
Einführung in die KI

Niclas Kusenbach

LaTeX version:  SCHOUTER

Table of Contents

Contents

1 Introduction to AI Principle	3	3.2 Suchalgorithmen: Tree Search vs. Graph Search	10
1.1 Was ist Künstliche Intelligenz?	3	3.2.1 Bewertungskriterien für Suchstrategien	10
1.2 Intelligenztests und Philosophie	3	3.3 Uninformierte Suche (Blind Search)	11
1.2.1 Der Turing-Test (Imitation Game)	3	3.3.1 Breadth-First Search (BFS) - Breitensuche	11
1.2.2 Das Chinesische Zimmer (The Chinese Room)	3	3.3.2 Uniform-Cost Search (UCS)	11
1.3 Kategorisierung von KI	4	3.3.3 Depth-First Search (DFS) - Tiefensuche	11
1.3.1 Starke vs. Schwache KI	4	3.3.4 Depth-Limited Search (DLS)	11
1.3.2 Eigenschaften eines KI-Systems	4	3.3.5 Iterative Deepening Search (IDS)	11
1.4 Dimensionen der KI-Definition	4	3.4 Informierte Suche (Heuristische Suche)	12
1.4.1 Rationalität vs. Gesetze des Denkens	4	3.4.1 Greedy Best-First Search	12
1.5 Grundlagen und verwandte Disziplinen	5	3.4.2 A* Search (A-Star)	12
1.5.1 Taxonomie der KI	5	3.5 Heuristiken für A*	12
1.6 Grenzen und Probleme moderner KI	5	3.5.1 Admissibility (Zulässigkeit)	12
1.7 Ergänzung: Agenten (aus der Übung)	5	3.5.2 Consistency (Konsistenz / Monotonie)	12
2 AI101-02: AI Systems	7	3.5.3 Dominanz von Heuristiken	13
2.1 KI-Systeme und Agenten	7	3.5.4 Beispiel: 8-Puzzle Heuristiken	13
2.2 Rationalität	7	4 AI101-04: Local Search and Adversarial Search	14
2.3 Task Environments (PEAS)	7	4.1 Lokale Suche (Local Search)	14
2.4 Eigenschaften von Umgebungen	8	4.1.1 Hill Climbing (Bergsteigen)	14
2.5 Agententypen	9	4.1.2 Simulated Annealing	15
2.5.1 1. Simple Reflex Agents (Einfache Reflex-Agenten)	9	4.1.3 Local Beam Search	15
2.5.2 2. Model-based Reflex Agents (Modellbasierte Reflex-Agenten)	9	4.2 Suche in kontinuierlichen Räumen	15
2.5.3 3. Goal-based Agents (Zielbasierte Agenten)	9	4.2.1 Gradient Descent (Gradientenabstieg)	15
2.5.4 4. Utility-based Agents (Nutzenbasierte Agenten)	9	4.3 Adversarial Search (Spiele)	16
2.5.5 5. Learning Agents (Lernende Agenten)	9	4.3.1 Minimax-Algorithmus	16
3 AI101-03 Uninformed and Informed Search	10	4.3.2 Alpha-Beta Pruning	16
3.1 Einführung und Problemdefinition	10	4.3.3 Umgang mit Ressourcenbeschränkungen	17
3.2 Suchalgorithmen: Tree Search vs. Graph Search	10	5 Constraint Satisfaction Problems (CSPs)	18
3.2.1 Bewertungskriterien für Suchstrategien	10		
3.3 Uninformierte Suche (Blind Search)	11		
3.3.1 Breadth-First Search (BFS) - Breitensuche	11		
3.3.2 Uniform-Cost Search (UCS)	11		
3.3.3 Depth-First Search (DFS) - Tiefensuche	11		
3.3.4 Depth-Limited Search (DLS)	11		
3.3.5 Iterative Deepening Search (IDS)	11		
3.4 Informierte Suche (Heuristische Suche)	12		
3.4.1 Greedy Best-First Search	12		
3.4.2 A* Search (A-Star)	12		
3.5 Heuristiken für A*	12		
3.5.1 Admissibility (Zulässigkeit)	12		
3.5.2 Consistency (Konsistenz / Monotonie)	12		
3.5.3 Dominanz von Heuristiken	13		
3.5.4 Beispiel: 8-Puzzle Heuristiken	13		

5.1	Definition und Komponenten	18	8.2.1	Wahrscheinlichkeitsraum und Axiome	28
5.2	Arten von Constraints	18	8.2.2	Zufallsvariablen	28
5.3	Lösungsansätze: Suche und Backtracking	18	8.3	Verteilungen und Rechenregeln	29
5.3.1	Backtracking Search	18	8.3.1	Joint Distribution (Verbund- wahrscheinlichkeit)	29
5.4	Heuristiken für Backtracking	19	8.3.2	Marginalisierung und Bedingte Wahrscheinlichkeit	29
5.4.1	1. Welche Variable soll als nächste belegt werden?	19	8.3.3	Satz von Bayes	29
5.4.2	2. Welchen Wert soll die Variable annehmen?	19	8.4	Bayesian Networks	29
5.5	Constraint Propagation (Inferenz)	19	8.4.1	Unabhängigkeit	30
5.5.1	Konsistenzarten	19	8.4.2	Definition und Semantik	30
5.5.2	Forward Checking vs. Arc Consis- tency	19	8.5	Inferenz in Bayesian Networks	30
5.6	Lokale Suche für CSPs	20	8.5.1	Exakte Inferenz: Variable Elimina- tion	30
5.7	Struktur von CSPs	20	8.5.2	Komplexität	30
5.7.1	Problem-Dekomposition	20	8.6	Approximative Inferenz: Sampling	31
5.7.2	Baumstrukturierte CSPs	20	8.6.1	Direct Sampling (ohne Evidenz) . .	31
5.7.3	Fast-Baum-Strukturen (Cutset Conditioning)	20	8.6.2	Rejection Sampling	31
8.6.3	Markov Chain Monte Carlo (MCMC)	31			
6	AI101-06: Logik und KI 1 - Aussagenlogik	21			
6.1	Einführung: Wissensbasierte Agenten . .	21			
6.2	Die Wumpus-Welt	21			
6.3	Logik: Syntax und Semantik	21			
6.4	Aussagenlogik (Propositional Logic) . .	21			
6.4.1	Syntax der Aussagenlogik	21			
6.4.2	Semantik: Wahrheitstabellen	22			
6.5	Inferenz (Schlussfolgern)	22			
6.5.1	Model Checking	22			
6.5.2	Logische Eigenschaften	22			
6.5.3	Logische Äquivalenzen	22			
6.6	Resolution	22			
6.6.1	Umwandlung in CNF	23			
6.6.2	Resolutions-Algorithmus	23			
6.7	Horn-Klauseln und Chaining	23			
6.7.1	Algorithmen für Definite Klauseln	23			
6.8	Grenzen der Aussagenlogik	23			
7	Logic and AI 2: First-Order Logic (FOL)	24			
7.1	Einführung und Motivation	24			
7.2	Syntax und Elemente der FOL	24			
7.2.1	Quantoren	24			
7.3	Modellierung: Von natürlicher Sprache zu FOL	24			
7.4	Inferenz in FOL	25			
7.4.1	Substitution (SUBST)	25			
7.4.2	Skolemierung	25			
7.4.3	Unifikation	25			
7.5	Resolution in FOL	25			
7.6	Prolog (Programming in Logic)	26			
7.7	Grenzen der Prädikatenlogik	26			
7.8	Neuro-Symbolic AI (Ausblick)	26			
8	Umgang mit Unsicherheit und Probabilis- tisches Schließen	28			
8.1	Einführung in Unsicherheit	28			
8.2	Grundlagen der Wahrscheinlichkeitstheorie	28			

1 Introduction to AI Principle

1.1 Was ist Künstliche Intelligenz?

Die Definition von Künstlicher Intelligenz (KI) ist nicht eindeutig und unterliegt einem stetigen Wandel („Moving Goalposts“). Aufgaben, die früher als KI galten, werden heute oft als reine Softwaretechnik angesehen, sobald sie gelöst sind.

Definitionen von KI

Es gibt verschiedene Ansätze, KI zu definieren:

- **John McCarthy (1971):** „Die Wissenschaft und Ingenieurskunst, **intelligente Maschinen** zu bauen, insbesondere intelligente Computerprogramme.“ Er betonte, dass KI nicht auf biologisch beobachtbare Methoden beschränkt sein muss.
- **Marvin Minsky (1969):** „Die Wissenschaft, Maschinen dazu zu bringen, Dinge zu tun, die **Intelligenz** erfordern würden, wenn sie von Menschen getan würden.“

Gründe für die schwierige Definition sind unter anderem:

- Fehlende offizielle Definition.
- Einfluss von Science-Fiction auf die öffentliche Wahrnehmung.
- Das Phänomen, dass ehemals schwere Aufgaben (z.B. Schach) nach ihrer Lösung als „einfache Rechnerei“ abgetan werden, während Aufgaben, die für Menschen einfach sind (z.B. Wahrnehmung, Motorik), für Maschinen sehr schwer sein können (Moravec's Paradox).

1.2 Intelligenztests und Philosophie

Um festzustellen, ob ein System intelligent ist, wurden verschiedene Tests und Gedankenexperimente entwickelt.

1.2.1 Der Turing-Test (Imitation Game)

Vorgeschlagen von Alan Turing. Die Grundannahme ist, dass ein Wesen intelligent ist, wenn es sich in seinem Verhalten nicht von einem anderen intelligenten Wesen unterscheiden lässt.

Ablauf:

1. Ein menschlicher Fragesteller interagiert blind (per Text) mit zwei Spielern: A und B.
2. Einer der Spieler ist ein Mensch, der andere ein Computer.
3. Wenn der Fragesteller nicht zuverlässig entscheiden kann, welcher Spieler der Computer ist, hat der Computer den Test bestanden.

Kritik: Der Test prüft *Verhalten*, nicht das *Verständnis* oder *Bewusstsein*. Ein System kann menschliches Verhalten simulieren, ohne die zugrunde liegenden Konzepte zu verstehen.

1.2.2 Das Chinesische Zimmer (The Chinese Room)

Ein Gegenargument zum Turing-Test von John Searle, das den Unterschied zwischen Syntax (Zeichenmanipulation) und Semantik (Bedeutung) hervorhebt.

