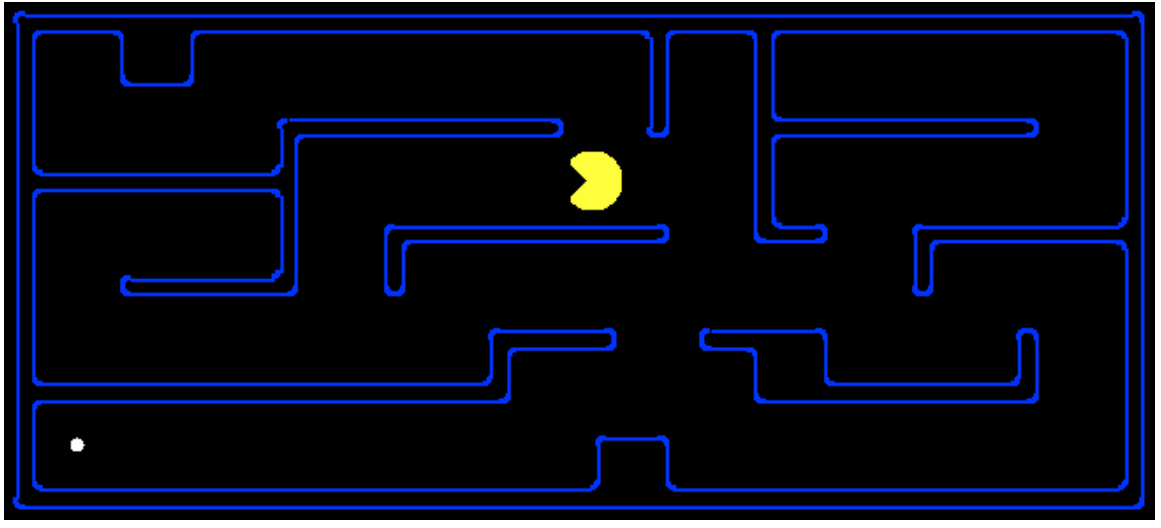
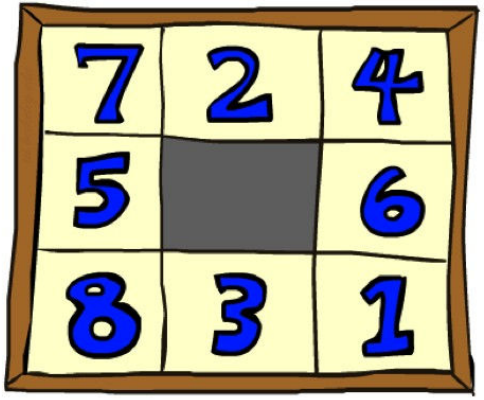
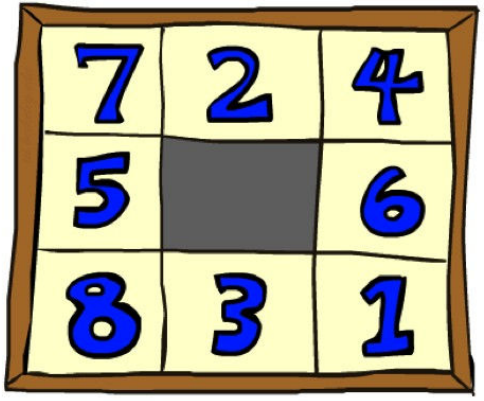
Was ist der kürzeste Weg?

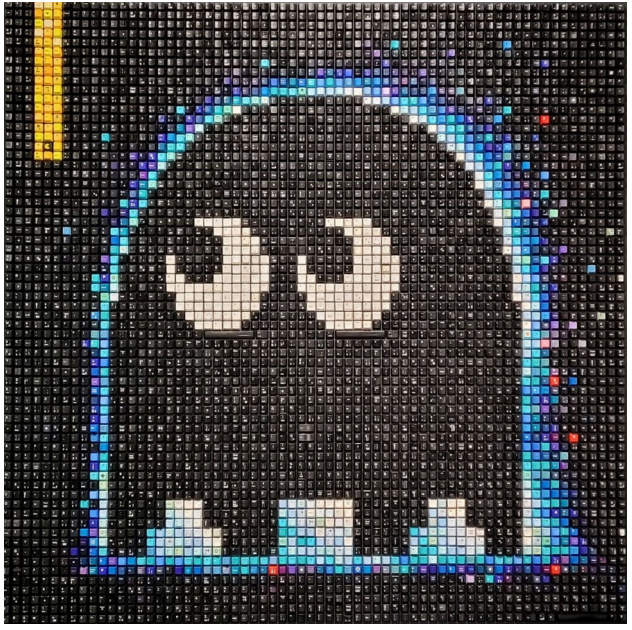


Wie sortiere ich die Zahlen?



Wie sortiere ich die Zahlen in möglichst wenigen Zügen mit wenig Rechenaufwand?





