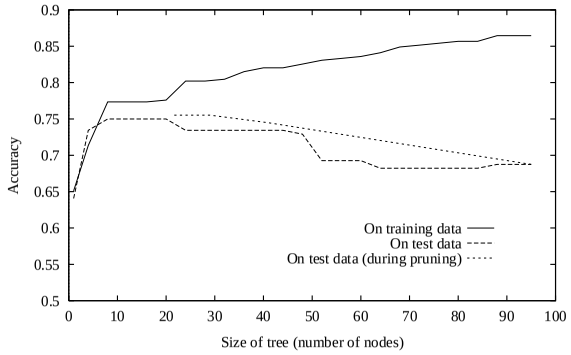
$$\text{Gain}(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - R(A)$$

Beispiel

$$\begin{aligned}
\text{GAIN(Patrons?)} &= 1 - \left[\frac{2}{12}I(0,1) + \frac{4}{12}I(0,1) + \frac{6}{12}I(\frac{2}{6},\frac{4}{6})\right] \\
&\approx 0.541 \\
\text{GAIN(Type?)} &= 1 - \left[\frac{2}{12}I(\frac{1}{2},\frac{1}{2}) + \frac{2}{12}I(\frac{1}{2},\frac{1}{2}) \right. \\
&\qquad \left. + \frac{4}{12}I(\frac{2}{4},\frac{2}{4}) + \frac{6}{12}I(\frac{2}{6},\frac{4}{6})\right] \\
&= 0
\end{aligned}$$

Noise

- Was ist Rauschen? Zufällige Fehler in der Lerndaten
- Effekt: Größere Bäume machen mehr Fehler auf neuen Daten (Overfitting)
- Vermeiden von Overfitting durch ein „Validation Set“: Trainingsmenge wird in zwei Teilmengen aufgeteilt; 70 % der Trainingsmenge wird zum Aufbau des Baumes verwendet, die übrigen 30% zum Bestimmen einer angemessenene Größe des Baumes („Pruning“)



Zusammenfassung: Entscheidungsbäume

- Eine Möglichkeit, boolesche Funktionen zu repräsentieren.
- Entscheidungsbäume können exponentiell in der Anzahl der Attribute sein.
- Es ist oft zu schwierig, den minimalen EB zu finden.
- Eine Methode zur Generierung von möglichst flachen EB beruht auf der Gewichtung der Attribute.

Lernen von allgemeinen Hypothesen

1-stelliges Zielprädikat $Q = \mathit{WillWait}(r)$

Der *Hypothesenraum* $\mathbf{H}$ ist die Menge aller logischen Definitionen für Q. Ist C_i eine mögliche Definition, so hat jede Hypothese H_i die Form

$$\forall x\, Q(x) \Leftrightarrow C_i(x)$$

Restaurant-Beispiel: Hypothese H_r

$$\begin{aligned}\forall r \quad & \mathit{WillWait}(r) \Leftrightarrow \\ & \mathit{Patrons}(r, \mathit{some}) \vee \\ & \mathit{Patrons}(r, full) \wedge \neg \mathit{Hungry}(r) \wedge \mathit{Type}(r, \mathit{French}) \vee \\ & \mathrm{Patrons}(r, \mathrm{full}) \wedge \neg \mathit{Hungry}(r) \wedge \mathit{Type}(r, \mathit{Thai}) \wedge \mathit{Fri/Sat}(r) \vee \\ & \mathit{Patrons}(r, full) \wedge \neg \mathit{Hungry}(r) \wedge \mathit{Type}(r, \mathit{Burger})\end{aligned}$$

Die Disjunktion der aktuellen Hypothesen $H_1 \vee H_2 \vee \cdots \vee H_n$ ist wahr.

Extension einer Hypothese: Menge der Beispiele, die C_i erfüllen.

Trainingsbeispiel und Hypothese

$D_i(X_i)$ ist Beschreibung des i-ten Beispiels X_i

$$\mathit{Alternate}(X_1) \wedge \neg \mathit{Bar}(X_1) \wedge \neg \mathit{Fri/Sat}(X_1) \wedge \mathit{Hungry}(X_1) \cdots \wedge \mathit{WillWait}(X_1)$$

Die Klassifizierung eines positiven Beispiels X_i ist $Q(X_i)$, die eines negativen ist $\neg Q(X_i)$.

Die vollständige Trainingsmenge ist die Konjunktion aller Sätze.

Eine Hypothese stimmt mit den Beispielen überein gdw. sie logisch konsistent mit der Trainingsmenge ist.

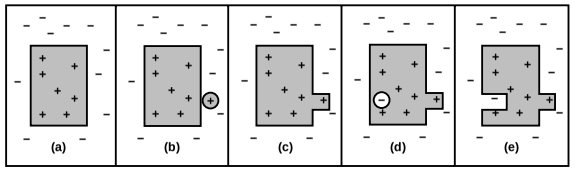
Hypothesen können mit einzelnen Beispielen inkonsistent sein:

1. *false negative*: H sagt, dass X negativ sein muss, aber tatsächlich ist X positiv.
2. *false positive*: H sagt, dass X positiv sein muss, aber tatsächlich ist X negativ.

Suchverfahren: Current-best Hypothesis

Betrachtet immer nur eine Hypothese, deren Extension bei auftretenden Inkonsistenzen verändert wird.

- **Generalisierung**: Erweiterung bei *false negative* (b+c)
- **Spezialisierung**: Verkleinern bei *false positive* (d+e)



Nach jeder Veränderung muss erneut auf Konsistenz getestet werden, damit auch die bisherigen Beispiele von der modifizierten Hypothese noch korrekt klassifiziert werden.

Der Algorithmus

function CURRENT-BEST-LEARNING(*examples*) **returns** a hypothesis

$H \leftarrow$ any hypothesis consistent with the first example in *examples*
for each remaining example in *examples* **do**
  if *e* is false positive for H **then**
    $H \leftarrow$ **choose** a specialization of H consistent with *examples*
  else if *e* is false negative for H **then**
    $H \leftarrow$ **choose** a generalization of H consistent with *examples*
  if no consistent specialization/generalization can be found **then fail**
end
return H

H_1 ist Generalisierung von H_2: $\forall x\, C_2(x) \Rightarrow C_1(x)$
$\rightsquigarrow$ *dropping conditions* (Konjunkte streichen)
$\rightsquigarrow$ schwächere Definition wird von mehr Beispielen erfüllt
Spezialisierung analog durch Hinzufügen von Konjunkten oder Streichen von Disjunkten (sprachabhängig).