Szenario:

- Eine Person, die kein Chinesisch versteht, sitzt in einem geschlossenen Raum.
- Sie hat ein Regelbuch (Programm), das beschreibt, wie auf chinesische Zeichenfolgen (Input) mit anderen chinesischen Zeichenfolgen (Output) reagiert werden soll.

- Für einen Außenstehenden wirkt es so, als verstünde die Person Chinesisch.

Schlussfolgerung: Die Person manipuliert nur Symbole anhand ihrer Form (Syntax), versteht aber deren Inhalt (Semantik) nicht. Ebenso könnte eine KI intelligent *wirken* (Turing-Test bestehen), ohne wirklich intelligent zu *sein*.

1.3 Kategorisierung von KI

1.3.1 Starke vs. Schwache KI

Unterscheidung der Reichweite

- **Generelle KI (General AI / Strong AI):** Ein System, das **jede** intellektuelle Aufgabe bewältigen kann, die auch ein Mensch lösen kann. Es besitzt Verständnis und Bewusstsein (Forschungsziel).
- **Schwache KI (Narrow AI / Weak AI):** Ein System, das darauf spezialisiert ist, eine **konkrete** oder eine begrenzte Menge von Aufgaben zu lösen (aktueller Stand der Technik, z.B. Schachcomputer, Bilderkennung).

1.3.2 Eigenschaften eines KI-Systems

Ein modernes KI-System sollte idealerweise folgende Eigenschaften aufweisen:

- **Adaptability (Anpassungsfähigkeit):** Die Fähigkeit, die Leistung durch Lernen aus Erfahrung zu verbessern.
- **Autonomy (Autonomie):** Die Fähigkeit, Aufgaben in Umgebungen ohne ständige Anleitung durch einen Benutzer oder Experten auszuführen.
- **Rationality (Rationalität):** Das Treffen der „richtigen“ Entscheidungen (siehe unten).

1.4 Dimensionen der KI-Definition

KI kann entlang zweier Dimensionen klassifiziert werden:

1. **Prozess:** Fokus auf Denkprozesse/Schlussfolgern vs. Fokus auf Verhalten/Handeln.
2. **Maßstab:** Erfolg gemessen am menschlichen Vorbild vs. Erfolg gemessen an einem idealen Rationalitätsbegriff.

Dies ergibt vier Felder der KI-Forschung:

	Menschlicher Maßstab	Ideal (Rationalität)
Denken	Systems that think like humans (Kognitionswissenschaft: Modellierung der menschlichen Denkweise)	Systems that think rationally (Logik: „Gesetze des Denkens“, korrekte Schlussfolgerungen)
Handeln	Systems that act like humans (Turing-Test Ansatz: Simulation menschlichen Verhaltens)	Systems that act rationally (Rationaler Agent: Maximierung des erwarteten Nutzens)

Table 1: Die vier Ansätze der KI

1.4.1 Rationalität vs. Gesetze des Denkens

- **Laws of Thought (Logik):** Beschäftigt sich mit unwiderlegbaren, logischen Schlussfolgerungen. Problem: In der Realität gibt es oft Unsicherheiten, für die Logik allein nicht ausreicht.
- **Rational Behavior (Rationales Handeln):** Das „Richtige“ tun. Das Richtige ist definiert als das, was die Erreichung der Ziele angesichts der verfügbaren Informationen maximiert (Maximierung des *Expected Utility*).
- **Vorteile der Rationalität:** Sie ist allgemeiner als reine Logik (funktioniert auch bei Unsicherheit) und wissenschaftlich besser handhabbar (Optimierungsproblem).

1.5 Grundlagen und verwandte Disziplinen

KI ist ein interdisziplinäres Feld, das auf vielen Bereichen aufbaut:

- **Philosophie:** Logik, Reasoning, Geist als physisches System, Grundlagen des Lernens.
- **Mathematik:** Formale Repräsentation, Beweise, Algorithmen, Wahrscheinlichkeitstheorie.
- **Psychologie / Kognitionswissenschaft:** Wahrnehmung, Motorik, Anpassung. Kognitionswissenschaft verbindet KI-Modelle mit experimentellen Techniken der Psychologie.
- **Ökonomie:** Entscheidungstheorie, Spieltheorie (rationale Entscheidungen).
- **Neurowissenschaften:** Physisches Substrat (Gehirn) für mentale Aktivitäten.
- **Linguistik:** Wissensrepräsentation, Grammatik.
- **Kontrolltheorie:** Stabilität, optimales Agentendesign.

1.5.1 Taxonomie der KI

Die Begriffe werden oft hierarchisch verstanden:

- **Künstliche Intelligenz (KI):** Der Überbegriff für Technik, die intelligente Züge zeigt.
- **Machine Learning (Maschinelles Lernen):** Ein Teilgebiet der KI, das Algorithmen nutzt, um aus Daten zu lernen (statt explizit programmiert zu werden).
- **Deep Learning:** Ein Teilgebiet des Machine Learning, das auf künstlichen neuronalen Netzen mit vielen Schichten basiert (aktuell sehr erfolgreich, siehe Nobelpreis Physik 2024 an Hinton/Hopfield).

1.6 Grenzen und Probleme moderner KI

Trotz großer Erfolge (z.B. AlphaGo, Stable Diffusion, Protein Folding) existieren signifikante Limitationen:

- **Bias (Voreingenommenheit):** KI-Modelle übernehmen Vorurteile aus den Trainingsdaten (z.B. rassistische Tendenzen in Gesichtserkennung oder Textgenerierung).
- **Adversarial Attacks:** KI kann leicht getäuscht werden. Durch für Menschen unsichtbare Änderungen an einem Bild (Rauschen) kann eine KI dazu gebracht werden, ein Objekt völlig falsch zu klassifizieren (z.B. Panda wird mit 99% Konfidenz als Gibbon erkannt).
- **Halluzinationen:** Generative KIs erzeugen plausibel klingende, aber faktisch falsche Informationen.
- **Isolation:** Aktuelle KI ist meist „Narrow AI“ und auf spezifische Probleme beschränkt, ohne allgemeines Weltverständnis.

1.7 Ergänzung: Agenten (aus der Übung)

Ein zentrales Konzept der KI ist der **Agent**.

- **Agent:** Eine Einheit, die ihre Umgebung über **Sensoren** wahrnimmt und mittels **Aktuatoren** auf diese Umgebung einwirkt.
- **Environment (Umgebung):** Die Welt, in der der Agent operiert.

Zur Beschreibung eines KI-Problems (z.B. Roboterfußball) werden oft folgende Aspekte analysiert:

- **Performance Measure:** Wie wird Erfolg gemessen? (z.B. Anzahl der Tore, gewonnene Spiele).
- **Environment:** Was beinhaltet die Welt? (z.B. Spielfeld, Ball, Gegner, Tore).
- **Actuators:** Wie kann der Agent handeln? (z.B. Motoren für Beine, Schussmechanik).
- **Sensors:** Wie nimmt der Agent wahr? (z.B. Kameras, Beschleunigungssensoren).

Umgebungseigenschaften können klassifiziert werden als:

- *Observable* (vollständig beobachtbar) vs. *Partially Observable*.

- *Deterministic* (Folgezustand durch aktuellen Zustand und Aktion bestimmt) vs. *Stochastic*.
- *Episodic* (Handlungen sind unabhängig voneinander) vs. *Sequential*.
- *Static* (Umgebung ändert sich nicht während der Entscheidungsfindung) vs. *Dynamic*.
- *Discrete* (endliche Anzahl an Zuständen/Aktionen) vs. *Continuous*.

2 AI101-02: AI Systems

Dieses Kapitel behandelt die grundlegenden Bausteine von KI-Systemen: Agenten und ihre Umgebungen. Es definiert Rationalität und klassifiziert verschiedene Arten von Umgebungen und Agentenarchitekturen.

2.1 KI-Systeme und Agenten

Ein KI-System wird definiert durch die Interaktion zwischen einem Agenten und seiner Umgebung.

Definition: Agent

Ein **Agent** ist eine Entität, die ihre Umgebung durch **Sensoren** wahrnimmt und auf diese Umgebung durch **Aktuatoren** (Effektoren) einwirkt.

- **Wahrnehmung (Percept):** Der sensorische Input des Agenten zu einem bestimmten Zeitpunkt.
- **Wahrnehmungssequenz (Percept Sequence):** Die vollständige Historie aller Wahrnehmungen, die der Agent bisher erhalten hat.
- **Agentenfunktion:** Eine mathematische Abbildung von Wahrnehmungssequenzen auf Aktionen: $f : P^* \rightarrow A$.
- **Agentenprogramm:** Die konkrete Implementierung der Agentenfunktion, die auf einer physischen Architektur läuft.

2.2 Rationalität

Ein rationaler Agent ist nicht zwangsläufig ein perfekter oder allwissender Agent. Rationalität bezieht sich auf die Qualität der Entscheidungsfindung basierend auf den verfügbaren Informationen.

Rationaler Agent

Ein **rationaler Agent** wählt für jede mögliche Wahrnehmungssequenz diejenige Aktion aus, die erwartungsgemäß sein Leistungsmaß (*Performance Measure*) maximiert, unter Berücksichtigung der Wahrnehmungssequenz und des eingebauten Wissens.

Rationalität hängt von vier Faktoren ab:

1. Dem **Leistungsmaß** (Performance Measure), das den Erfolg definiert.
2. Dem **Wissen**, das der Agent bereits besitzt (Prior Knowledge).
3. Den **Aktionen**, die der Agent ausführen kann.
4. Der bisherigen **Wahrnehmungssequenz**.

Wichtige Unterscheidung:

- **Allwissenheit (Omniscience):** Kennt das tatsächliche Ergebnis jeder Aktion (in der Realität unmöglich).
- **Rationalität:** Maximiert das *erwartete* Ergebnis basierend auf dem aktuellen Wissen.

2.3 Task Environments (PEAS)

Um einen Agenten zu entwerfen, muss die Aufgabenstellung spezifiziert werden. Dies geschieht durch das PEAS-Modell.

PEAS

- **Performance Measure** (Leistungsmaß): Woran wird Erfolg gemessen?
- **Environment** (Umgebung): Wo operiert der Agent?
- **Actuators** (Aktuatoren): Womit handelt der Agent?
- **Sensors** (Sensoren): Womit nimmt der Agent wahr?