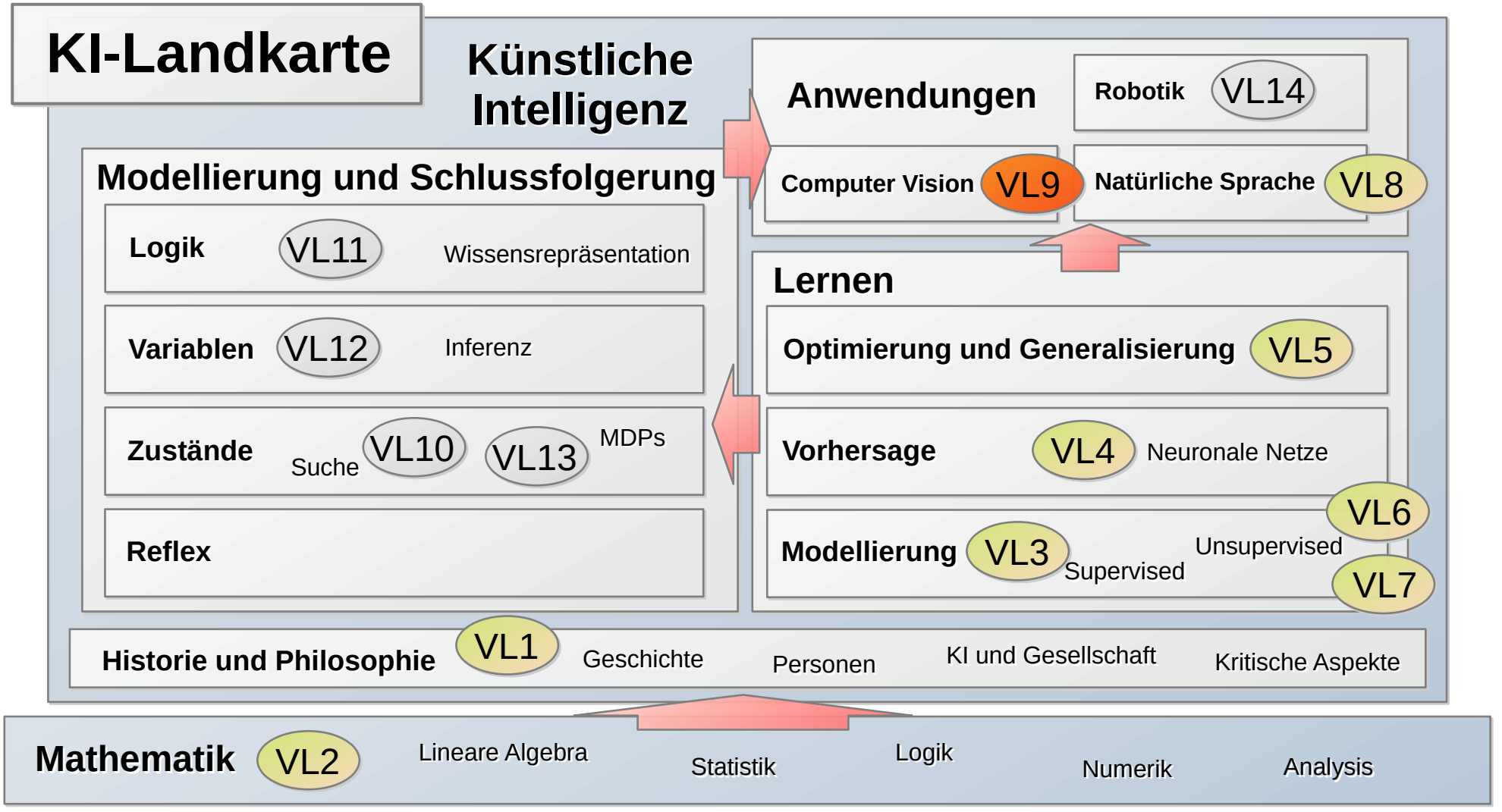
Grundlagen der Künstlichen Intelligenz

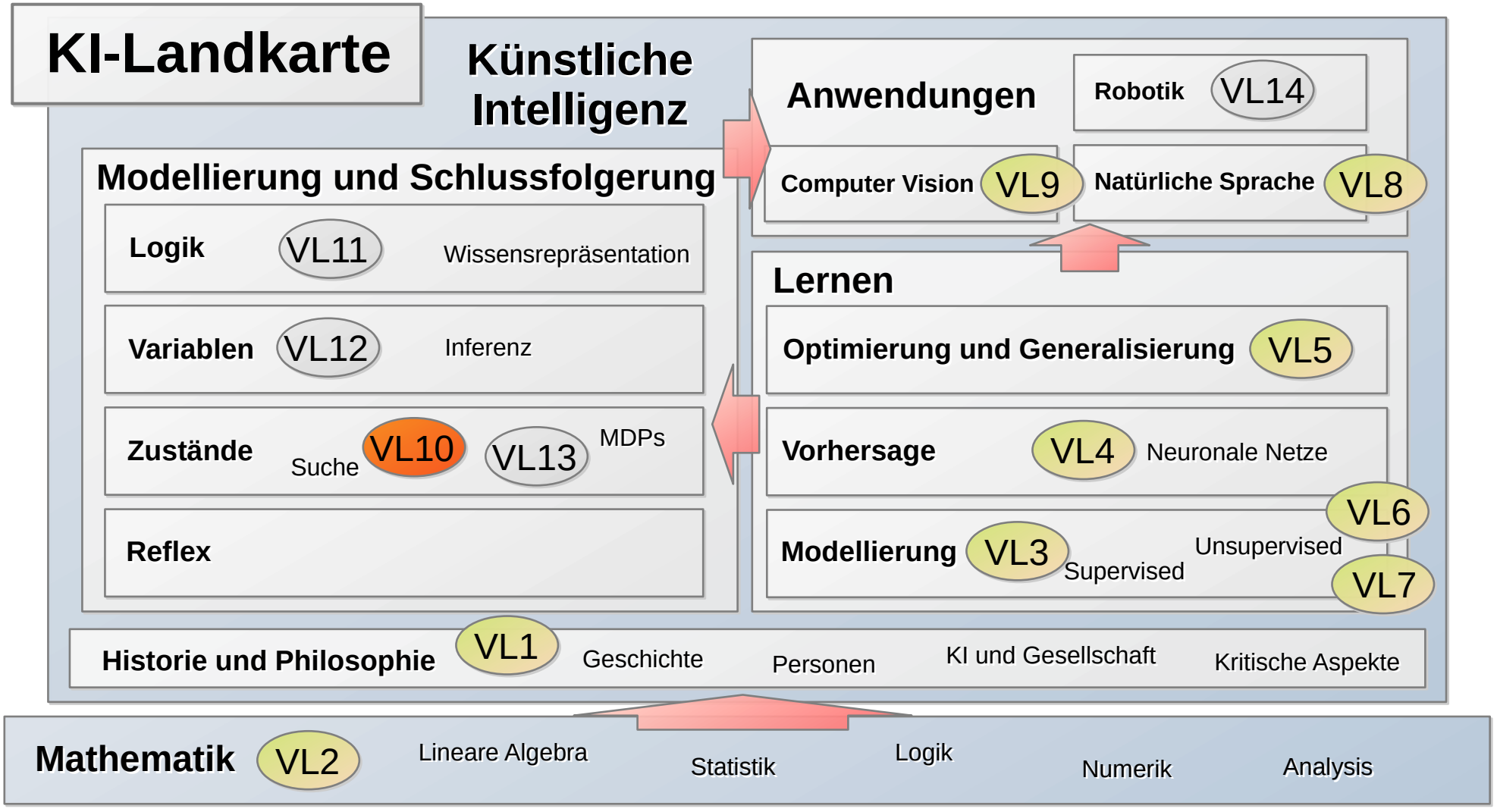
Wintersemester 25/26

Vorlesung 11

Suche, Reasoning, Planning

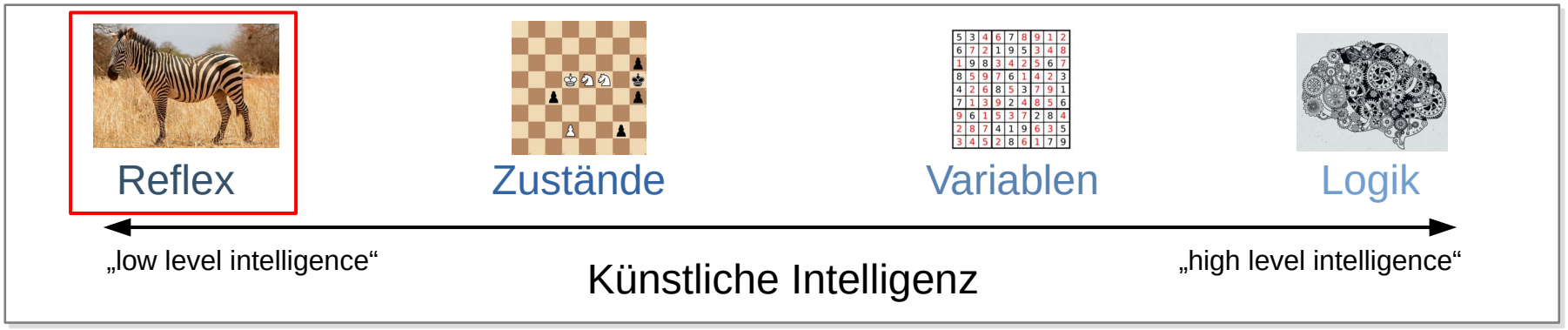
Prof. Dr. Pascal Friederich
T.T.-Prof. Dr. Peer Nowack


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL8
Modellierung und Schlussfolgerung
Logik
VL11
Wissensrepräsentation
Variablen
VL12
Inferenz
Zustände
Suche
VL10
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL5
Vorhersage
VL4
Neuronale Netze
Modellierung
VL3
Supervised
Unsupervised
VL6
VL7
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis


KI-Landkarte
Künstliche Intelligenz
Anwendungen
Robotik
VL14
Computer Vision
VL9
Natürliche Sprache
VL8
Modellierung und Schlussfolgerung
Logik
VL11
Wissensrepräsentation
Variablen
VL12
Inferenz
Zustände
Suche
VL10
VL13
MDPs
Reflex
Lernen
Optimierung und Generalisierung
VL5
Vorhersage
VL4
Neuronale Netze
Modellierung
VL3
Supervised
Unsupervised
VL6
VL7
Historie und Philosophie
VL1
Geschichte
Personen
KI und Gesellschaft
Kritische Aspekte
Mathematik
VL2
Lineare Algebra
Statistik
Logik
Numerik
Analysis

Bisher: Machine Learning

Reflex
- Direktes Mapping von Input zu Output



Anwendungen
- Vorhersage (Klassifikation, Regression)
- Computer Vision: Klassifikation, Objekterkennung, …
- Sprache: Speech-to-text, Übersetzung, ...