Beispiel (1)

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

1. Beispiel X_1 ist positiv. $Alternate(X_1)$ ist wahr

$$H_1 : \forall x\, \mathit{WillWait}(x) \Leftrightarrow \mathit{Alternate}(x).$$

2. Beispiel X_2 ist false positive: H_1 spezialisieren

$$H_2 : \forall x\, \mathit{WillWait}(x) \Leftrightarrow \mathit{Alternate}(x) \wedge \mathit{Patrons}(x, \mathit{some})$$

Beispiel (2)

3. X_3 ist false negative: H_2 generalisieren

$$H_3 : \forall x\, \mathit{WillWait}(x) \Leftrightarrow \mathit{Patrons}(x, \mathit{some})$$

4. X_4 ist false negative: H_3 generalisieren
   Fallenlassen von $\mathit{Patrons}(x, \mathit{some})$ widerspricht X_2. Disjunktion hinzufügen, z.B.

$$H_4 : \forall x\, \mathit{WillWait}(x) \Leftrightarrow \mathit{Patrons}(x, \mathit{some}) \vee [\mathit{Patrons}(x, \mathit{full}) \wedge \mathit{Fri/Sat}(x)]$$

Nachteile von Current-best Hypothesis

- Alle vorherigen Beispiele müssen nochmals überprüft werden.
- Es wird nicht garantiert die einfachste Hypothese gefunden.
- Gute Heuristiken für die Suche sind schwierig zu finden.
- Suche kann leicht in eine Sackgasse führen, in der keine Modifizierung zu einer konsistenten Hypothese führt.

Backtracking ist notwendig, da zu jedem Zeitpunkt nur eine Hypothese aufrechterhalten wird. Der Hypothesenraum ist jedoch eine Disjunktion!

$\rightsquigarrow$ *Version Space*

Version-Space/Candidate-Elimination Learning

function VERSION-SPACE-LEARNING(*examples*) **returns** a version space
  local variables: V, the version space: the set of all hypotheses

  $V \leftarrow$ the set of all hypotheses
  for each example e in *examples* **do**
    if V is not empty **then** $V \leftarrow$ VERSION-SPACE-UPDATE(V, e)
  end
  return V

---

function VERSION-SPACE-UPDATE(V, e) **returns** an updated version space

  $V \leftarrow \{h \in V : h \text{ is consistent with } e\}$

Mit jedem inkonsistenten Beispiel wird der Hypothesenraum reduziert. Das Verfahren ist *inkrementell* und auch ein Beispiel für *least commitment*, da der Hypothesenraum nur an auftretende Inkonsistenzen angepasst wird.

$\rightsquigarrow$ kompakte Repräsentation des Hypothesenraums?

Boundary Set

Generalisierung/Spezialisierung definieren eine *partielle Ordnung* auf den Hypothesen.

Der Hypothesenraum lässt sich durch die Menge der

- allgemeinsten (*most general*) Hypothesen **G**-Set und
- speziellsten (*most specific*) Hypothesen **S**-Set repräsentieren.

Theorem 2 *Eine Hypothese H ist konsistent gdw. sie eine Generalisierung eines Elements von* **S***-Set und eine Spezialisierung eines Elements von* **G***-Set ist.*

G-Set = {TRUE}
S-Set = {FALSE}

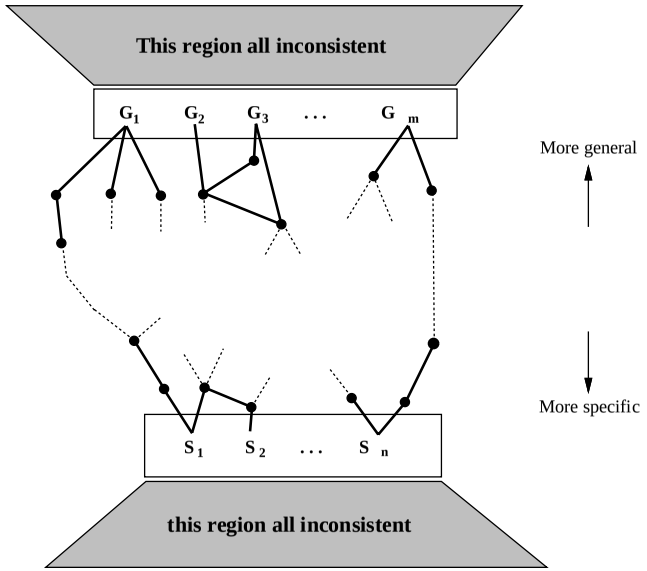
TRUE und FALSE sind boolesche Funktionen (Konstanten).

Update des Boundary Sets (1)

Seien $\mathbf{G} = \{G_1, \ldots, G_n\}$ und $\mathbf{S} = \{S_1, \ldots, S_n\}$.
Sei X ein neues Beispiel.

- X ist false positive für S_i.
  $\leadsto S_i$ zu allgemein: $\mathbf{S}' = \mathbf{S} \setminus \{S_i\}$
- X ist false negative für S_i.
  $\leadsto S_i$ zu spezifisch: generalisiere S_i.
- X ist false positive für G_i.
  $\leadsto G_i$ zu allgemein: spezialisiere G_i.
- X ist false negative für G_i.
  $\leadsto G_i$ zu spezifisch: $\mathbf{G}' = \mathbf{G} \setminus \{G_i\}$

Update des Boundary Sets (2)



Candidate Elimination Algorithm

$G =$ maximally general hypotheses in H

$S =$ maximally specific hypotheses in H

For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that h is consistent with d, and some member of G is more general than h
  - Remove from S any hypothesis that is more general than another hypothesis in S

Candidate Elimination Algorithm (ctd.)

- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that h is consistent with d, and some member of S is more specific than h
  - Remove from G any hypothesis that is less general than another hypothesis in G

Terminierung (1)

3 Möglichkeiten:

1. Genau eine Hypothese bleibt übrig.
   $\rightsquigarrow$ Lösung
2. Der Hypothesenraum kollabiert (**S** oder **G** leer).
   $\rightsquigarrow$ keine konsistente Hypothese möglich
3. Eine Menge von Hypothesen bleibt übrig.
   $\rightsquigarrow$ Disjunktion als Ergebnis (Klassifikation eines Testbeispiels ist Mehrheit der Klassifikation der einzelnen Disjunkte.)

Terminierung (2)

Probleme:

- Bei Rauschen (Noise) oder ungenügender Information kollabiert der Hypothesenraum immer.
- Bei Zulassen beliebiger Disjunktion enthält **S** eine speziellste Hypothese (die Disjunktion der Beschreibungen aller positiven Beispiele), und **G** eine allgemeinste Hypothese (die Negation der Disjunktion der Beschreibungen aller negativen Beispiele).

Warum Lernen funktioniert

Wie kann man entscheiden, dass h dicht an f liegt, wenn f doch aber unbekannt ist?

$\rightsquigarrow$ *Probably Approximately Correct*

Stationarity als Grundannahme des PAC-Learning: Trainings- und Testmenge werden mit der gleichen Wahrscheinlichkeitsverteilung aus der Population ausgewählt.

Wieviele Beispiele braucht man?

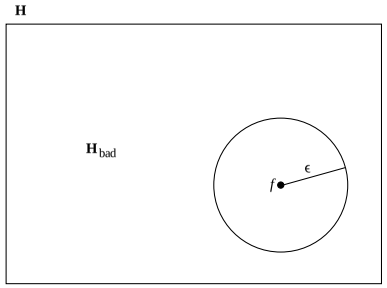
X	Beispielmenge
D	Population
H	Hypothesenraum ($f \in H$)
m	Anzahl der Beispiele in der Trainingsmenge

$$error(h) = P(h(x) \neq f(x)) \leq \epsilon$$

PAC Lernen

Eine Hypothese h heißt *approximativ korrekt* gdw. $error(h) \leq \epsilon$.

Zu zeigen: nach dem Training mit m Beispielen ist jede konsistente Hypothese mit hoher Wahrscheinlichkeit approximativ korrekt.



Wie hoch ist die Wahrscheinlichkeit, daß eine falsche Hypothese $h_b \in H_{bad}$ konsistent mit den ersten m Beispielen ist?

Sample Complexity

Annahme: $error(h_b) > \epsilon$. $\Rightarrow$

$$\begin{aligned} P(h_b \text{ kons. mit 1 Beispiel}) &\leq (1-\epsilon) \\ P(h_b \text{ kons. mit } m \text{ Beispielen}) &\leq (1-\epsilon)^m \\ P(H_{bad} \text{ enthält kons. } h) &\leq |H_{bad}|(1-\epsilon)^m \end{aligned}$$

Es gilt auf jeden Fall $|H_{bad}| \leq |H|$ und damit

$$\leq |H|(1-\epsilon)^m$$

Forderung:

$$|H|(1-\epsilon)^m < \delta$$

wegen $\ln(1-\epsilon) \leq -\epsilon$ gilt das, falls

$$m \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$$

Sample complexity: Anzahl benötigter Beispiele als Funktion über ϵ und δ.

Sample Complexity (2)

Sample complexity realisiert ein Maß, um den Trainingsaufwand abzuschätzen.

Ist eine Hypothese mit $m \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$ Beispielen konsistent, dann ist Fehler einer Wahrscheinlichkeit von wenigstens $1-\delta$ höchstens ϵ.

Problem: Bestimmung der Größe des Hypothesenraums

Beispiel: boolesche Funktionen

Anzahl boolescher Funktionen über n Attributen ist $H = 2^{2^n}$.

Die Sample complexity wächst daher mit 2^n.

Weil 2^n gerade die Anzahl möglicher Beispiele ist, gibt es keinen Lernalgorithmus, der besser ist als eine Lookup-Tabelle und gleichzeitig konsistent mit allen bekannten Beispielen ist.

Die Abschätzung

$$\begin{aligned} |H|(1-\epsilon)^m &\leq \delta \\ ln\,|H|(1-\epsilon)^m &\leq ln\,\delta \\ ln\,|H| + m\,ln(1-\epsilon) &\leq ln\,\delta \end{aligned}$$

Es gilt die Abschätzung $ln\,(1+\alpha) \leq \alpha$ und mit $\alpha = -\epsilon$ erhalten wir $ln\,(1-\epsilon) \leq -\epsilon$.

$$\begin{aligned} ln\,|H| - m\epsilon &\leq ln\,\delta \\ \frac{1}{\epsilon}\,ln\,|H| - m &\leq \frac{1}{\epsilon}\,ln\,\delta \\ \frac{1}{\epsilon}\,ln\,|H| &\leq m + \frac{1}{\epsilon}\,ln\,\delta \\ \frac{1}{\epsilon}\left(ln\,|H| - ln\,\delta\right) &\leq m \end{aligned}$$

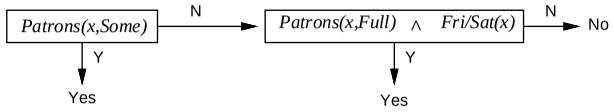
$\rightsquigarrow$ Für die booleschen Funktionen wächst m exponentiell, da H doppelt exponentiell wächst.

Dilemma: Schränkt man den Hypothesenraum nicht ein, dann kann kein Algorithmus richtig lernen. Schränkt man ihn ein, dann wird f eventuell eliminiert.