Beispiel: Autonomes Taxi

- **P:** Sicherheit, Schnelligkeit, Legalität, Komfort, Profit.
- **E:** Straßen, anderer Verkehr, Fußgänger, Wetter.
- **A:** Lenkung, Gaspedal, Bremse, Hupe, Blinker.
- **S:** Kameras, Radar, Tacho, GPS, Motorensensoren.

2.4 Eigenschaften von Umgebungen

Die Art der Umgebung bestimmt maßgeblich das Design des Agenten. Umgebungen werden anhand folgender Dimensionen klassifiziert:

1. Fully Observable (Vollständig beobachtbar) vs. Partially Observable:

- *Fully*: Die Sensoren geben zu jedem Zeitpunkt den kompletten Zustand der Umgebung wieder (kein interner Speicher nötig).
- *Partially*: Teile des Zustands sind verdeckt oder Sensoren sind ungenau (interner Zustand/Gedächtnis nötig).

2. Single Agent vs. Multi-Agent:

- *Single*: Der Agent agiert allein (z.B. Kreuzworträtsel).
- *Multi*: Es gibt andere Agenten, die kooperativ oder kompetitiv sein können (z.B. Schach, Straßenverkehr).

3. Deterministic vs. Stochastic:

- *Deterministic*: Der nächste Zustand wird vollständig durch den aktuellen Zustand und die Aktion des Agenten bestimmt.
- *Stochastic*: Es gibt Unsicherheiten/Zufall (z.B. Wetter, Würfelglück). Wenn Wahrscheinlichkeiten unbekannt sind, nennt man es *non-deterministic*.

4. Episodic vs. Sequential:

- *Episodic*: Die Erfahrung ist in atomare Episoden unterteilt. Eine Aktion in Episode *A* hat keine Auswirkung auf Episode *B* (z.B. Bildklassifizierung).
- *Sequential*: Die aktuelle Entscheidung beeinflusst alle zukünftigen Entscheidungen (z.B. Schach, Fahren).

5. Static vs. Dynamic:

- *Static*: Die Umgebung ändert sich nicht, während der Agent "nachdenkt" (z.B. Kreuzworträtsel).
- *Dynamic*: Die Umgebung ändert sich kontinuierlich (z.B. Taxi fahren).
- *Semi-dynamic*: Der Zustand ändert sich nicht, aber die Bewertung (Performance) ändert sich mit der Zeit (z.B. Schach mit Schachuhr).

6. Discrete vs. Continuous:

- Bezieht sich auf die Zustände, Zeit oder Aktionen. Schach ist diskret (feste Felder), Taxi fahren ist kontinuierlich.

2.5 Agententypen

Es gibt vier grundlegende Typen von Agentenprogrammen, aufsteigend nach Komplexität und Fähigkeit.

2.5.1 1. Simple Reflex Agents (Einfache Reflex-Agenten)

Diese Agenten entscheiden nur basierend auf der *aktuellen* Wahrnehmung. Sie ignorieren die Wahrnehmungshistorie.

- **Funktionsweise:** Condition-Action Rules (Wenn-Dann-Regeln).
- **Formel:** Aktion = Funktion(Aktuelle Wahrnehmung).
- **Einschränkung:** Funktionieren nur zuverlässig in *fully observable* Umgebungen. Drohen in unendliche Schleifen zu geraten, wenn die Umgebung nur teilweise beobachtbar ist.

2.5.2 2. Model-based Reflex Agents (Modellbasierte Reflex-Agenten)

Diese Agenten besitzen einen internen Zustand (*Internal State*), um mit partieller Beobachtbarkeit umzugehen.

- **Interner Zustand:** Repräsentiert Wissen über die Welt, das aktuell nicht sichtbar ist (Gedächtnis).
- **Modell der Welt:** Wissen darüber, wie die Welt funktioniert (Übergangsmodelle) und wie Aktionen die Welt verändern.
- **Ablauf:** Update State → Wähle Regel → Aktion.

2.5.3 3. Goal-based Agents (Zielbasierte Agenten)

Wissen über den Zustand reicht nicht aus; der Agent benötigt ein Ziel (*Goal*).

- **Planung/Suche:** Der Agent simuliert verschiedene Sequenzen von Aktionen, um zu sehen, ob sie zum Ziel führen.
- **Unterschied zum Reflex:** Der Reflex-Agent reagiert, der Goal-based Agent plant in die Zukunft.
- **Flexibilität:** Ziele können sich ändern, der Agent passt sein Verhalten an.

2.5.4 4. Utility-based Agents (Nutzenbasierte Agenten)

Ziele sind oft binär (erreicht/nicht erreicht). Nutzen (*Utility*) erlaubt eine feinere Unterscheidung.

- **Utility Function:** Bildet einen Zustand auf eine reelle Zahl ab, die den Grad der "Zufriedenheit" des Agenten angibt.
- **Vorteil:** Ermöglicht rationale Entscheidungen bei
 - Zielkonflikten (z.B. Geschwindigkeit vs. Sicherheit).
 - Unsicherheit (Abwägung von Erfolgswahrscheinlichkeit und Nutzen).

2.5.5 5. Learning Agents (Lernende Agenten)

Jeder der obigen Agententypen kann lernfähig gemacht werden. Dies erlaubt dem Agenten, in unbekannten Umgebungen zu operieren und kompetenter zu werden.

Komponenten eines lernenden Agenten

- **Performance Element:** Der eigentliche Agent (einer der 4 obigen Typen), der Entscheidungen trifft.
- **Critic (Kritiker):** Bewertet, wie gut der Agent ist, basierend auf einem festen Leistungsstandard.
- **Learning Element:** Nutzt das Feedback des Kritikers, um das Performance Element zu verbessern.
- **Problem Generator:** Schlägt Aktionen vor, die zu neuen Erfahrungen führen (Exploration), auch wenn diese kurzfristig suboptimal sind.

3 AI101-03 Uninformed and Informed Search

3.1 Einführung und Problemdefinition

Problemlösende Agenten sind zielbasierte Agenten, die atomare Repräsentationen verwenden (Zustände als Black Boxes). Der Prozess besteht aus vier Phasen:

1. **Zielformulierung:** Definition des Ziels basierend auf der aktuellen Situation.
2. **Problemformulierung:** Entscheidung über zu betrachtende Aktionen und Zustände.
3. **Suche:** Prozess des Findens einer Aktionssequenz, die zum Ziel führt.
4. **Ausführung:** Durchführung der gefundenen Aktionen.

Wohlformuliertes Suchproblem

Ein Suchproblem wird durch fünf Komponenten definiert:

1. **Initial State (Startzustand)** s_0 : Der Zustand, in dem der Agent beginnt.
2. **Actions (Aktionen)** $A(s)$: Die Menge der möglichen Aktionen in einem Zustand s .
3. **Transition Model (Überführungsmodell)** $Result(s, a)$: Beschreibt, was eine Aktion tut. Rückgabe ist der Folgezustand.
4. **Goal Test (Zieltest)**: Bestimmt, ob ein Zustand ein Zielzustand ist.
5. **Path Cost (Pfadkosten)** $c(s, a, s')$: Additive Kostenfunktion. Meistens sind Schrittkosten nicht-negativ.

Lösung: Eine Sequenz von Aktionen, die vom Startzustand zum Ziel führt.

Optimale Lösung: Eine Lösung mit den geringsten Pfadkosten.

3.2 Suchalgorithmen: Tree Search vs. Graph Search

Der Kern aller Suchalgorithmen ist die Expansion von Zuständen.

- **Frontier (Open List):** Menge der generierten, aber noch nicht expandierten Knoten.
- **Expansion:** Anwenden der Aktionen auf einen Zustand, um Kindknoten zu generieren.

Unterschied Tree vs. Graph Search

- **Tree Search:** Verfolgt nicht, welche Zustände bereits besucht wurden. Kann in Schleifen geraten und redundante Pfade mehrfach besuchen.
- **Graph Search:** Speichert besuchte Zustände in einer *Explored Set* (Closed List), um Redundanz und Schleifen zu vermeiden.

3.2.1 Bewertungskriterien für Suchstrategien

- **Completeness (Vollständigkeit):** Findet der Algorithmus garantiert eine Lösung, wenn eine existiert?
- **Optimality (Optimalität):** Findet er die kostengünstigste Lösung?
- **Time Complexity:** Wie lange dauert die Suche? (Anzahl generierter Knoten).
- **Space Complexity:** Wie viel Speicher wird benötigt? (Maximale Anzahl Knoten im Speicher).

Parameter der Komplexität:

- b : Verzweigungsfaktor (Branching factor) - max. Anzahl Nachfolger eines Knotens.
- d : Tiefe des flachsten Zielknotens.

- m : Maximale Tiefe des Suchraums (kann ∞ sein).

3.3 Uninformierte Suche (Blind Search)

Uninformierte Strategien haben keine Information darüber, wie nah ein Zustand am Ziel ist. Sie unterscheiden sich nur in der Reihenfolge der Knotenexpansion.

3.3.1 Breadth-First Search (BFS) - Breitensuche

Expandiert den flachsten Knoten in der Frontier zuerst (FIFO-Queue).

- **Vollständig:** Ja (wenn b endlich).
- **Optimal:** Ja, aber nur wenn alle Schrittkosten gleich sind (z.B. 1). Sonst nicht.
- **Zeit:** $O(b^d)$ (Exponentiell).
- **Speicher:** $O(b^d)$ (Jeder generierte Knoten muss gespeichert werden).

Problem: Speicherbedarf ist das größte Problem der BFS.

3.3.2 Uniform-Cost Search (UCS)

Expandiert den Knoten mit den geringsten Pfadkosten $g(n)$ zuerst (Priority Queue).

- Äquivalent zu BFS, wenn alle Schrittkosten gleich sind.
- **Vollständig:** Ja (wenn Kosten $\epsilon > 0$).
- **Optimal:** Ja.
- **Komplexität:** Hängt von den Kosten ab, kann schlechter als b^d sein, wenn viele Schritte mit kleinen Kosten existieren.

3.3.3 Depth-First Search (DFS) - Tiefensuche

Expandiert den tiefsten Knoten in der Frontier zuerst (LIFO-Queue / Stack).

- **Vollständig:** Nein (kann in unendlichen Pfaden oder Schleifen hängen bleiben, außer bei Graph Search in endlichen Räumen).
- **Optimal:** Nein (findet irgendeinen Pfad, nicht zwingend den kürzesten).
- **Zeit:** $O(b^m)$. Schlecht, wenn $m \gg d$.
- **Speicher:** $O(b \cdot m)$ (Linear!). Nur der aktuelle Pfad und Geschwisterknoten werden gespeichert.