Heute

Suche
- Rationale Planung mit klarem Ziel
- Welt modelliert als Zustände

Anwendungen
- Spiele (Schach, Go, Pacman, …)
- Allgemein: Planung, Strategie
- Robotik: Bewegungsplanung

  → Planung durch Zustände und Aktionen

Übersicht

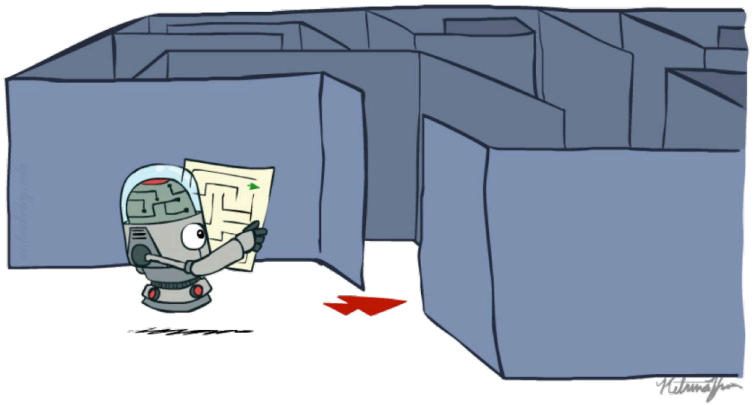
Agenten die planen und handeln
- Rationalität
- Agentendesign

Suchprobleme
- Umgebungen mit Zustandsräumen
- Zustandsräume definieren

Uninformierte Suchverfahren
- Tiefensuche („Depth-First Search“)
- Breitensuche (“Breadth-First Search“)
- Uniform-Cost Suche (“Uniform-Cost Search“)

Heuristische Suchverfahren
- A* Tree Suche

Logik- und Wissens-Repräsentationen



Diese Vorlesung basiert auf Folien von Dan Klein und Pieter Abbeel: Kurs CS188 “Intro to AI”, UC Berkeley, http://ai.berkeley.edu

Teil 1

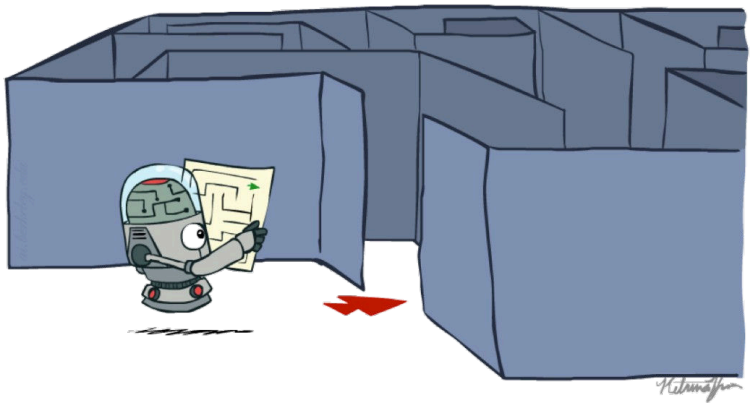
Agenten die planen und handeln
- Rationalität
- Agentendesign

Suchprobleme
- Umgebungen mit Zustandsräumen
- Zustandsräume definieren

Uninformierte Suchverfahren
- Tiefensuche („Depth-First Search“)
- Breitensuche (“Breadth-First Search“)
- Uniform-Cost Suche (“Uniform-Cost Search“)

Heuristische Suchverfahren
- A* Tree Suche

Logik- und Wissens-Repräsentationen



Intelligente Agenten

Viele Definitionen

- **Hier:** Ein intelligenter Agent ist ein **rationaler Entscheidungsträger**, der in der Lage ist, ein externes Umfeld **wahrzunehmen** und **autonom** darauf zu reagieren.

Wichtige Punkte

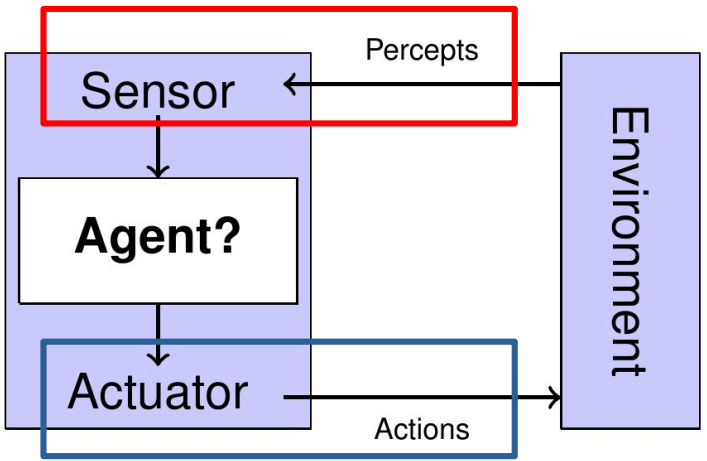
- **Rationalität**: Optimierung eines Leistungsmaßes → Klares Ziel
- **Autonomie**: Das Verhalten wird von der eigenen Wahrnehmung geleitet

→ Wir sind hauptsächlich an rationalen, autonomen Agenten interessiert.

Autonomie

Ein **autonomer** Agent …

- … **nimmt** seine Umgebung mit **wahr**: **Input**
- … **agiert** mit der Umgebung: **Aktionen**



Kreislauf

- Aktionen beeinflussen die Umwelt ...
- ... was sich auf den **Input** auswirkt ...
- ... was sich auf die **Aktionen** auswirkt ...

Rationalität

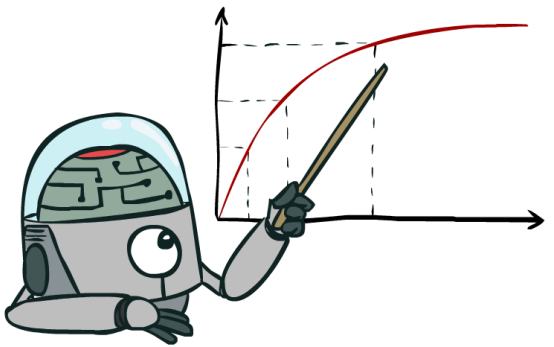
Ein **rationaler Agent** wählt Handlungen, die seinen erwarteten **Nutzen** (“utility”) **maximieren**.