Lernen von Entscheidungslisten

Im Vergleich zu Entscheidungsbäumen:

- die Gesamtstruktur ist einfacher
- der einzelne Test komplexer



Dies entspricht der Hypothese

$$H_4 : \forall x \quad \mathit{WillWait}(x) \Leftrightarrow \mathit{Patrons}(x, \mathit{some}) \vee [\mathit{Patrons}(x, \mathit{full}) \ \wedge \ \mathit{Fri/Sat}(x)]$$

Sind Tests von beliebiger Länge zugelassen, so können beliebige boolesche Funktionen dargestellt werden.

k-DL: Sprache mit Tests der Länge $\leq k$.

$$\boxed{\textbf{k-DT} \subseteq \textbf{k-DL}}$$

function DECISION-LIST-LEARNING(*examples*) **returns** a decision list, *No* or failure

if *examples* is empty **then return** the value *No*
t ← a test that matches a nonempty subset $examples_t$ of *examples*
such that the members of $examples_t$ are all positive or all negative
if there is no such *t* **then return** failure
if the examples in $examples_t$ are positive **then** *o* ← *Yes*
else *o* ← *No*
return a decision list with initial test *t* and outcome *o*
and remaining elements given by DECISION-LIST-LEARNING(*examples* − $examples_t$)

$$\begin{aligned}
|\text{k-DL(n)}| &\leq 3^{|Conj(n,k)|} \times |Conj(n,k)|! && \text{(Yes,No,kein-Test, alle Permutationen)} \\
|Conj(n,k)| &= \sum_{i=0}^{k} \binom{2n}{i} && \text{(Kombination ohne Wdh. von pos/neg Attribut)} \\
&= O(n^k) \\
|\text{k-DL(n)}| &= 2^{O(n^k \, ld(n^k))} && \text{(mit Eulerscher Summenformel)} \\
m &\geq \frac{1}{\epsilon}\left(ln\,\frac{1}{\delta} + O(n^k \, ld(n^k))\right)
\end{aligned}$$

Zusammenfassung

Induktives Lernen als das Lernen der Repräsentation einer Funktion anhand von Beispielen ihres Eingabe/Ausgabe-Verhaltens.

- *Entscheidungsbäume/discrimination nets* lernen deterministische boolesche Funktionen.
- *Current-best Hypothesis* als Methode zum Lernen logischer Theorien, die zu jedem Zeitpunkt genau eine Hypothese verfolgt.
- *Version Space/Candidate Elimination* als Methode zum Lernen logischer Theorien, deren Grundlage eine kompakte Repräsentation des Hypothesenraums ist.
- *PAC Lernen* beschäftigt sich mit der Komplexität des Lernens.
- *Entscheidungslisten* als „einfach“ zu lernende Funktionen.

Grundlagen der KI

15. Lernen in Neuronalen Netzen

Grundlagen, Netzwerkstrukturen, Perzeptron, Backpropagation

Wolfram Burgard

Inhalt

Motivation (1)

vom mathematischen Standpunkt:

Neuronale Netze als *Methode*, Funktionen zu repräsentieren.

- durch Netzwerke von einfachen Berechnungselementen (vergleichbar mit logischen Schaltkreisen)
- die aus Beispielen gelernt werden können

$\rightsquigarrow$ in dieser Vorlesung

vom biologischen Standpunkt:

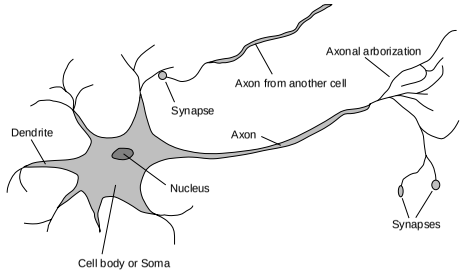
Neuronale Netze als (adäquates ?) *Modell* des Gehirns und seiner Funktionsweise.

$\rightsquigarrow$ Konnektionismus
$\rightsquigarrow$ nicht in dieser Vorlesung

Motivation (2)

Bisher: KI von „oben“: Modellierung eines intelligenten Agenten durch algorithmische Realisierung von bestimmten Aspekten rationalen Handelns.

Jetzt: KI von „unten“: Nachbildung der Struktur und der Verarbeitungsmechanismen des Gehirns (grob):



Viele Prozessoren (Neuronen) und Verbindungen (Synapsen), die parallel und lokal Informationen verarbeiten.

Konventionelle Rechner vs. Gehirn

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/sec	10^5	10^{14}

Beachte: Hirnschaltzeit langsam $(10^{-3}s)$, aber Updates erfolgen parallel. Dagegen braucht die serielle Simulation auf einem Rechner mehrere hundert Zyklen für 1 Update.

Vorteile natürlicher neuronaler Netze:

- Hohe Verarbeitungsgeschwindigkeit durch **massive Parallelität**.
- Funktionstüchtig selbst bei Ausfall von Teilen des Netzes (**Fehlertoleranz**).
- Langsamer Funktionsabfall bei fortschreitenden Ausfällen von Neuronen. (**Graceful degradation**).
- Gut geeignet für **induktives Lernen**.

Forschung auf dem Gebiet neuronaler Netze

1943: McCulloch und Pitts führen die Idee eines künstlichen Neurons ein.

1943-1969: Forschung an neuronalen Netzen meist Einschichten-Netze = *Perzeptrons*

1969: Minsky und Papert zeigen, dass *Perzeptrons* sehr beschränkt sind

1969-1980: Kaum noch Arbeiten auf dem Gebiet

Seit 1981: Renaissance neuronaler Netze

Heute:

- In der KI als *Werkzeug* benutzt zum **Approximieren von Funktionen**
- Insbesondere für **senso-motorische Aufgaben** gut geeignet
- In der Biologie und Physiologie als Modell

Grundbegriffe künstlicher neuronaler Netze

Einheiten (*Units*): stellen die Knoten (Neuronen) im Netz dar.

Verbindungen (*Links*): Kanten zwischen den Knoten des Netzes. Jeder Knoten hat Ein- und Ausgabekanten.

Gewichte (*Weights*): Jede Kante hat eine Gewichtung, in der Regel eine reelle Zahl.

Ein-/Ausgabeknoten (*Input and Output units*): Besonders gekennzeichnete Knoten, die mit der Außenwelt verbunden sind.

Aktivierungsniveau (*Activation level*): Der von einem Knoten aus seinen Eingabekanten zu jedem Zeitpunkt berechnete Wert. Dieser Wert wird über Ausgabekanten an die Nachbarknoten weitergeleitet.