3.3.4 Depth-Limited Search (DLS)

DFS mit einem vordefinierten Tiefenlimit l .

- Löst das Endlos-Pfad-Problem der DFS.
- Unvollständig, wenn Lösung tiefer als l ($d > l$).
- Nicht optimal.

3.3.5 Iterative Deepening Search (IDS)

Kombiniert die Vorteile von BFS (Vollständigkeit) und DFS (Speichereffizienz). Führt DLS mit Limit $l = 0, 1, 2, \dots$ nacheinander aus.

- **Vollständig:** Ja.
- **Optimal:** Ja (bei gleichen Schrittkosten).
- **Zeit:** $O(b^d)$. Knoten werden mehrfach generiert, aber da die unterste Ebene die Mehrheit ausmacht, ist der Overhead gering (ca. 11% mehr Aufwand bei $b = 10$).

- **Speicher:** $O(b \cdot d)$ (Linear).

Fazit: IDS ist oft die bevorzugte uninformierte Suchmethode für große Suchräume mit unbekannter Tiefe.

3.4 Informierte Suche (Heuristische Suche)

Nutzt problem spezifisches Wissen in Form einer **Heuristikfunktion** $h(n)$, um die Suche zu lenken.

Heuristik $h(n)$

$h(n)$ = geschätzte Kosten vom Knoten n zum Ziel.

- $h(n) \geq 0$
- Für Zielknoten gilt $h(\text{Goal}) = 0$.

3.4.1 Greedy Best-First Search

Expandiert den Knoten, der dem Ziel am nächsten scheint.

- **Bewertungsfunktion:** $f(n) = h(n)$.
- **Vollständig:** Nein (wie DFS, kann in Schleifen geraten).
- **Optimal:** Nein.
- **Zeit/Speicher:** $O(b^m)$ im schlechtesten Fall. Gute Heuristiken können dies drastisch verbessern.

3.4.2 A* Search (A-Star)

Kombiniert UCS und Greedy. Minimiert die geschätzten Gesamtkosten des Pfades durch n .

- **Bewertungsfunktion:** $f(n) = g(n) + h(n)$
- $g(n)$: Tatsächliche Kosten vom Start bis n .
- $h(n)$: Geschätzte Kosten von n bis zum Ziel.
- $f(n)$: Geschätzte Gesamtkosten des Pfades durch n .

3.5 Heuristiken für A*

Damit A* optimal ist, muss die Heuristik bestimmte Eigenschaften erfüllen.

3.5.1 Admissibility (Zulässigkeit)

Eine Heuristik $h(n)$ ist **admissible**, wenn sie die Kosten zum Ziel *niemals überschätzt*.

$$0 \leq h(n) \leq h^*(n)$$

(wobei $h^*(n)$ die wahren Kosten zum Ziel sind).

- Notwendig für Optimalität bei **Tree Search**.
- Beispiel Luftlinie: Die direkte Distanz ist immer kürzer oder gleich der Straßenentfernung.

3.5.2 Consistency (Konsistenz / Monotonie)

Eine Heuristik $h(n)$ ist **consistent**, wenn für jeden Knoten n und jeden Nachfolger n' gilt:

$$h(n) \leq c(n, a, n') + h(n')$$

(Dreiecksungleichung).

- Notwendig für Optimalität bei **Graph Search**.

- Konsistenz impliziert Admissibility.
- Bei konsistenten Heuristiken steigen die $f(n)$ -Werte entlang eines Pfades monoton an (oder bleiben gleich).

3.5.3 Dominanz von Heuristiken

Wenn $h_2(n) \geq h_1(n)$ für alle n (und beide zulässig sind), dann **dominiert** h_2 die Heuristik h_1 .

- h_2 ist näher an den wahren Kosten (h^*).
- A* mit h_2 expandiert weniger Knoten als mit h_1 und ist effizienter.

3.5.4 Beispiel: 8-Puzzle Heuristiken

- $h_{MIS}(n)$: Anzahl der falsch platzierten Kacheln (Misplaced Tiles).
- $h_{MAN}(n)$: Summe der Manhattan-Distanzen aller Kacheln zu ihrer Zielposition.

Es gilt: $h_{MAN}(n) \geq h_{MIS}(n)$. Daher dominiert die Manhattan-Distanz die "Misplaced Tiles"-Heuristik und ist für A* besser geeignet.

4 AI101-04: Local Search and Adversarial Search

Diese Einheit behandelt Suchstrategien für zwei spezielle Problemklassen:

- **Lokale Suche:** Optimierungsprobleme, bei denen der Pfad zur Lösung irrelevant ist und nur der Endzustand zählt (z.B. N-Damen-Problem, Chip-Design).
- **Adversarial Search:** Probleme, bei denen ein Agent gegen einen Gegner agiert (Spiele wie Schach oder Go).

4.1 Lokale Suche (Local Search)

Bei vielen Optimierungsproblemen ist der Zustandsraum riesig oder unendlich, aber der Weg zum Ziel ist unwichtig. Lokale Suchalgorithmen operieren auf einem einzelnen aktuellen Zustand (oder einer kleinen Menge) und bewegen sich nur zu dessen Nachbarn.

Eigenschaften der Lokalen Suche

- **Speichereffizienz:** Verbraucht meist konstanten Speicher ($O(1)$ oder $O(k)$).
- **Anwendung:** Geeignet für riesige oder kontinuierliche Zustandsräume.
- **Ziel:** Finden des globalen Optimums einer **Zielfunktion** (Objective Function).

4.1.1 Hill Climbing (Bergsteigen)

Hill Climbing ist der einfachste lokale Suchalgorithmus ("Gierige lokale Suche"). Er versucht kontinuierlich, den aktuellen Zustand zu verbessern, indem er zum besten Nachbarn wechselt.

Algorithmus:

1. Starte mit einem zufälligen Zustand.
2. Generiere alle Nachbarn des aktuellen Zustands.
3. Wähle den Nachbarn mit der besten Bewertung (höchster Wert der Zielfunktion).
4. Wenn der beste Nachbar besser ist als der aktuelle Zustand: Gehe dorthin.
5. Sonst: Terminiere (Gipfel erreicht).

Metapher: „Das Erklimmen des Mount Everest bei dichtem Nebel und Amnesie.“

Probleme des Hill Climbing: Der Algorithmus garantiert nicht das Finden des globalen Optimums, da er in lokalen Optima stecken bleiben kann.

- **Lokales Maximum:** Ein Gipfel, der höher ist als alle direkten Nachbarn, aber niedriger als das globale Maximum.
- **Plateau:** Ein flacher Bereich, in dem alle Nachbarn den gleichen Wert haben. Der Algorithmus hat keine Richtungsinformation (Random Walk notwendig).
 - **Flat Local Maximum:** Ein Plateau, das ein lokales Maximum ist.
 - **Shoulder:** Ein Plateau, von dem aus es noch weiter bergauf gehen könnte.
- **Ridge (Grat):** Eine schmale Erhebung, die ansteigt, bei der aber alle direkten Nachbarn (z.B. Norden, Süden, Osten, Westen) bergab führen. Der Anstieg verläuft oft diagonal, was für einfache Bewegungsoperatoren schwer zu erkennen ist.

Varianten zur Verbesserung:

- **Stochastic Hill Climbing:** Wählt zufällig einen der besseren Nachbarn aus (nicht zwingend den besten). Dies erhöht die Chance, lokale Maxima zu umgehen oder Plateaus zu überwinden.

- **Random-Restart Hill Climbing:** Führt den Algorithmus mehrfach mit unterschiedlichen zufälligen Startzuständen aus.
 - Wenn die Anzahl der Versuche gegen unendlich geht, nähert sich die Wahrscheinlichkeit, das globale Optimum zu finden, 1 an (asymptotisch vollständig).

4.1.2 Simulated Annealing

Inspiziert vom physikalischen Prozess des Ausglühens in der Metallurgie (Erhitzen und langsames Abkühlen, um stabile Kristallstrukturen zu bilden). Ziel ist es, lokale Maxima zu verlassen, indem man *schlechte* Züge mit einer gewissen Wahrscheinlichkeit zulässt.

Funktionsweise

Kombiniert Hill Climbing (Effizienz) mit Random Walk (Exploration).

- Es gibt einen Temperaturparameter T , der gemäß einem Abkühlungsplan (*schedule*) sinkt.
- Ein zufälliger Nachbar wird gewählt.
- Ist der Nachbar besser ($\Delta E > 0$): Der Zug wird immer akzeptiert.
- Ist der Nachbar schlechter ($\Delta E < 0$): Der Zug wird mit Wahrscheinlichkeit P akzeptiert:

$$P = e^{\frac{\Delta E}{T}}$$

Interpretation:

- Hohes T : Hohe Wahrscheinlichkeit, schlechte Züge zu akzeptieren (ähnlich Random Walk).
- $T \rightarrow 0$: Wahrscheinlichkeit sinkt gegen 0 (ähnlich Hill Climbing).
- Wird T langsam genug gesenkt, findet der Algorithmus garantiert das globale Optimum.

4.1.3 Local Beam Search

Statt nur einen Zustand zu betrachten, verfolgt dieser Algorithmus k Zustände parallel.

- Starte mit k zufälligen Zuständen.
- Generiere in jedem Schritt alle Nachfolger aller k Zustände.
- Wenn einer davon das Ziel ist: Stopp.
- Sonst: Wähle die k besten Nachfolger aus der *gesamten* Menge aller Nachfolger aus.

Unterschied zu k mal Random-Restart: Die k Zustände sind nicht unabhängig. Informationen werden geteilt, da sich die Suche auf vielversprechende Regionen des Zustandsraums konzentriert („Wo ein guter Zustand ist, sind oft auch andere“).

4.2 Suche in kontinuierlichen Räumen

Viele reale Probleme (z.B. Training neuronaler Netze) haben kontinuierliche Zustandsräume.

4.2.1 Gradient Descent (Gradientenabstieg)

Wenn die Zielfunktion $f(\mathbf{x})$ differenzierbar ist, nutzen wir den Gradienten ∇f , um die Richtung des steilsten Anstiegs/Abstiegs zu finden.