Heute

- Agenten haben ein Ziel und Kosten
- Beispiel: Erreiche das Ziel mit geringsten Kosten

Später

- Agenten haben einen numerischen Nutzen und Belohnungen
- Beispiel: Maximiere die Gesamtbelohnung
- Belohnung: Größter Gesamtgewinn oder erwarteter Gesamtgewinn

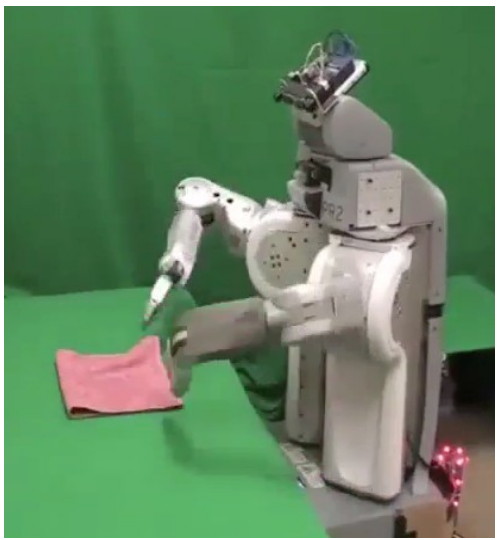


Maximiere den Nutzen!

Nutzen

Klare Nutzenfunktion

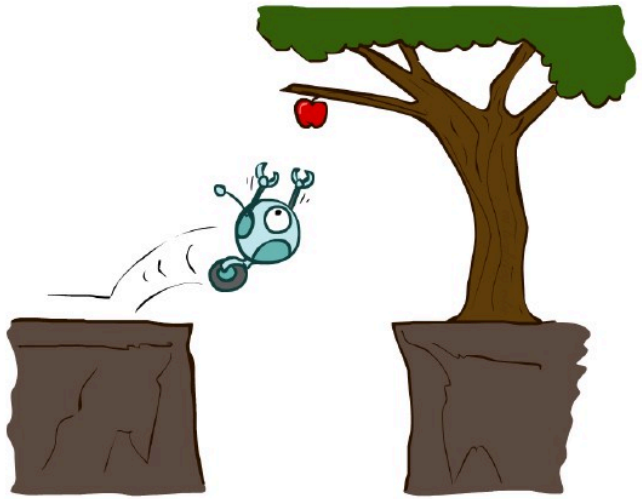
Nicht ganz klare Nutzenfunktion



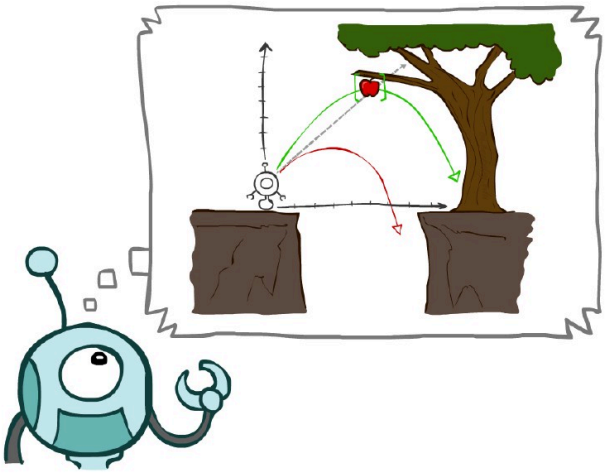
Selbst klare Nutzenfunktion kann irreführend sein

Planende Agenten und Reflex-Agenten

Reflex



Planung



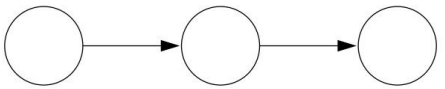
Reflex Agenten …
… wählen eine Handlung auf Grundlage der aktuellen Wahrnehmung (plus ggf. Erinnerung).
… bedenken nur, wie die Welt **ist**.
… denken **nicht** an die zukünftigen Konsequenzen ihres Handelns.

Planende Agenten…
… entscheidungen auf der Grundlage von (hypothetischen) Handlungsfolgen.
… müssen ein Modell für Auswirkungen von Handlungen haben und wie die Welt **sein würde**.
… müssen ein Ziel formulieren.

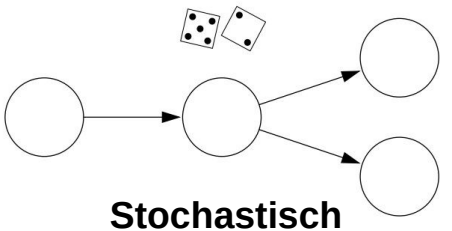
Design von Agenten

Umgebungseigenschaften bestimmen das Design des Agenten:

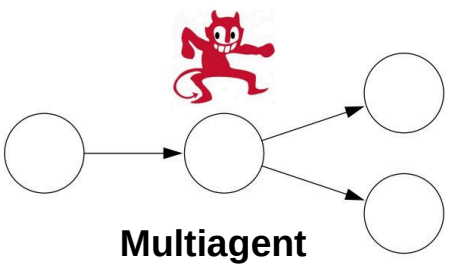
- **Vollständig/teilweise beobachtbar**
  → Ggf. **Erinnerung/Speicher** benötigt (interner Zustand)
  Bsp: Pacman vs. 1st-person Labyrinth
- **Diskret/kontinuierlich**
  → Der Agent kann eventuell nicht **alle Zustände** aufzählen
  Bsp: Pacman vs. Bootsrennen
- **Deterministisch/Stochastisch**
  → Agent muss ggf. mit **Unsicherheit** umgehen
  Bsp: Pacman vs. Blackjack
- **Einzelagent/Multiagent**
  → Agent muss sich ggf. **zufällig** verhalten
  Bsp: Pacman vs. Schach



Deterministisch



Stochastisch



Multiagent

Design von Agenten bei **Suchproblemen**

Heute: Umgebungseigenschaften in Standardsuchproblemen

- **Vollständig** beobachtbar → **Kein Speicher** erforderlich.
- **Diskrete** Zustände und Aktionen → Agent kann alle Zustände **aufzählen**.
- **Deterministisch** → Keine Ungewissheit.
- **Einzelagent** → Keine Zufallsstrategien erforderlich.

→ Einfachster Fall von Planung

Teil 2

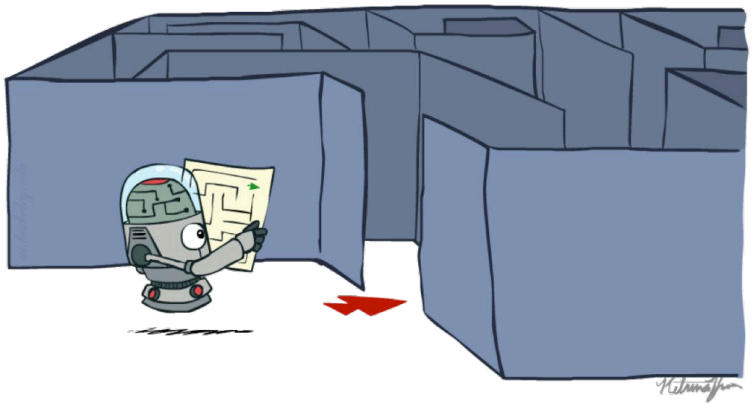
Agenten die planen und handeln
- Rationalität
- Agentendesign

Suchprobleme
- Umgebungen mit Zustandsräumen
- Zustandsräume definieren

Uninformierte Suchverfahren
- Tiefensuche („Depth-First Search“)
- Breitensuche (“Breadth-First Search“)
- Uniform-Cost Suche (“Uniform-Cost Search“)