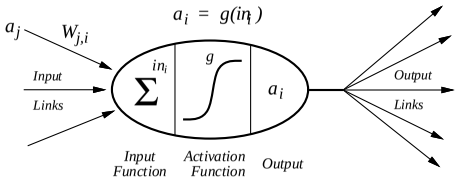
Funktionsweise einer Einheit

Eingabefunktion:
in_i berechnet die Stärke der Eingabe für Einheit i als *Linearkombination* von Eingabeaktivierung a_j und Gewicht $W_{j,i}$ über alle Knoten a_j, die mit i verbunden sind:

$$in_i = \sum_j W_{j,i} \times a_j = \mathbf{W}_i \cdot \mathbf{a}_i$$

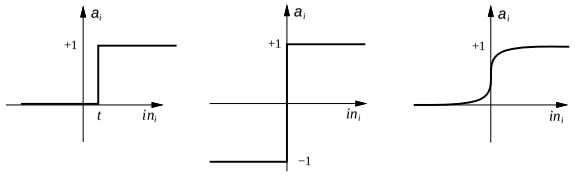
Aktivierungsfunktion:
berechnet mit Hilfe einer i.allg. *nicht-linearen Funktion* g das Aktivierungsniveau a_i:

$$a_i := g(in_i) = g\left(\sum_j W_{j,i} \times a_j\right)$$



Aktivierungsfunktionen

$$step_t(x) = \begin{cases} 1, \text{if } x \geq t \\ 0, \text{if } x < t \end{cases} \qquad sign(x) = \begin{cases} +1, \text{if } x \geq t \\ -1, \text{if } x < t \end{cases} \qquad sigmoid(x) = \frac{1}{1+e^{-x}}$$



(a) Step function (b) Sign function (c) Sigmoid function

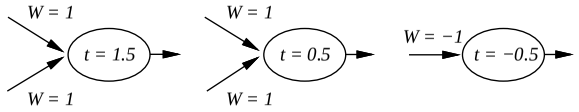
Beachte: t in $step_t$ stellt einen Schwellwert dar. Analogie zur Reizleitung im Nervensystem.

Mathematisch ist t äquivalent zu zusätzlicher Eingabe mit Aktivierungsniveau $a_0 = -1$ und $W_{0,i} = t$. Damit

$$a_i = step_t\left(\sum_{j=1}^{n} W_{j,i} \times a_j\right) = step_0\left(\sum_{j=0}^{n} W_{j,i} \times a_j\right)$$

Was kann man mit neuronalen Netzen darstellen?

- Funktionen ...
- insbesondere *boolesche Funktionen*
  hier am Beispiel mit der $step_t$ Funktion:



- ... und durch Verschaltung vieler Einheiten beliebige boolesche Funktionen
  $\rightsquigarrow$ Schaltkreistheorie
- allerdings können die Netze dann sehr groß werden. Ein Ergebnis aus der Schaltkreistheorie besagt, dass die *meisten* booleschen Funktionen mit $O(2^n/n)$ Gattern dargestellt werden müssen.

$\rightsquigarrow$ Aber neuronale Netze können noch viel mehr ...

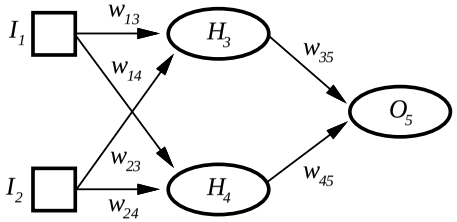
Netzwerktopologien

Rekurrent: Verbindungen können beliebige Topologien aufweisen.

Feed-forward: Verbindungen stellen gerichteten azyklischen Graphen dar (DAG).

Feed-forward-Netze

FF-Netzwerke werden normalerweise in **Schichten** konstruiert. Dabei keine Verbindungen zwischen Einheiten einer Schicht, immer nur zur jeweils nächsten Schicht. Beispiel:



Feed-forward sind Netze am besten verstanden. Die Ausgabe ist allein eine Funktion der Eingaben und der Gewichte.

Rekurrente Netze sind oft instabil, oszillieren oder zeigen chaotisches Verhalten. Interner Zustand bildet sich durch Rückpropagierung im Aktivierungsniveau ab - Kurzzeitgedächtnis (Gehirn ist massiv rekurrent.)

Rekurrente Netzwerktypen: *Hopfield-Netze*

- Rekurrente Netze mit symmetrischen bidirektionalen Verbindungen ($W_{i,j} = W_{j,i}$).
- Alle Einheiten fungieren als Ein- und Ausgabeeinheiten.
- Aktivierungsfunktion ist die *sign* Funktion, d.h. Aktivierungsniveau = ± 1.

$\rightsquigarrow$ Realisieren **assoziative Speicher**. Nach dem Training mit Beispielen und einer neuen Eingabe wird das "ähnlichste" Beispiel gefunden.

Beispiel: Trainieren mit n Fotos. Neue Eingabe = Teil eines schon gesehenen Fotos. Netz rekonstruiert das Gesamtbild.

Beachte: Jedes Gewicht im Netz ist partielle Darstellung aller Bilder.

Satz Ein Hopfield Netz kann $0.138 \times N$ Beispiele sicher speichern, wobei N die Zahl der Einheiten im Netz ist.

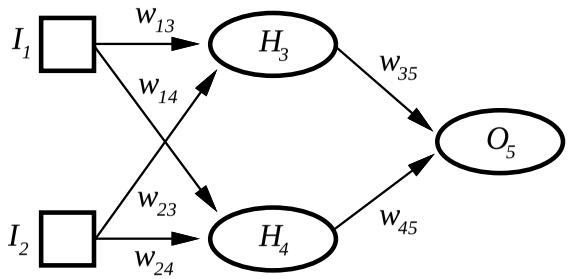
Rekurrente Netzwerktypen: *Boltzmann-Maschinen*

- Ebenfalls symmetrische Gewichte.
- Es gibt Einheiten, die weder Ein- noch Ausgabeeinheiten sind.
- Stochastische Aktivierungsfunktion.

$\rightsquigarrow$ Zustandsübergänge ähneln Suchverfahren mit *Simulated annealing*: Suche nach bester Approximation der Trainingsmenge.

- Formal identisch zu einer Variante von **bayesschen Netzen**, die mit stochastischer Simulation bearbeitet werden.

Geschichtete Feed-Forward (GFF) Netze (1)



Eingabeeinheiten: (I_1, I_2) erhalten ihr Aktivierungsniveau von der Umwelt.

Ausgabeeinheiten: (O_5) teilen ihr Ergebnis der Umwelt mit.