Update-Regel

Um eine Kostenfunktion $L(\theta)$ zu minimieren, aktualisieren wir die Parameter θ iterativ:

$$\theta \leftarrow \theta - \alpha \nabla L(\theta)$$

Dabei ist α die **Lernrate** (Step Size).

Wahl der Lernrate α :

- **Zu klein:** Konvergenz ist sehr langsam, viele Iterationen nötig.
- **Zu groß:** Der Algorithmus kann über das Ziel hinausschießen, oszillieren oder sogar divergieren.

4.3 Adversarial Search (Spiele)

In Multi-Agenten-Umgebungen beeinflussen die Aktionen anderer Agenten das Ergebnis. Wir betrachten **Nullsummenspiele** (Zero-Sum Games) mit perfekter Information (z.B. Schach, Tic-Tac-Toe).

- **MAX:** Unser Agent, möchte den Nutzen (Utility) maximieren.
- **MIN:** Der Gegner, möchte den Nutzen minimieren (bzw. seinen eigenen maximieren).

4.3.1 Minimax-Algorithmus

Der Minimax-Algorithmus berechnet den optimalen Zug durch rekursive Suche im Spielbaum.

Minimax-Wert eines Knotens:

- **Terminal-Knoten (Blatt):** Utility-Wert des Zustands (z.B. +1 Sieg, -1 Niederlage, 0 Unentschieden).
- **MAX-Knoten:** $\max(\text{Werte der Nachfolger})$.
- **MIN-Knoten:** $\min(\text{Werte der Nachfolger})$.

Eigenschaften:

- **Vollständig:** Ja, bei endlichem Baum.
- **Optimal:** Ja, gegen einen optimalen Gegner.
- **Zeitkomplexität:** $O(b^m)$ (exponentiell), mit Verzweigungsfaktor b und Tiefe m .
- **Platzkomplexität:** $O(b \cdot m)$ (bei Tiefensuche).

4.3.2 Alpha-Beta Pruning

Alpha-Beta Pruning ist eine Optimierung des Minimax-Algorithmus, die irrelevante Zweige des Suchbaums ignoriert („abschneidet“), ohne das Ergebnis zu verändern.

Idee: Wenn wir bereits einen Zug gefunden haben, der uns einen gewissen Wert garantiert, und wir in einem anderen Zweig sehen, dass der Gegner uns dort auf einen schlechteren Wert zwingen kann, müssen wir diesen Zweig nicht weiter untersuchen.

Parameter α und β

Wir führen zwei Werte durch die Rekursion mit:

- **α (Alpha):** Der Wert der besten Alternative für MAX, die bisher auf dem Pfad gefunden wurde (untere Schranke für MAX). Initial: $-\infty$.
- **β (Beta):** Der Wert der besten Alternative für MIN, die bisher auf dem Pfad gefunden wurde (obere Schranke für MIN). Initial: $+\infty$.

Pruning-Regeln:

1. MIN-Knoten (Update β):

- Berechne Wert v des Kindknotens.
- Wenn $v \leq \alpha$: **Pruning!** (MAX wird diesen Ast nie wählen, da er α bereits sicher hat).
- Sonst: $\beta = \min(\beta, v)$.

2. MAX-Knoten (Update α):

- Berechne Wert v des Kindknotens.

- Wenn $v \geq \beta$: **Pruning!** (MIN wird diesen Ast nie zulassen, da er β bereits sicher hat).
- Sonst: $\alpha = \max(\alpha, v)$.

Effizienz:

- Im besten Fall (perfekte Sortierung der Züge): Komplexität reduziert sich auf $O(b^{m/2})$. Das bedeutet effektiv eine Verdopplung der durchsuchbaren Tiefe.
- Im schlechtesten Fall (schlechteste Sortierung): Keine Verbesserung, $O(b^m)$.
- *Move Ordering* ist daher entscheidend (z.B. Schlagen von Figuren zuerst prüfen).

4.3.3 Umgang mit Ressourcenbeschränkungen

Da komplette Suchbäume für komplexe Spiele (Schach: 10^{40} Zustände, Go: 10^{170}) zu groß sind, wird die Suche oft bei einer Tiefe d abgebrochen.

- **Heuristische Evaluierungsfunktion (H-Minimax):** Schätzt den Wert einer Position an den Blättern der begrenzten Suche (z.B. Materialwert im Schach: Bauer=1, Dame=9).
- **Horizon Effect:** Ein unvermeidbarer negativer Event (z.B. Verlust der Dame) wird durch „Verzögerungstaktiken“ aus dem Suchhorizont geschoben, sodass die Evaluierung fälschlicherweise positiv wirkt.

5 Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems (CSPs) stellen einen Paradigmenwechsel gegenüber der klassischen Pfadsuche dar. Während bei Planungsproblemen der Pfad zum Ziel entscheidend ist, interessiert bei CSPs nur der Zielzustand selbst (Identifikationsproblem). Der Zustand ist dabei nicht mehr atomar ("Blackbox"), sondern faktorisiert, d. h. er besteht aus Variablen und Werten.

5.1 Definition und Komponenten

Ein CSP wird durch drei Komponenten definiert:

1. **Variablen** $X = \{X_1, X_2, \dots, X_n\}$: Die Objekte, denen Werte zugewiesen werden müssen.
2. **Domänen** (Wertebereiche) $D = \{D_1, D_2, \dots, D_n\}$: Für jede Variable X_i gibt es eine Menge D_i von möglichen Werten.
3. **Constraints** (Beschränkungen) C : Eine Menge von Regeln, die spezifizieren, welche Kombinationen von Werten für Teilmengen der Variablen zulässig sind.

Zustandszuweisungen

- **Partielle Zuweisung**: Nur einigen Variablen wurden Werte zugewiesen (z. B. $X_1 = v_1$).
- **Konsistente (legale) Zuweisung**: Eine Zuweisung, die keinen Constraint verletzt.
- **Vollständige Zuweisung**: Jeder Variablen ist ein Wert zugewiesen.
- **Lösung**: Eine vollständige und konsistente Zuweisung.

Knoten = Variablen (Regionen), Kanten = Constraints (benachbart)

5.2 Arten von Constraints

Constraints schränken die möglichen Belegungen der Variablen ein:

- **Unäre Constraints**: Betreffen eine einzelne Variable (z. B. $SA \neq \text{green}$).
- **Binäre Constraints**: Betreffen Paare von Variablen (z. B. $SA \neq WA$). Dies ist die häufigste Form und kann durch einen **Constraint-Graphen** dargestellt werden.
- **Constraints höherer Ordnung**: Betreffen 3 oder mehr Variablen (z. B. bei Kryptogrammen wie $TWO + TWO = FOUR$).
- **Soft Constraints (Präferenzen)**: Dienen der Optimierung (z. B. "Rot ist besser als Grün"). Diese wandeln das CSP oft in ein *Constrained Optimization Problem* um.

5.3 Lösungsansätze: Suche und Backtracking

Da die Reihenfolge der Zuweisungen bei CSPs keine Rolle spielt (Kommutativität), muss nicht der gesamte Zustandsraum als Baum mit $n! \cdot v^n$ Blättern durchsucht werden. Es reicht aus, pro Ebene eine Variable zu betrachten. Dies reduziert den Suchraum auf v^n Blätter.

5.3.1 Backtracking Search

Backtracking ist eine Tiefensuche (Depth-First Search), die für CSPs optimiert ist.

- Wähle eine unzugewiesene Variable.
- Probiere nacheinander alle Werte aus der Domäne.
- Wenn ein Wert konsistent ist: Weise ihn zu und rekursiere.

- Wenn ein Wert inkonsistent ist oder die Rekursion fehlschlägt: Mache die Zuweisung rückgängig (Backtrack) und versuche den nächsten Wert.

Ohne Heuristiken entspricht dies einer uninformierten Suche. Um die Effizienz zu steigern (z. B. Lösung des n -Damen-Problems für $n > 25$), werden Heuristiken für die Auswahl von Variablen und Werten sowie Inferenzmethoden (Propagation) benötigt.

5.4 Heuristiken für Backtracking

Um die Suche zu beschleunigen, müssen an jedem Entscheidungspunkt drei Fragen beantwortet werden:

5.4.1 1. Welche Variable soll als nächste belegt werden?

Hier gilt das Prinzip: *Fail-first* (Scheitere so früh wie möglich, um große Teile des Suchbaums abzuschneiden).

Variablen-Auswahl

- **Minimum Remaining Values (MRV):** Wähle die Variable mit den *wenigsten* verbleibenden legalen Werten. Dies führt am schnellsten zu einem Fehler, wenn ein Zweig keine Lösung enthält.
- **Degree Heuristic:** (Wird oft als Tie-Breaker für MRV genutzt). Wähle die Variable, die an den *meisten* Constraints mit anderen *noch nicht zugewiesenen* Variablen beteiligt ist.

5.4.2 2. Welchen Wert soll die Variable annehmen?

Werte-Auswahl

Least Constraining Value (LCV): Vervorzug des Werts, der die meisten benachbarten Variablen ausschließt. Dies

Hier gilt das Prinzip: *Fail-last* (Lasse möglichst viele Optionen für die Zukunft offen).

5.5 Constraint Propagation (Inferenz)

Anstatt nur zu suchen und bei Konflikten zurückzugehen, kann man durch Inferenz den Suchraum proaktiv verkleinern, indem man Werte ausschließt, die unmöglich Teil einer Lösung sein können.

5.5.1 Konsistenzarten

- **Node Consistency (Knotenkonsistenz):** Jede Variable erfüllt ihre unären Constraints.
- **Arc Consistency (Kantenkonsistenz):** Eine Variable X ist kantenkonsistent zu einer Variable Y , wenn für *jeden* Wert $x \in D_X$ mindestens ein Wert $y \in D_Y$ existiert, der den binären Constraint zwischen X und Y erfüllt.
- **Path Consistency / k-Consistency:** Verallgemeinerung auf Tripel oder k Variablen. Stärkere Konsistenz prüft mehr, ist aber rechenaufwendiger.

5.5.2 Forward Checking vs. Arc Consistency

- **Forward Checking:** Sobald X ein Wert zugewiesen wird, werden alle inkonsistenten Werte aus den Domänen der direkten Nachbarn entfernt. Wenn eine Domäne leer wird, backtrackt der Algorithmus. *Nachteil:* Erkennt Konflikte nicht früh genug (sieht z. B. nicht, dass zwei zukünftige Variablen inkonsistent zueinander geworden sind).
- **Arc Consistency (AC-3 Algorithmus):** Propagiert Constraints durch das gesamte Netzwerk. Wenn aus D_X ein Wert entfernt wird, müssen alle Nachbarn von X erneut geprüft werden, da deren Unterstützung verloren gegangen sein könnte.