Heuristische Suchverfahren
- A* Tree Suche

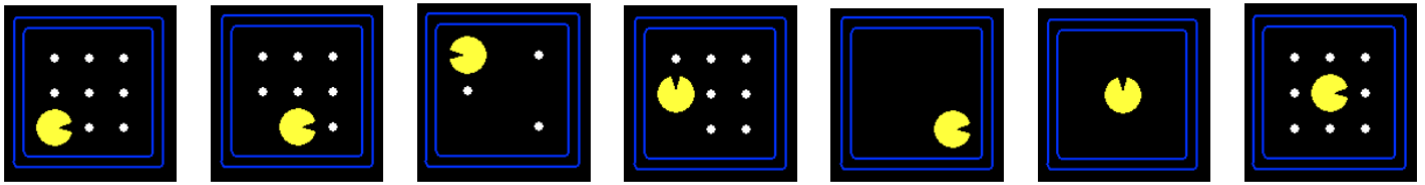
Logik- und Wissens-Repräsentationen



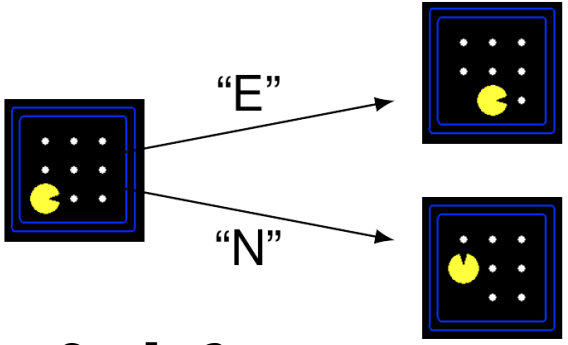
Suchprobleme

Suchprobleme bestehen aus

- Zustandsraum



- Nachfolgerfunktion (mit Aktionen, Kosten)

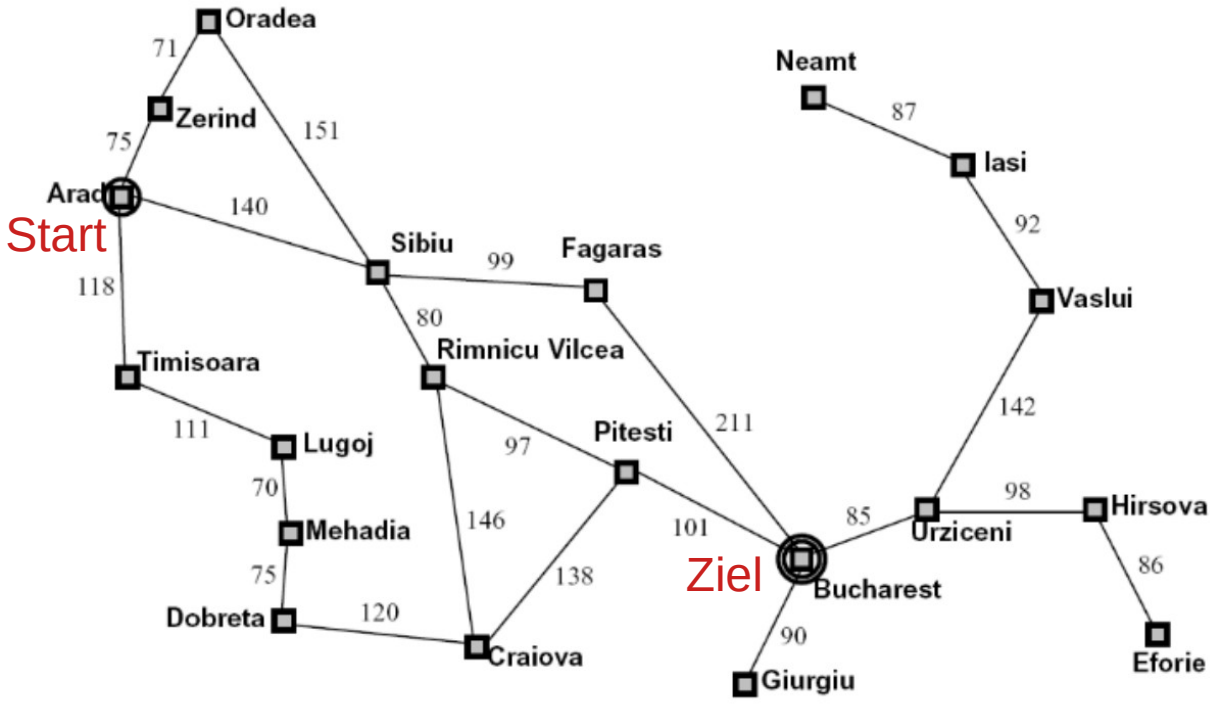


- Ein Startzustand und ein **Zieltest**: `State == Goal ?`

Lösung von Suchproblemen

- Abfolge von Aktionen (ein Plan), die den Startzustand zu einem Zielzustand (Zieltest ist positiv) überführt.

Beispiele: Bahnfahrt in Rumänien - Wegfindung



Zustandsraum
- Alle Städte

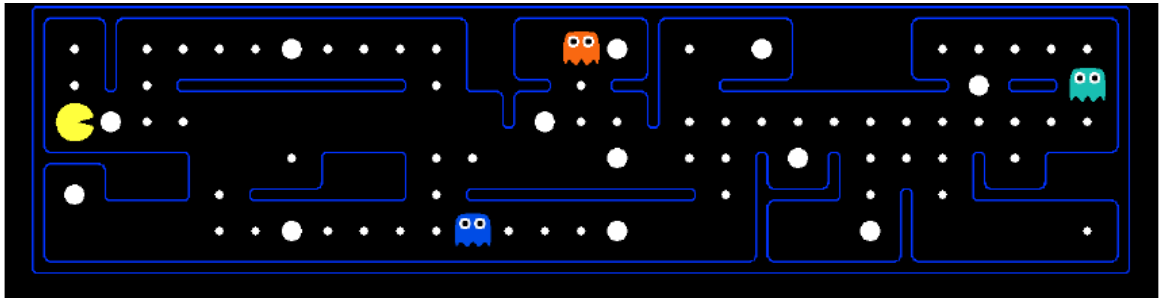
Nachfolgerfunktion
- Aktion: Gehe in benachbarte Stadt
- Kosten: Entfernung

Startzustand
- z.B. Arad

Test der Ziele
- z.B. Stadt == Bukarest?

Was befindet sich im Zustandsraum

Umgebungszustand: Jedes Detail der Umgebung



Suchzustand: Nur die für die Planung benötigten Details (Abstraktion)

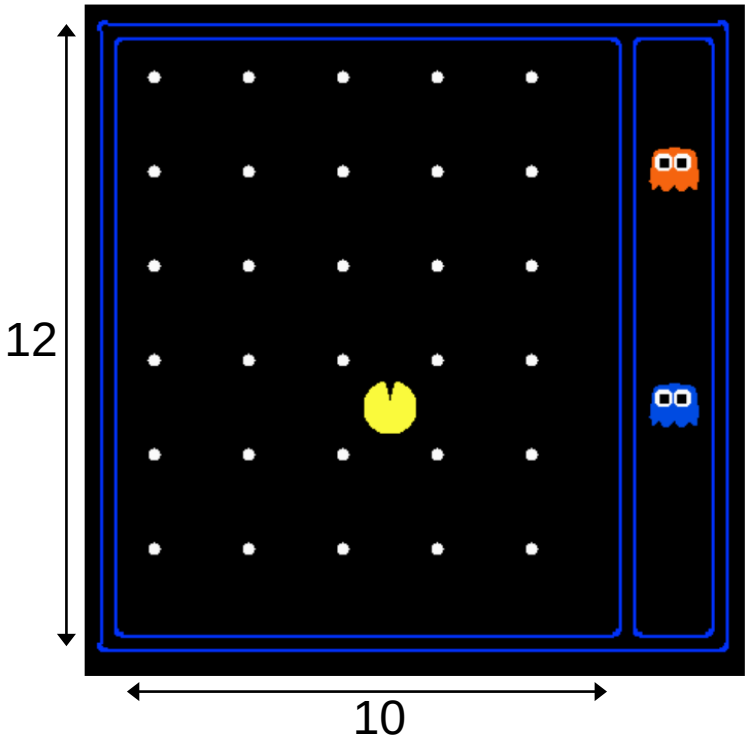
Problem: Wegfindung

- Zustände: Ort (x, y)
- Aktionen: N, S, E, W
- Nachfolgerfunktion:
  - Standort aktualisieren
  - Ggf. Kosten addieren
- Ziel der Prüfung: (x, y) == END?