Geschichtete Feed-Forward (GFF) Netze (2)

Verborgene Einheiten (*hidden units*): (H_3, H_4) Einheiten ohne direkte Verbindungen nach draußen, angeordnet in **verborgenen Schichten** (*Hidden layers*).

Anzahl der Schichten: wird bestimmt unter Ignorierung der Eingabeschicht.

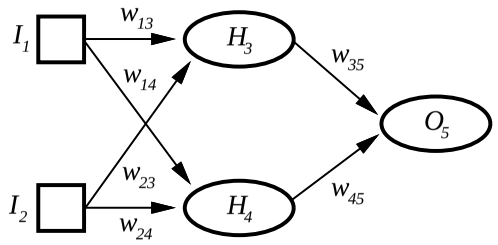
Perzeptron: Einschichtiges FF-Netz (ohne verborgene Schicht).

Darstellungskraft von GFF-Netzen

- GFF-Netze mit einer verborgenen Schicht können beliebige *stetige Funktionen* darstellen.
- GFF-Netze mit zwei verborgenen Schicht können daneben auch *diskontinuierliche Funktionen* darstellen.

Bei fester Topologie und Aktivierungsfunktion g sind darstellbare Funktionen charakterisierbar durch parametrisierte Form (Parameter = Gewichte).

Beispiel



$a_5 = g(W_{35}a_3 + W_{45}a_4)$
$= g(W_{35}g(W_{13}a_1 + W_{23}a_2) + W_{45}g(W_{14}a_1 + W_{24}a_2))$

$\Rightarrow$ **Lernen**
= Suche nach richtigen Parameterwerten
= **nichtlineare Regression**

Lernen mit neuronalen Netzen

Gegeben eine Menge von **Beispielen** $E = \{e_1, \ldots, e_n\}$, wobei jedes Beispiel aus *Eingabewerten* und *Ausgabewerten* besteht, soll das Netzwerk die **Funktion** lernen, die die Beispiele erzeugt hat – dabei ist die Netzwerktopologie vorgegeben und die Gewichte sollen angepasst werden.

Lernen in **Epochen**: Normalerweise legt man dem Netz die Beispiele mehrfach (in Epochen) vor.

```
function NEURAL-NETWORK-LEARNING(examples) returns network

  network ← a network with randomly assigned weights
  repeat
      for each e in examples do
          O ← NEURAL-NETWORK-OUTPUT(network, e)
          T ← the observed output values from e
          update the weights in network based on e, O, and T
      end
  until all examples correctly predicted or stopping criterion is reached
  return network
```


Optimale Netzwerkstruktur?

Die Wahl des richtigen Netzes ist ein schwieriges Problem . . .

Außerdem kann das optimale Netz exponentiell groß werden (relativ zu Ein-/Ausgabe).

- Netz zu klein: gewünschte Funktion nicht darstellbar.
- Netz zu groß: Beim **Trainieren** eines Netzes lernt dieses die Beispiele „auswendig“, ohne zu generalisieren $\Rightarrow$ *Overfitting*.

Es gibt keine gute Theorie zur Wahl des richtigen Netzes.

Heuristik des *Optimal brain damage*

Wähle Netz mit maximaler Zahl an Verbindungen. Nach 1. Training reduziere Zahl der Verbindungen mit Hilfe von Informationstheorie. Iteriere diesen Prozess.

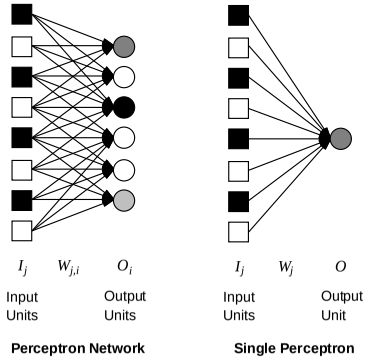
Bsp.: Netz zum Erkennen von Postleitzahlen. 3/4 der anfänglichen Verbindungen wurden eingespart.

Es gibt auch Verfahren um von einem kleinen Netz zu einem besseren größeren zu kommen.

Perzeptron

Einschichten-FF-Netze (Perzeptrons) wurden insbesondere in den 50ern studiert, da sie die einzigen GFF-Netze waren, für die Lernverfahren bekannt waren.

Da alle Ausgabeknoten unabhängig voneinander sind, konzentrieren wir uns auf einen einzigen Ausgabeknoten O:

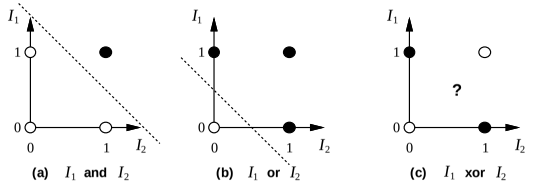


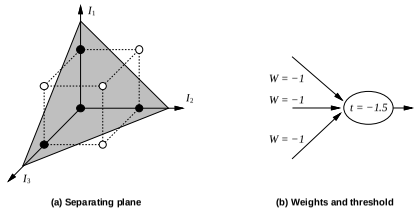
$$O = Step_0\left(\sum_j W_j I_j\right) = Step_0(\mathbf{W} \cdot \mathbf{I})$$

mit $I_0 = -1$ für Schwellwert

Was können Perzeptrons darstellen?

Perzeptrons können genau die Klasse der **linear separierbaren Funktionen** darstellen. Beispiele:





Grund: $\mathbf{W} \cdot \mathbf{I} = 0$ teilt den gesamten Raum gerade in zwei Teile entlang einer Graden (2D), einer Ebene (3D), ...

Was können Perzeptrons lernen?

$\rightsquigarrow$ alle linear separierbaren Funktionen

Die meisten Lernverfahren folgen dem Ansatz der *Current Best Hypothesis*.

Hypothese = Netz
= aktuelle Werte der Gewichte

Das Netzwerk wird mit zufällig gewählten Gewichten, in der Regel aus $[-0.5, 0.5]$, initialisiert.

Durch Verändern der Gewichte wird die Konsistenz mit den Trainingsbeispielen hergestellt:

- Reduzieren der Differenz zwischen vorhergesagtem und tatsächlichem Wert

$\rightsquigarrow$ Perzeptron Lernregel

Die Perzeptron-Lernregel

Sei
- T die korrekte Ausgabe und
- O die Ausgabe des Netzes, dann ist
- $Err = T - O$ der Fehler.