AC-3 Algorithmus Kernidee

Verwende eine Warteschlange (Queue) aller Kanten (Arcs). Solange die Queue nicht leer ist:

1. Entnimm Kante (X_i, X_j) .
2. Entferne alle Werte aus D_i , die keinen Partner in D_j haben.
3. Wenn Werte aus D_i entfernt wurden: Füge alle Kanten (X_k, X_i) (Nachbarn, die von X_i abhängen) wieder zur Queue hinzu.

Die Kombination aus Backtracking und AC-3 wird als **MAC** (Maintaining Arc Consistency) bezeichnet.

5.6 Lokale Suche für CSPs

Für Probleme, bei denen der Pfad egal ist, kann auch Lokale Suche (Hill Climbing, Simulated Annealing) auf *vollständigen* (aber inkonsistenten) Zuweisungen arbeiten.

Min-Conflicts Heuristik:

- Wähle zufällig eine Variable, die einen Constraint verletzt.
- Wähle für diese Variable den Wert, der die *Anzahl der verletzten Constraints* minimiert.

Diese Methode ist erstaunlich effizient (z. B. Million-Queens in Minuten), außer in einem kritischen Bereich des Verhältnisses von Constraints zu Variablen (*Critical Ratio*).

5.7 Struktur von CSPs

Die Struktur des Constraint-Graphen beeinflusst die Komplexität der Lösung massiv.

5.7.1 Problem-Dekomposition

Wenn ein Graph in unabhängige Teilgraphen zerfällt, kann die Komplexität von $O(d^n)$ auf $O(n/c \cdot d^c)$ reduziert werden (lineare Skalierung in n)

5.7.2 Baumstrukturierte CSPs

Wenn der Constraint-Graph ein Baum ist (keine Zyklen), kann das CSP in linearer Zeit $O(n \cdot d^2)$ gelöst werden. **Algorithmus für Bäume:** *Topologische Sortierung:* Ordne Variablen so, dass jeder Knoten nach seinem Elternknoten kommt (Wurzel bis Blätter). *Rückwärts-Pass (Konsistenz):* Mache von den Blättern zur Wurzel hin alle Kanten gerichtete kantenkonsistent (entferne Werte im Elternknoten, die keine Entsprechung im Kind haben). *Vorwärts-Pass (Zuweisung):* Weise von der Wurzel zu den Blättern Werte zu. Da die Konsistenz hergestellt wurde, gibt es kein Backtracking.

5.7.3 Fast-Baum-Strukturen (Cutset Conditioning)

Für Graphen, die "fast" Bäume sind: Identifiziere ein **Cycle Cutset** (Menge von Variablen, deren Entfernung den Graphen zum Baum macht). Instantiiere die Variablen im Cutset (probiere alle Kombinationen). Löse den verbleibenden Baum für jede Instantiierung ("Residual CSP"). Die Laufzeit ist exponentiell nur in der Größe des Cutsets, nicht in n .

6 AI101-06: Logik und KI 1 - Aussagenlogik

6.1 Einführung: Wissensbasierte Agenten

Während reflexbasierte Agenten oder Suchalgorithmen oft nur über begrenztes Verständnis ihrer Umgebung verfügen, nutzen wissensbasierte Agenten explizite Repräsentationen von Wissen, um Schlussfolgerungen zu ziehen und neue Fakten abzuleiten.

Komponenten eines wissensbasierten Agenten

Das Herzstück ist die **Knowledge Base (KB)**: Eine Menge von Sätzen (Sentences) in einer formalen Sprache, die Fakten über die Welt repräsentieren.

- **TELL**: Operation zum Hinzufügen neuen Wissens zur KB.
- **ASK**: Operation zum Abfragen von Wissen. Der Agent muss ableiten können, was aus der KB folgt.

6.2 Die Wumpus-Welt

Die Wumpus-Welt ist eine Standardumgebung zur Illustration logischer Agenten.

- **Umgebung**: 4×4 Gitter, Start bei $[1,1]$.
- **Elemente**: Wumpus (stinkt), Gruben (erzeugen Luftzug), Gold (glitzert).
- **Wahrnehmungen (Percepts)**: $[Stench, Breeze, Glitter, Bump, Scream]$.
- **Eigenschaften**: Deterministisch, diskret, statisch, partiell beobachtbar.

6.3 Logik: Syntax und Semantik

- **Syntax**: Definiert die zulässigen Sätze (Formeln) der Sprache.
- **Semantik**: Definiert die "Wahrheit" von Sätzen in Bezug auf eine mögliche Welt.
- **Modell (m)**: Eine mathematische Abstraktion, die jedem Symbol einen Wahrheitswert zuweist.

Entailment (Logische Folgerung)

Ein Satz α folgt logisch aus der Wissensbasis KB (geschrieben $KB \models \alpha$), wenn in *jedem* Modell, in dem KB wahr ist, auch α wahr ist.

$$M(KB) \subseteq M(\alpha)$$

(Die Menge der Modelle, die die KB erfüllen, ist eine Teilmenge der Modelle, die α erfüllen.)

6.4 Aussagenlogik (Propositional Logic)

Die Aussagenlogik ist die einfachste Form der Logik, basierend auf Fakten, die wahr oder falsch sein können.

6.4.1 Syntax der Aussagenlogik

- **Atomsätze**: Einzelne Symbole (z.B. $P, Q, W_{1,3}$), die für Propositionen stehen.
- **Komplexe Sätze**: Werden durch logische Verknüpfungen gebildet.

Operatoren (nach absteigender Präzedenz):

1. \neg (Negation / Nicht)
2. \wedge (Konjunktion / Und)

3. \vee (Disjunktion / Oder)
4. \Rightarrow (Implikation / Wenn... dann)
5. \Leftrightarrow (Bikonditional / Genau dann, wenn)

6.4.2 Semantik: Wahrheitstabellen

Die Semantik wird durch Wahrheitstabellen definiert.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Table 2: Wahrheitstabelle. **Wichtig:** Die Implikation $P \Rightarrow Q$ ist nur falsch, wenn die Prämisse P wahr und die Konklusion Q falsch ist.

6.5 Inferenz (Schlussfolgern)

6.5.1 Model Checking

Ein einfacher Inferenz-Algorithmus ist die **Truth Table Enumeration**: 1. Iteriere über alle möglichen Modelle (Belegungen der Variablen). 2. Prüfe, ob in allen Modellen, in denen die KB wahr ist, auch α wahr ist. Dies ist *sound* (korrekt) und *complete* (vollständig), aber ineffizient ($O(2^n)$).

6.5.2 Logische Eigenschaften

- **Gültigkeit (Tautologie):** Ein Satz ist in *allen* Modellen wahr (z.B. $P \vee \neg P$).
- **Erfüllbarkeit (Satisfiability):** Ein Satz ist in *mindestens einem* Modell wahr.
- **Unerfüllbarkeit (Contradiction):** Ein Satz ist in *keinem* Modell wahr (z.B. $P \wedge \neg P$).

Wichtiger Zusammenhang (Beweis durch Widerspruch):

$$KB \models \alpha \quad \text{genau dann, wenn} \quad (KB \wedge \neg \alpha) \text{ ist unerfüllbar.}$$

6.5.3 Logische Äquivalenzen

Zwei Sätze sind äquivalent ($\alpha \equiv \beta$), wenn sie in denselben Modellen wahr sind. Wichtige Umformungen für die Prüfung:

- **De Morgan:** $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$ und $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$
- **Implikations-Eliminierung:** $P \Rightarrow Q \equiv \neg P \vee Q$
- **Bikonditional-Eliminierung:** $P \Leftrightarrow Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$
- **Distributivgesetze:** $(P \wedge (Q \vee R)) \equiv ((P \wedge Q) \vee (P \wedge R))$

6.6 Resolution

Die Resolution ist ein Inferenzverfahren, das Widersprüche aufdeckt. Es arbeitet auf Sätzen in der **Konjunktiven Normalform (CNF)**.

CNF (Conjunctive Normal Form)

Eine Konjunktion von Klauseln. Jede Klausel ist eine Disjunktion von Literalen. Beispiel: $(A \vee \neg B) \wedge (B \vee C \vee \neg D)$

6.6.1 Umwandlung in CNF

Jeder aussagenlogische Satz kann in CNF umgewandelt werden: 1. Eliminiere \Leftrightarrow . 2. Eliminiere \Rightarrow (ersetze $A \Rightarrow B$ durch $\neg A \vee B$). 3. Verschiebe \neg nach innen (De Morgan, doppelte Negation). 4. Wende Distributivgesetz an (\vee über \wedge).

6.6.2 Resolutions-Algorithmus

Um zu zeigen, dass $KB \models \alpha$: 1. Füge $\neg\alpha$ zur KB hinzu: $KB \wedge \neg\alpha$. 2. Wandle alles in CNF um. 3. Wende wiederholt die **Resolutionsregel** an:

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

wobei l_i und m_j komplementäre Literale sind (z.B. P und $\neg P$). 4. Wenn die **leere Klausel** (Widerspruch) abgeleitet wird, ist α bewiesen.

6.7 Horn-Klauseln und Chaining

Resolution ist mächtig, aber NP-vollständig. Für eingeschränkte Formen gibt es effizientere Algorithmen (lineare Zeit).

- **Definite Klausel:** Genau ein positives Literal. (Äquivalent zu einer Implikation: $(A \wedge B) \Rightarrow C$).
- **Horn-Klausel:** Höchstens ein positives Literal.

6.7.1 Algorithmen für Definite Klauseln

- **Forward Chaining:** Startet bei den bekannten Fakten in der KB und wendet Regeln an, um neue Fakten zu generieren, bis das Ziel (Query) erreicht ist. (Datengetrieben).
- **Backward Chaining:** Startet beim Ziel (Query) und sucht rückwärts nach Regeln, die dieses Ziel beweisen können. (Zielgetrieben).

Beide basieren auf der *Modus Ponens* Regel: $\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$.