Problem: „Eat-all-dots“

- Zustände: {(x, y), Booleans für jeden Dot}
- Aktionen: N, S, E, W
- Nachfolgerfunktion:
  - Standort aktualisieren
  - Ggf. Dots updaten
- Zieltest: Alle Dots auf False

Größe des Zustandsraums?

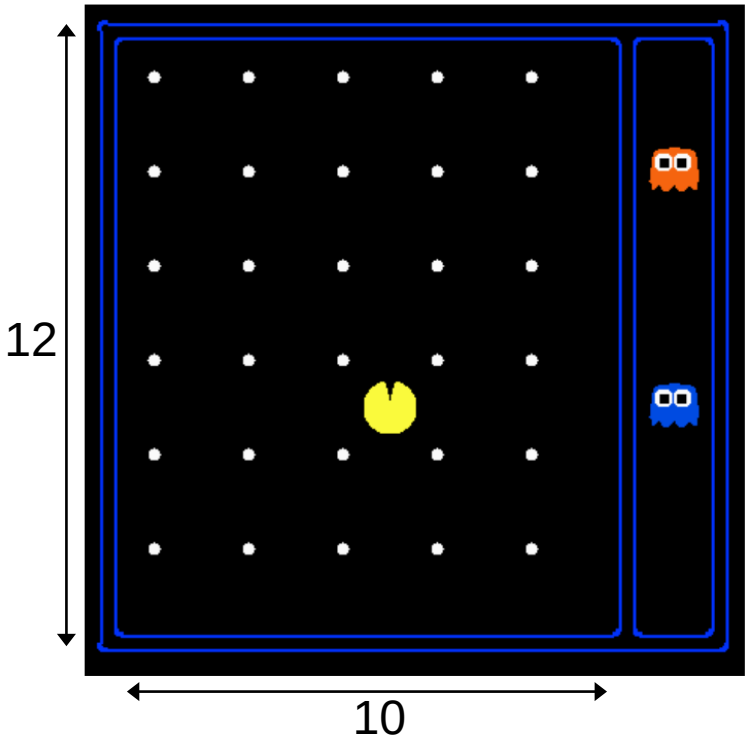


Umgebungszustände

- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

Wie viele Umgebungszustände?

Größe des Zustandsraums?



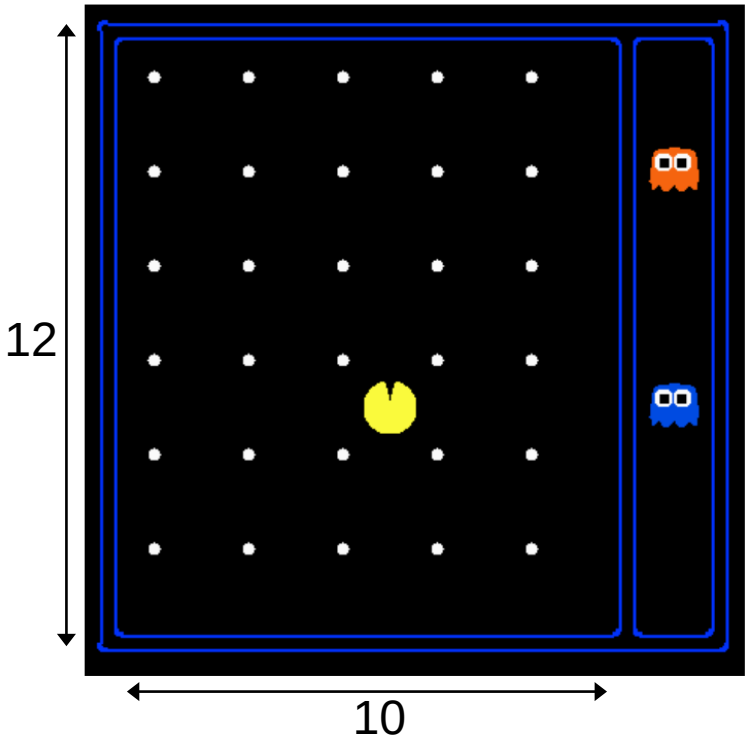
Umgebungszustände

- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

Wie viele Umgebungszustände?

- $120 \times 4 \times (2^{30}) \times (12^2)$

Größe des Zustandsraums?



Umgebungszustände

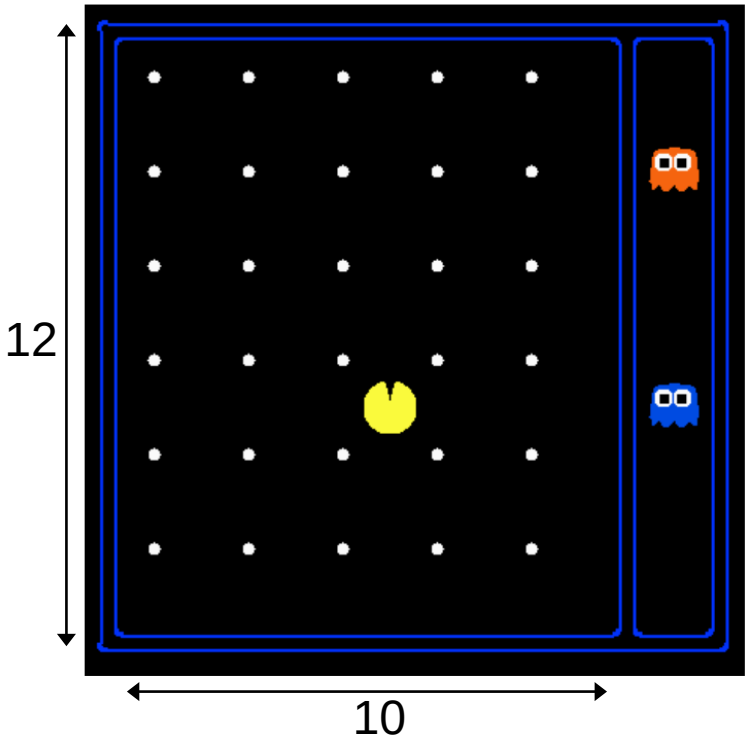
- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

Wie viele Umgebungszustände?

- $120 \times 4 \times (2^{30}) \times (12^2)$

Zustände für die Wegfindung?

Größe des Zustandsraums?