Err positiv $\leadsto O$ erhöhen
Err negativ $\leadsto O$ erniedrigen

Da jede Eingabeeinheit $W_j I_j$ zu O beiträgt, sollte bei positivem I_j das Gewicht W_j größer werden, und bei negativem I_j das Gewicht W_j kleiner werden.

$$\boxed{W_j := W_j + \alpha \times I_j \times Err}$$

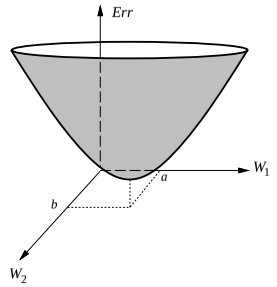
wobei α eine positive Konstante ist, die **Lernrate** genannt wird.

Satz [Rosenblatt 1960]. Lernen mit der Perzeptron-Lernregel konvergiert immer zu korrekten Werten bei linear separierbaren Funktionen.

Wieso funktioniert die Perzeptron-Lernregel?

Perzeptron-Lernen kann als *Gradientenabstieg* im Raum aller Gewichte aufgefasst werden, wobei der Gradient durch die *Fehlerfläche* bestimmt wird. Jede Gewichtskonfiguration bestimmt einen Punkt auf der Fläche.

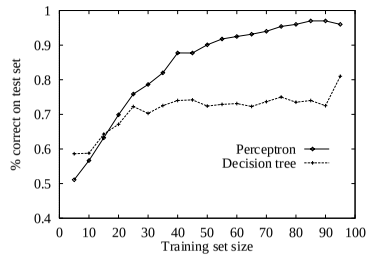
$\leadsto$ *partielle Ableitung* bzgl. der einzelnen Gewichte.



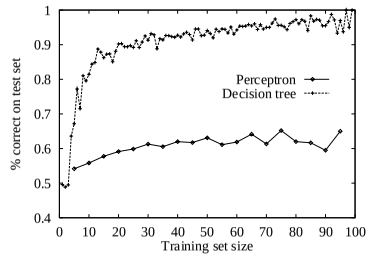
Da man leicht zeigen kann, dass es bei linear separierbaren Funktionen keine lokalen Minima gibt, folgt der Perzeptron-Konvergenzsatz.

Lernkurven: Perzeptron vs. Entscheidungsbaum

Majoritätsproblem

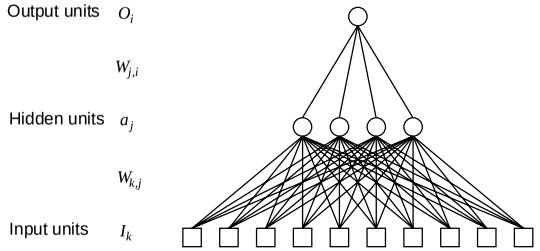


Restaurantproblem (nicht linear separierbar):



Mehrschichten-FF-Netze

Beispiel:



. . . geeignet zum Lernen des Restaurantproblems

$\leadsto$ Lernalgorithmen für FF Netze sind nicht effizient
$\leadsto$ sie konvergieren nicht zum globalen Minimum

Aber aus dem PAC Lernen wissen wir, dass das Lernen allgemeiner Funktionen aus Beispielen im ungünstigsten Fall nicht handhabbar ist - unabhängig von der Methode!

Lernen in Mehrschichten-Netzen: *Backpropagation*

Es wird versucht, für jedes Gewicht $W_{j,i}$ und $W_{k,j}$ seinen Anteil am Fehler zu bestimmen und die Gewichte entsprechend zu ändern $\leadsto$ der Fehler wird rückwärts durchs Netz propagiert.

$$W_{j,i} \quad := \quad W_{j,i} + \alpha \times a_j \times \underbrace{Err_i \times g'(in_i)}_{\Delta_i},$$

wobei g' die Ableitung von g ist. Da g differenzierbar sein muss, wählt man für g meist die *sigmoid*-Funktion.

Um wieviel müssen wir nun die $W_{k,j}$ Gewichte ändern?
$\leadsto$ Abhängig von Δ_i und $W_{j,i}$:

$$\Delta_j \quad = \quad g'(in_j) \times \sum_i (W_{j,i} \times \Delta_i)$$
$$W_{k,j} \quad := \quad W_{k,j} + \alpha \times I_k \times \Delta_j$$

Der Backpropagation-Algorithmus

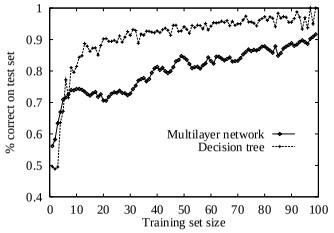
function BACK-PROP-UPDATE(*network*, *examples*, α) **returns** a network with modified weights
inputs: *network*, a multilayer network
examples, a set of input/output pairs
α, the learning rate

repeat
for each *e* **in** *examples* **do**
/* Compute the output for this example */
O $\leftarrow$ RUN-NETWORK(*network*, $\mathbf{I}^e$)
/* Compute the error and Δ for units in the output layer */
$\mathbf{Err}^e \leftarrow \mathbf{T}^e - \mathbf{O}$
/* Update the weights leading to the output layer */
$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i^e \times g'(in_i)$
for each subsequent layer **in** *network* **do**
/* Compute the error at each node */
$\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$
/* Update the weights leading into the layer */
$W_{k,j} \leftarrow W_{k,j} + \alpha \times I_k \times \Delta_j$
end
end
until *network* has converged
return *network*

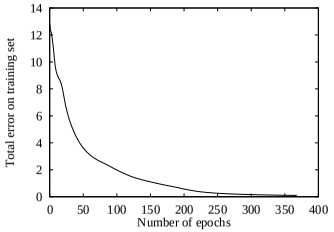
Err_i^e = Fehler $T_i^e - O_i^e$ für i-te Ausgabeeinheit
$\mathbf{Err}^e$ = Fehlervektor
Δ_i = $Err_i \times g'(in_i)$

Lernkurven

. . . für das Restaurantbeispiel mit dem Zweischichtennetz:



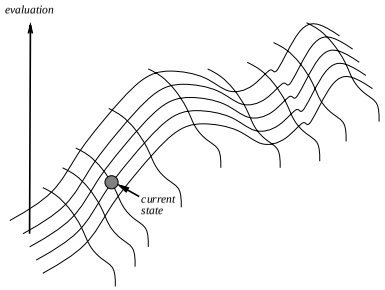
Fehler in Abhängigkeit von der Anzahl der Lernepochen:



Backpropagation = Gradientenabstieg

Backpropagation kann wieder als *Gradientenabstieg* im Raum aller Gewichte aufgefasst werden, wobei der Gradient durch die *Fehlerfläche* bestimmt wird. Allerdings treten nun nicht mehr rein konvexe Fehlerflächen auf, sondern der Lernalgorithmus ist mit dem Problem *lokaler Minima* konfrontiert.