6.8 Grenzen der Aussagenlogik

- **Mangelnde Ausdruckskraft:** Keine Objekte oder Relationen.
- **Regel-Explosion:** Für jede Instanz muss eine eigene Regel geschrieben werden (z.B. $Breeze_{1,1} \Leftrightarrow \dots$, $Breeze_{1,2} \Leftrightarrow \dots$).
- **Lösung:** Prädikatenlogik erster Stufe (First-Order Logic).

7 Logic and AI 2: First-Order Logic (FOL)

7.1 Einführung und Motivation

Die Aussagenlogik (Propositional Logic) hat erhebliche Einschränkungen. Sie behandelt Fakten atomar und besitzt kein Verständnis für Objekte und deren Beziehungen untereinander.

- In der Aussagenlogik haben Terme keine formale Bedeutung, nur eine intuitive (z. B. ist “RoommateCarryingUmbrella” technisch gesehen nur eine Variable P).
- **Prädikatenlogik erster Stufe** (First-Order Logic, FOL) führt Objekte, Variablen und Quantoren ein, um Wissen kompakter und strukturierter darzustellen.

7.2 Syntax und Elemente der FOL

Die FOL baut auf folgenden Grundbausteinen auf:

Elemente der FOL

- **Objekte:** Konstanten, die spezifische Entitäten bezeichnen (z. B. $John, Umbrella_0, Earth$).
- **Relationen (Prädikate):** Beziehungen zwischen Objekten.
 - *Unäre Relationen:* Eigenschaften eines Objekts (z. B. $IsUmbrella(x)$).
 - *n-äre Relationen:* Beziehungen zwischen n Objekten (z. B. $Carrying(John, Umbrella_0)$).
- **Funktionen:** Abbildungen, die sich auf Objekte beziehen, ohne sie explizit zu benennen (z. B. $Roommate(Person_0)$ oder $LeftLegOf(John)$). Wichtig: Eine Funktion liefert ein Objekt zurück, keinen Wahrheitswert.
- **Gleichheit:** $Term_1 = Term_2$ (z. B. $Roommate(Person_0) = Person_1$).

7.2.1 Quantoren

Um Aussagen über Mengen von Objekten zu treffen, werden Variablen (x, y, z) und Quantoren verwendet:

1. **Allquantor** (\forall): “Für alle...”

- Beispiel: Alle Löwen sind Katzen.
- $\forall x : Lion(x) \implies Cat(x)$
- *Hinweis:* Der Allquantor wird meistens mit der Implikation (\implies) verwendet.

2. **Existenzquantor** (\exists): “Es existiert (mindestens) ein...”

- Beispiel: Es gibt eine Katze, die kein Löwe ist.
- $\exists x : Cat(x) \wedge \neg Lion(x)$
- *Hinweis:* Der Existenzquantor wird meistens mit der Konjunktion (\wedge) verwendet.

Dualität der Quantoren: Die Quantoren lassen sich ineinander umformen:

$$\forall x : P(x) \equiv \neg \exists x : \neg P(x)$$

$$\exists x : P(x) \equiv \neg \forall x : \neg P(x)$$

7.3 Modellierung: Von natürlicher Sprache zu FOL

Das Übersetzen von Sätzen erfordert präzise Muster. Hier einige wichtige Beispiele und Strukturen (auch basierend auf Übungsaufgaben):

- **Einfache Eigenschaft:** “John hat einen Regenschirm” $\exists y : (Has(John, y) \wedge IsUmbrella(y))$
- **Verschachtelung:** “Jede Person, die einen Schirm hat, ist nicht nass” $\forall x : (Person(x) \implies ((\exists y : Has(x, y) \wedge Umbrella(y)) \implies \neg Wet(x)))$
- **Mindestens zwei (Distinctness):** “John hat mindestens zwei Schirme” $\exists x, y : (Has(John, x) \wedge Umbrella(x) \wedge Has(John, y) \wedge Umbrella(y) \wedge \neg(x = y))$
- **Höchstens zwei:** “John hat höchstens zwei Schirme” $\forall x, y, z : ((Has(John, x) \dots \wedge Has(John, z) \dots) \implies (x = y \vee x = z \vee y = z))$
- **Genau ein (Uniqueness):** “Es liegt genau eine Münze in der Kiste” $\exists x (Coin(x) \wedge InBox(x) \wedge \forall y ((Coin(y) \wedge InBox(y)) \implies x = y))$

7.4 Inferenz in FOL

7.4.1 Substitution (SUBST)

Um logische Schlüsse zu ziehen, müssen Variablen oft durch konkrete Terme ersetzt werden.

- **Syntax:** $SUBST(\{x/John\}, IsHealthy(x)) \rightarrow IsHealthy(John)$
- **Universal Instantiation:** Aus $\forall x : \alpha$ kann $SUBST(\{x/g\}, \alpha)$ für jeden Grundterm g abgeleitet werden.
- **Existential Instantiation:** Aus $\exists x : \alpha$ kann $SUBST(\{x/k\}, \alpha)$ abgeleitet werden, wobei k eine **neue** Konstante ist (Skolem-Konstante), die noch nicht in der Wissensbasis vorkommt.

7.4.2 Skolemisierung

Die Eliminierung von Existenzquantoren ist ein zentraler Schritt für Resolutionsbeweise. Dabei ist die Position des Quantors entscheidend:

Skolemisierung Regeln

1. **Existenzquantor steht allein (oder ganz außen):** Ersetze die Variable durch eine neue Konstante (Skolem-Konstante).

$$\exists x : P(x) \rightarrow P(A)$$

2. **Existenzquantor steht im Wirkungsbereich eines Allquantors:** Die existierende Variable hängt von der allquantifizierten Variable ab. Ersetze y durch eine **Skolem-Funktion** $f(x)$.

$$\forall x \exists y : IsParentOf(x, y) \rightarrow \forall x : IsParentOf(x, f(x))$$

Hierbei bildet $f(x)$ das passende y für jedes x ab.

7.4.3 Unifikation

Ein Algorithmus, der eine Substitution θ findet, sodass zwei Ausdrücke identisch werden. Dies ist die Grundlage für den verallgemeinerten Modus Ponens und die Resolution. Beispiel:

- Term 1: $Knows(John, x)$
- Term 2: $Knows(y, Jane)$
- Unifikator $\theta = \{x/Jane, y/John\}$ resultiert in $Knows(John, Jane)$.

7.5 Resolution in FOL

Wie in der Aussagenlogik ist die Resolution eine Widerlegungsmethode (Beweis durch Widerspruch). Um sie anzuwenden, müssen Sätze in die **Konjunktive Normalform (CNF)** gebracht werden.

Umwandlung in First-Order CNF

1. **Implikationen eliminieren:** $A \implies B$ wird zu $\neg A \vee B$.
2. **Negationen nach innen ziehen:** (De Morgan Regeln), sodass \neg nur direkt vor Atomen steht.
3. **Variablen standardisieren:** Umbenennen, sodass jeder Quantor eindeutige Variablennamen verwendet.
4. **Skolemierung:** Eliminierung aller Existenzquantoren (durch Konstanten oder Funktionen).
5. **Allquantoren verwerfen:** Da nun alle Variablen allquantifiziert sind, können die \forall weggelassen werden (implizite Annahme).
6. **Verteilung (Distributivgesetz):** Umwandlung in Konjunktion von Disjunktionen (UND von ODERs).
7. **Klauselbildung:** Darstellung als Menge von Klauseln.

Beispielablauf: Satz: “Jeder, der nicht getötet wird und isst, ist Nahrung”

$$\forall x \forall y : (Eats(x, y) \wedge \neg Killed(x)) \implies Food(y)$$

\rightarrow Implikation weg: $\neg(Eats(x, y) \wedge \neg Killed(x)) \vee Food(y) \rightarrow$ De Morgan: $\neg Eats(x, y) \vee Killed(x) \vee Food(y)$ Dies ist bereits eine Klausel.

7.6 Prolog (Programming in Logic)

Prolog ist eine Programmiersprache, die auf FOL (speziell Horn-Klauseln) basiert und Unifikation sowie Backtracking-Suche verwendet.

- **Fakten:** ‘eats(sam, dal).’ (Entspricht $Eats(Sam, Dal)$).
- **Regeln:** ‘head :- body.’ (Bedeutet: Head ist wahr, *wenn* Body wahr ist. Logisch: $Body \implies Head$).
- **Syntax:**
 - Großbuchstaben sind Variablen (z. B. ‘Person’, ‘X’).
 - Kleinbuchstaben sind Atome/Konstanten (z. B. **sam**, **curry**).
 - `_` ist eine anonyme Variable.
 - Komma `,` bedeutet UND (Konjunktion).
- **Inferenz:** Prolog nutzt Tiefensuche (Depth-First Search) mit Backtracking. Wenn eine Unifikation fehlschlägt, geht das System zurück und probiert die nächste Alternative.

7.7 Grenzen der Prädikatenlogik

- **Entscheidbarkeit:** FOL ist **semi-entscheidbar**.
 - Wenn ein Satz aus der Wissensbasis folgt, existiert ein Algorithmus, der dies beweist (er terminiert).
 - Wenn ein Satz *nicht* folgt, terminiert der Algorithmus möglicherweise nie.
- **Ausdrucksstärke:** Man kann nicht über Relationen selbst quantifizieren (z. B. “Für alle Eigenschaften $p \dots$ ”). Dies würde Logik höherer Stufe (Higher-Order Logic) erfordern.
- **Gödels Unvollständigkeitssatz:** In jedem formalen System, das mächtig genug ist, um Arithmetik (Induktion) abzubilden, gibt es wahre Aussagen, die innerhalb des Systems nicht beweisbar sind. Vollständigkeit in komplexen Systemen ist unmöglich.

7.8 Neuro-Symbolic AI (Ausblick)

Moderne Ansätze versuchen, die logische Schlussfolgerung (Symbole, Prolog) mit neuronalen Netzen (Wahrnehmung, Lernen) zu kombinieren.

- Ziel: Ein System, das sowohl aus Daten lernt (Gradient Descent) als auch logische Regeln befolgt.

- Beispiel: Ein Agent, der Pixeldaten wahrnimmt (Neural), aber Entscheidungen auf Basis logischer Regeln (z. B. “Wenn Gegner nah, dann springen”) trifft oder diese Regeln lernt.