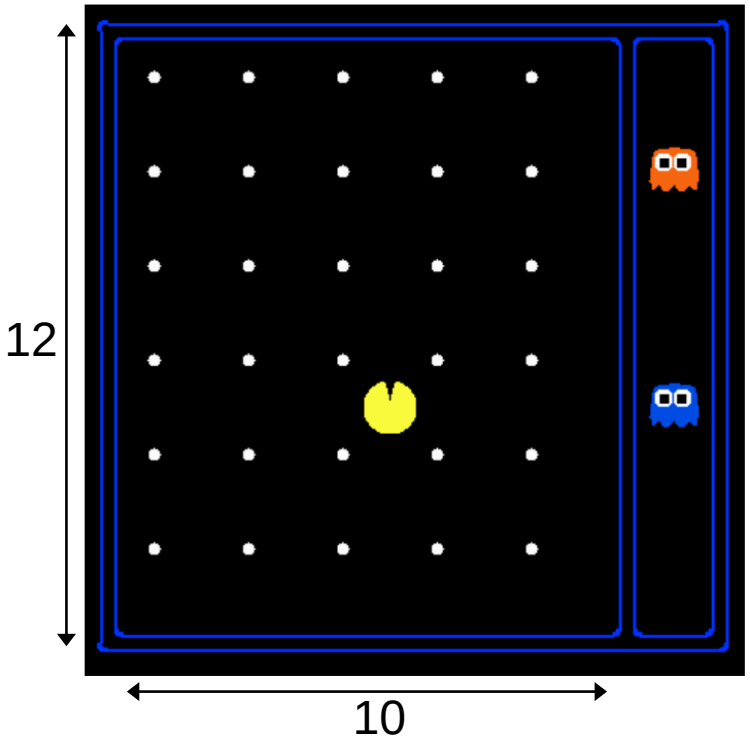


Umgebungszustände
- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

Wie viele Umgebungszustände?
- $120 \times 4 \times (2^{30}) \times (12^2)$

Zustände für die Wegfindung?
- 120

Größe des Zustandsraums?



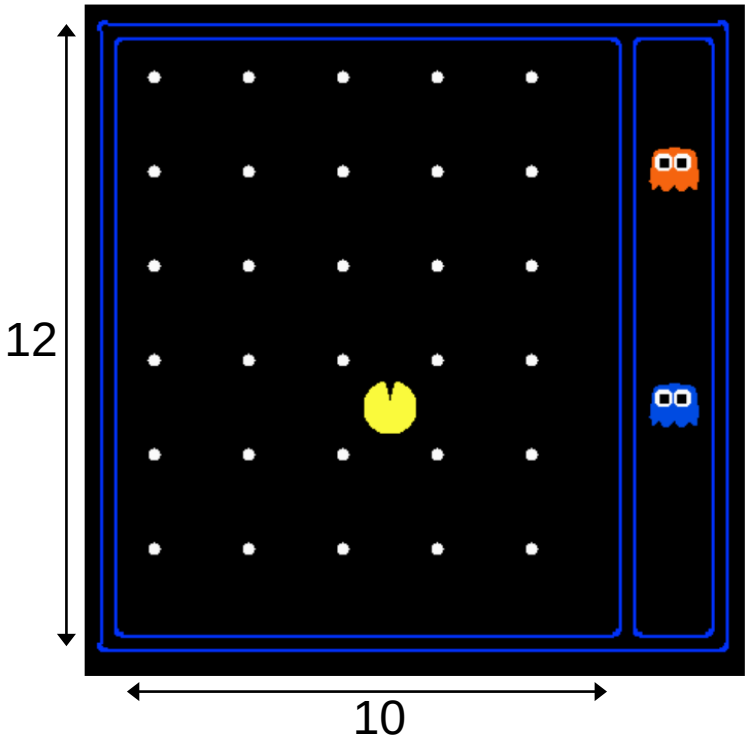
Umgebungszustände
- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

Wie viele Umgebungszustände?
- $120 \times 4 \times (2^{30}) \times (12^2)$

Zustände für die Wegfindung?
- 120

Zustände für "Eat-all-dots"?

Größe des Zustandsraums?



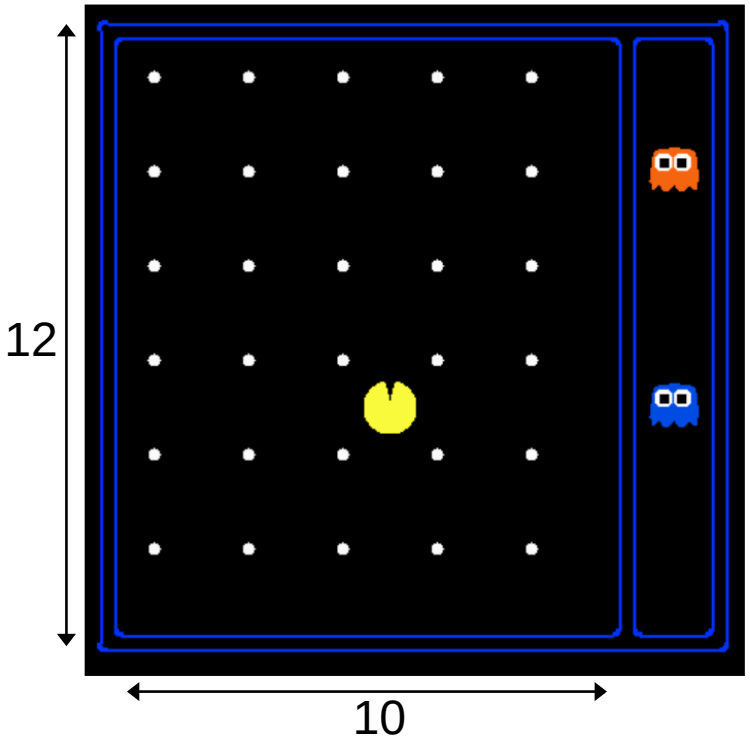
Umgebungszustände
- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

Wie viele Umgebungszustände?
- $120 \times 4 \times (2^{30}) \times (12^2)$

Zustände für die Wegfindung?
- 120

Zustände für "Eat-all-dots"?
- $120 \times (2^{30})$

Größe des Zustandsraums?



Umgebungszustände
- Agentenpositionen: 120
- Agentenrichtungen: N, S, E, W
- Anzahl der Dots: 30
- Geisterpositionen: 12

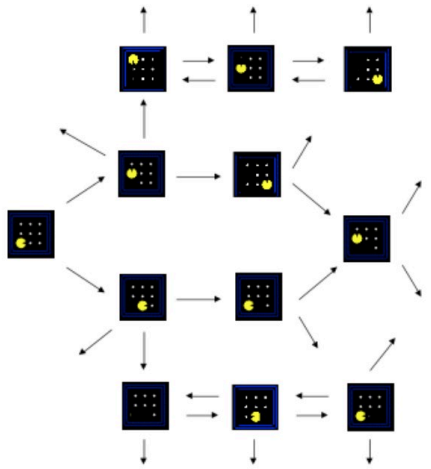
Wie viele Umgebungszustände?
- $120 \times 4 \times (2^{30}) \times (12^2)$

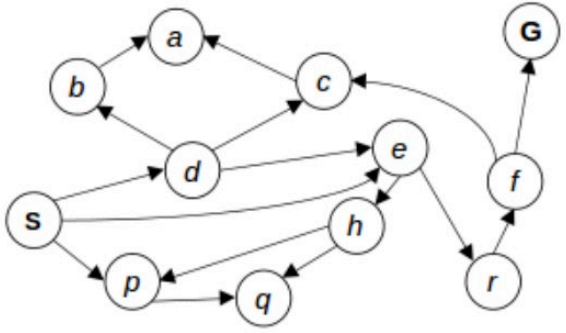
Zustände für die Wegfindung?
- 120

Zustände für "Eat-all-dots"?
- $120 \times (2^{30})$

Fluch der Dimensionalität („Curse of Dimensionality“)
- Exponentiell großer Zustandsraums
- Zu viele Zustände für einen Computer!