$\leadsto$ Neustarten
$\leadsto$ Rauschen
$\leadsto$ Tabusuche



Herleitung ...

Sei $\mathbf{W}$ der Gewichtsvektor und E die folgende Fehlerfunktion:

$$\begin{aligned} E(\mathbf{W}) &= \frac{1}{2}\sum_i (T_i - O_i)^2 = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i}\, a_j))^2 \\ &= \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i}\, g(\sum_k W_{k,j} I_k)))^2 \end{aligned}$$

Partielle Ableitung von E nach $W_{j,i}$: $\dfrac{\partial E}{\partial W_{j,i}}$.

a_j ist unabhängig von $W_{j,i}$. Die meisten Summanden enthalten $W_{j,i}$ nicht.

Abzuleiten: $\frac{1}{2}(T_i - g(\sum_j W_{j,i}\ a_j))^2$:

$$\begin{aligned} \frac{\partial E}{\partial W_{j,i}} &= \frac{2}{2}(T_i - g(\sum_j W_{j,i}\, a_j)) \times -g'(\sum_j W_{j,i}\, a_j) \times a_j = -(T_i - O_i) \times g'(in_i) \times a_j \\ &= -a_j \Delta_i \end{aligned}$$

Wir gehen in Richtung entgegen der Steigung!

Für $W_{k,j}$ kann man zeigen: $\frac{\partial E}{\partial W_{k,j}} = -I_k \Delta_j$

Neuronale Netze ...

- **Wofür sind sie gut?** Neuronale Netze eignen sich zum Lernen von Funktionen über Attributen, die auch *kontinuierlich* sein können.
- **Größe des Netzes**: Bis zu $2^n/n$ Einheiten, um beliebige boolesche Funktionen darzustellen, meist aber weniger.
- **Effizienz des Lernens**: keine brauchbaren theoretischen Ergebnisse. Außerdem Problem der lokalen Minima.
- **Generalisierung**: Gut, wenn man das richtige Netz gewählt hat ...
- **Rauschen**: stellt kein Problem dar.
- **Transparenz**: schlecht! Oft weiß man nicht, warum das Netz gut funktioniert.
- **Verwendung zusätzlichen Wissens**: schlecht. Es gibt keine Theorie, wie man Vorwissen einem neuronalen Netz mitgeben kann.

Beispielanwendung (1): Aussprache lernen

- Das NETtalk-Programm [Sejnowski & Rosenberg 87] lernt aus einem transkribierten englischsprachigen Text Ausspracheregeln.
- 29 Eingabeeinheiten (26 Buchstaben + Zeichensetzung), 80 verborgene Einheiten, Ausgabeschicht steuert Merkmale des zu produzierenden Lautes.
- Nach 50 Epochen auf einem Text mit 1024 Worten erreicht NETtalk 95% Korrektheit auf der Trainingsmenge.
- Auf Testmenge allerdings nur 78%!
- Das kann auch mit anderen Methoden einfach erreicht werden (z.B. Hidden Markov Modelle).
- Das System hat allerdings die Suggestion hervorgerufen, dass es den kindlichen Spracherwerb nachahmt (die Tonlage!) ...

Beispielanwendung (2): Handgeschriebene Postleitzahlen erkennen

- Vorverarbeitung zur Segmentierung der einzelnen Ziffern.
- 16×16 Eingabeeinheiten, 3 verborgene Schichten mit 768, 192, bzw. 30 Einheiten, und 10 Ausgabeeinheiten.
- Netzwerktopologie organisiert als Merkmalsdetektoren, deshalb nur 9760 Verbindungen (statt 200000).
- Training auf 7300 Beispielen. Korrektheit auf den Testdaten, nachdem 12% als mehrdeutig zurückgewiesen wurden: 99%!
- Realisierung als VLSI-Schaltkreis.

Beispielanwendung (3): Autofahren

- ALVINN [Pomerleau 93] steuert ein Fahrzeug auf einer Straße.
- Eingabe von Farbstereokamera, das zu einem 30×32 Bild für die Eingabeeinheiten transformiert wird.
- 5 verborgene Einheiten und 30 Ausgabeeinheiten für Steuerungskommandos.
- Trainingsdaten werden durch Beobachtung eines menschlichen Fahrers gewonnen.
- **Achtung**: Menschliche Fahrer sind zu gut!
- Nach 5 Minuten Fahrt als Trainingsdaten kann ALVINN auf den trainierten Straßen bis zu 70 mph schnell fahren.
- **Probleme**: Andere Straßentypen und andere Beleuchtungsverhältnisse.
- MANIAC [Jochem et al 93] ist ein Metasystem, das das „richtige“ ALVINN Netz für die aktuelle Straße auswählt.

Zusammenfassung & Ausblick

- Neuronale Netze sind ein Berechnungsmodell, das einige Aspekte des Gehirns nachbildet.
- **Feed-Forward** Netze sind die am einfachsten zu analysierenden Netze.
- Ein **Perzeptron** ist ein einschichtiges FF-Netz, mit dem man aber nur *linear separierbare* Funktionen repräsentieren kann. Mit Hilfe der **Perzeptron-Lernregel** können solche Funktionen erlernt werden.
- **Mehrschichtige Netze** können beliebige Funktionen darstellen.
- **Backpropagation** ist ein Verfahren, um Funktionen in solchen Netzen zu lernen, und das auf Gradientenabstieg beruht. In der Praxis kann man damit viele Probleme lösen, allerdings gibt es keine Garantien.

$\leadsto$ *Problem:* Verbindung von symbolischen und „sub-symbolischen“ Verfahren