8 Umgang mit Unsicherheit und Probabilistisches Schließen

Dieses Kapitel behandelt die Grundlagen der Unsicherheit in der Künstlichen Intelligenz, die Wahrscheinlichkeitstheorie als Werkzeug zur Modellierung von Glaubensgraden (Degrees of Belief) sowie Bayesian Networks zur kompakten Repräsentation von Wissen und Inferenzmethoden.

8.1 Einführung in Unsicherheit

In der klassischen Logik gehen Agenten davon aus, dass Aussagen entweder wahr oder falsch sind und Aktionen deterministische Ergebnisse haben. Die reale Welt ist jedoch oft unsicher.

Gründe für Unsicherheit

- **Partielle Beobachtbarkeit:** Der Agent hat keinen Zugriff auf den vollständigen Zustand der Welt (z.B. unbekannte Verkehrslage).
- **Rauschende Sensoren:** Messwerte können fehlerhaft sein.
- **Unsicherheit in Aktionsergebnissen:** Aktionen gelingen nicht immer wie geplant (z.B. Reifenpanne).
- **Komplexität:** Es ist unmöglich, alle Eventualitäten ("Qualification Problem") in logischen Regeln zu modellieren (z.B. "Ich komme pünktlich an, außer...").

Anstatt strikter logischer Wahrheit modellieren wir **Wahrscheinlichkeiten** als Maß für den **Glaubensgrad** (Degree of Belief) eines Agenten. Eine Wahrscheinlichkeit von 0.1 für einen Stau bedeutet nicht, dass die Straße zu 10

8.2 Grundlagen der Wahrscheinlichkeitstheorie

8.2.1 Wahrscheinlichkeitsraum und Axiome

Ein Wahrscheinlichkeitsmodell besteht aus einem **Stichprobenraum** Ω (Menge aller möglichen Welten/Ergebnisse). Ein Ereignis A ist eine Teilmenge von Ω .

Kolmogorov-Axiome

Diese Axiome beschränken die Menge rationaler Glaubenssätze:

1. Alle Wahrscheinlichkeiten liegen zwischen 0 und 1: $0 \leq P(A) \leq 1$.
2. Wahre Aussagen haben Wahrscheinlichkeit 1, falsche 0: $P(\text{true}) = 1, P(\text{false}) = 0$.
3. Die Wahrscheinlichkeit einer Disjunktion (Vereinigung):

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

Ein Verstoß gegen diese Axiome führt zu irrationalem Verhalten. Dies wird durch das **Dutch Book Theorem** bewiesen: Wenn ein Agent Wahrscheinlichkeiten akzeptiert, die den Axiomen widersprechen, kann ein Gegner (Bookmaker) eine Wette so konstruieren, dass der Agent garantiert Geld verliert, unabhängig vom Ausgang des Ereignisses.

8.2.2 Zufallsvariablen

Anstatt mit atomaren Ereignissen zu arbeiten, nutzen wir **Zufallsvariablen** (Random Variables, RVs), um Komplexität zu reduzieren.

- **Diskret:** Endliche Menge von Werten (z.B. $Wetter \in \{\text{sonnig}, \text{regen}, \text{schnee}\}$). Werte müssen disjunkt und erschöpfend sein. **Kontinuierlich:** Unendlicher Wertebereich (z.B. Temperatur).

Eine Proposition (Aussage) wie $Wetter = \text{regen}$ entspricht der Menge aller atomaren Ereignisse, in denen diese Aussage wahr ist.

8.3 Verteilungen und Rechenregeln

8.3.1 Joint Distribution (Verbundwahrscheinlichkeit)

Die **Joint Distribution** $P(X_1, \dots, X_n)$ weist jeder möglichen Kombination von Werten aller Zufallsvariablen eine Wahrscheinlichkeit zu.

$$P(x, y) = P(X = x \wedge Y = y)$$

Die Tabelle der Joint Distribution wächst exponentiell mit der Anzahl der Variablen ($O(2^n)$ bei binären Variablen), erlaubt aber die Beantwortung jeder beliebigen Wahrscheinlichkeitsanfrage.

8.3.2 Marginalisierung und Bedingte Wahrscheinlichkeit

Marginalisierung

Man kann Wahrscheinlichkeiten von Teilmengen von Variablen berechnen, indem man über die nicht interessierenden Variablen summiert ("Summing Out"):

$$P(Y) = \sum_{x \in X} P(x, Y)$$

Bedingte Wahrscheinlichkeit

Die Wahrscheinlichkeit von X , gegeben Evidenz Y :

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

Daraus folgt die **Produktregel**:

$$P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

Die **Kettenregel** (Chain Rule) erlaubt die Zerlegung einer Joint Distribution in bedingte Wahrscheinlichkeiten:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, \dots, X_{i-1})$$

8.3.3 Satz von Bayes

Der Satz von Bayes ist fundamental für das Lernen aus Beobachtungen. Er erlaubt die Inversion von bedingten Wahrscheinlichkeiten.

$$\underbrace{P(H|E)}_{\text{Posterior}} = \frac{\underbrace{P(E|H)}_{\text{Likelihood}} \cdot \underbrace{P(H)}_{\text{Prior}}}{\underbrace{P(E)}_{\text{Evidenz/Marginalisierung}}}$$

Erweitert mit Marginalisierung im Nenner:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E|H)P(H) + P(E|\neg H)P(\neg H)}$$

- **Prior**: Wahrscheinlichkeit der Hypothese vor Evidenz.
- **Likelihood**: Wahrscheinlichkeit der Evidenz, angenommen die Hypothese ist wahr.
- **Posterior**: Neue Wahrscheinlichkeit der Hypothese nach Beobachtung der Evidenz.

8.4 Bayesian Networks

Da die vollständige Joint Distribution ($O(2^n)$) speicher- und rechenintensiv ist, nutzen wir Unabhängigkeiten zur kompakten Darstellung.

8.4.1 Unabhängigkeit

Zwei Variablen X und Y sind **unabhängig**, wenn $P(X|Y) = P(X)$ bzw. $P(X, Y) = P(X)P(Y)$. Viel häufiger ist die **bedingte Unabhängigkeit**: X und Y sind unabhängig gegeben Z , wenn $P(X|Y, Z) = P(X|Z)$.

8.4.2 Definition und Semantik

Ein **Bayesian Network (BN)** ist ein gerichteter azyklischer Graph (DAG), bestehend aus:

- **Knoten**: Zufallsvariablen.
- **Kanten**: Direkte Abhängigkeiten ($X \rightarrow Y$ bedeutet X beeinflusst Y).
- **CPTs (Conditional Probability Tables)**: Jeder Knoten X_i hat eine Wahrscheinlichkeitsverteilung $P(X_i|Pa(X_i))$, wobei $Pa(X_i)$ die Elternknoten sind.

BN Semantik

Ein BN definiert die Joint Distribution als Produkt der lokalen bedingten Wahrscheinlichkeiten:

$$P(X_1, \dots, X_n) = \prod_{i=1}^n P(X_i|Pa(X_i))$$

Lokale Markov-Annahme: Ein Knoten ist bedingt unabhängig von all seinen Nicht-Nachfahren (Non-Descendants), gegeben seine Eltern.

8.5 Inferenz in Bayesian Networks

Inferenz beantwortet Anfragen der Form $P(Query|Evidence)$.

8.5.1 Exakte Inferenz: Variable Elimination

Variable Elimination vermeidet die Berechnung der vollen Joint Distribution, indem Summen “nach innen gezogen” werden. Der Prozess eliminiert Variablen, die weder Query noch Evidenz sind, nacheinander.

Algorithmus:

1. Schreibe die Joint Distribution als Produkt der Faktoren (CPTs).
2. Wähle eine Eliminationsreihenfolge für die versteckten Variablen.
3. Für jede zu eliminierende Variable Z :
 - Sammle alle Faktoren, die Z enthalten.
 - Multipliziere diese Faktoren zu einem neuen temporären Faktor.
 - Summiere Z aus diesem Faktor heraus (Marginalisierung).
 - Ersetze die alten Faktoren durch den neuen Faktor.
4. Normalisiere das Ergebnis am Ende.

Die Operationen sind:

- **Faktor-Produkt**: $f_3(A, B, C) = f_1(A, B) \cdot f_2(B, C)$.
- **Summing Out**: $f_{new}(B) = \sum_a f(a, B)$.

8.5.2 Komplexität

Exakte Inferenz in Bayesian Networks ist **NP-hard** (bewiesen durch Reduktion auf 3-SAT). In einfach verbundenen Netzen (Polytrees) ist die Komplexität linear, im schlimmsten Fall jedoch exponentiell.

8.6 Approximative Inferenz: Sampling

Da exakte Inferenz oft zu aufwendig ist, nutzen wir stochastische Verfahren (Monte Carlo Methoden), die gegen die wahre Wahrscheinlichkeit konvergieren.

8.6.1 Direct Sampling (ohne Evidenz)

Man zieht Stichproben in topologischer Reihenfolge (von Eltern zu Kindern) entsprechend der CPTs.

$$P(X) \approx \frac{N(X)}{N_{total}}$$

8.6.2 Rejection Sampling

Um $P(X|e)$ zu berechnen: Generiere Samples wie oben. Verwirf (reject) alle Samples, die der Evidenz e widersprechen. Zähle die verbleibenden Samples. Problem: Bei unwahrscheinlicher Evidenz werden fast alle Samples verworfen (ineffizient).

8.6.3 Markov Chain Monte Carlo (MCMC)

MCMC-Verfahren wie **Gibbs Sampling** wandern durch den Zustandsraum, anstatt Samples unabhängig zu generieren.

Markov Blanket

Der **Markov Blanket** einer Variable X besteht aus:

- Eltern von X
- Kindern von X
- Anderen Eltern der Kinder von X

X ist bedingt unabhängig von allen anderen Knoten im Netzwerk, gegeben seinen Markov Blanket.

ieren.

1. Initialisiere alle Variablen mit zufälligen Werten (Evidenz fixieren).
2. Wiederhole für viele Schritte:
 - Wähle eine Nicht-Evidenz-Variable X_i .
 - Ziehe einen neuen Wert für X_i aus der Verteilung $P(X_i|\text{Markov Blanket}(X_i))$.
3. Schätze die Wahrscheinlichkeit basierend auf der Häufigkeit der Zustände in den gesammelten Samples.

Dies erzeugt eine Markov-Kette, deren stationäre Verteilung der gesuchten Posterior-Verteilung entspricht (unter der Bedingung, dass die Kette irreduzibel und ergodisch ist).