Zustandsraumdarstellung als **Graph** („State space graph“)



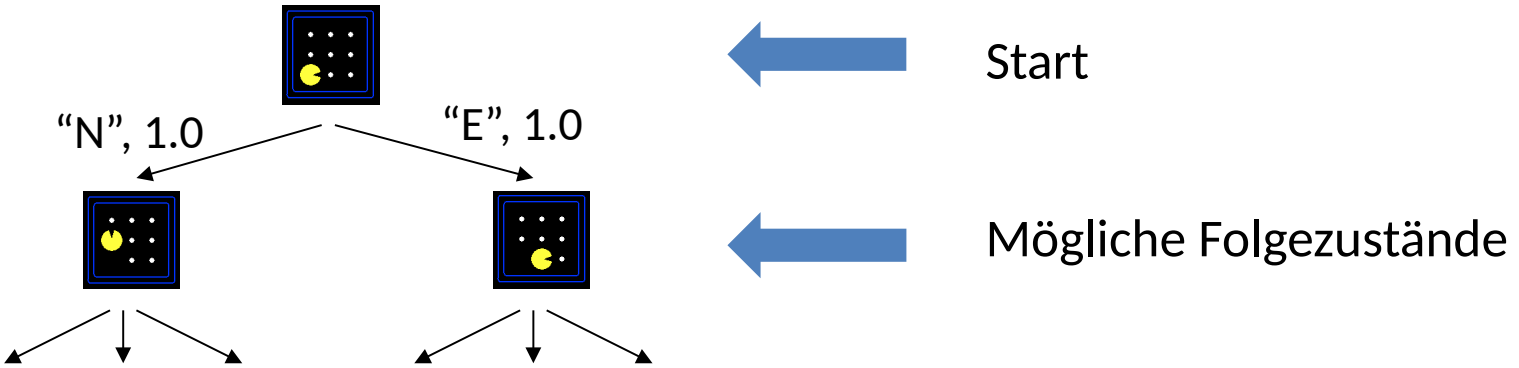


State space graph: Mathematische Darstellung eines Suchproblems

- Knotenpunkte sind (abstrahierte) Umgebungszustände
- Die Kanten stellen Nachfolgerzustände dar
- Der Zieltest besteht aus einem (oder mehreren) Zielknoten

→ **Selten vollständig darstellbar (da zu groß)**
→ Trotzdem nützliche Idee/Vorstellung

Zustandsraumdarstellung als **Suchbaum**

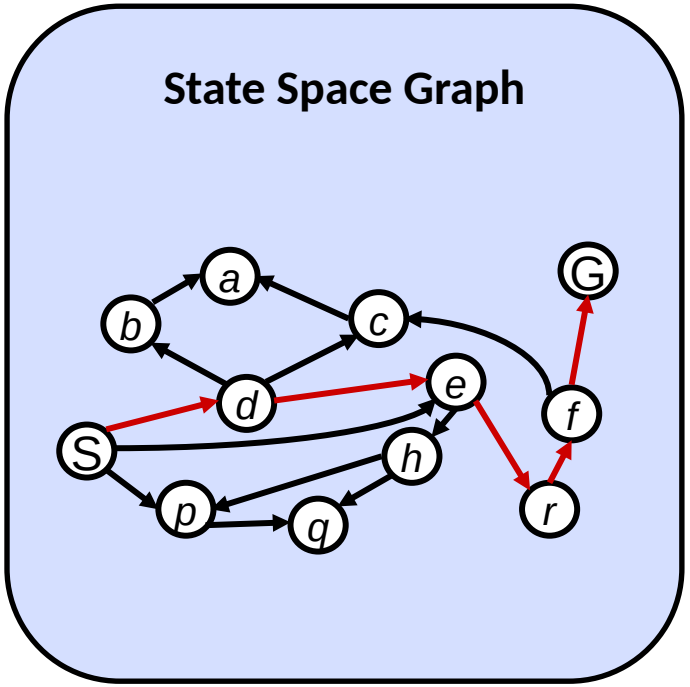


Ein Suchbaum

- Ein "Was-wäre-wenn"-Baum von Pfaden
- Der Wurzelknoten ist der Startzustand
- Nachfolger entsprechen den möglichen Folgezuständen
- Wichtig: Knoten sehen aus wie Zustände, entsprechen aber in Wirklichkeit Pfaden, um diese Zustände zu erreichen

  → Mehrere identische Knoten im Suchbaum möglich: Diese entsprechen unterschiedlichen Pfaden um diesen Zustand zu erreichen.

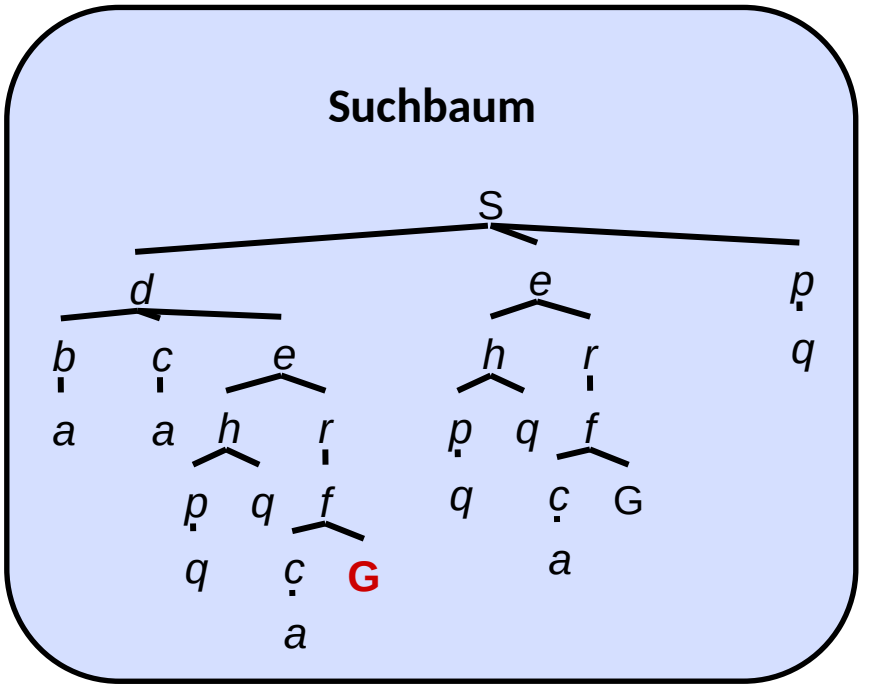
State Space Graphs vs. Suchbäume



Jede Node im Suchbaum ist ein ganzer Pfad im State Space Graph.

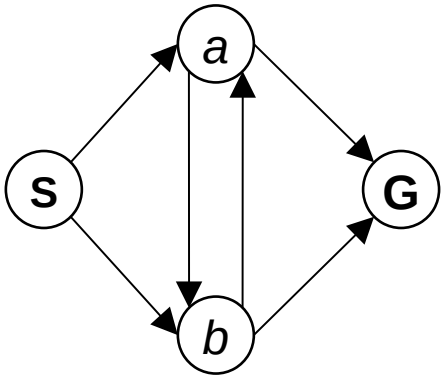
Wir konstruieren beides

- bei Bedarf
- so wenig wie möglich

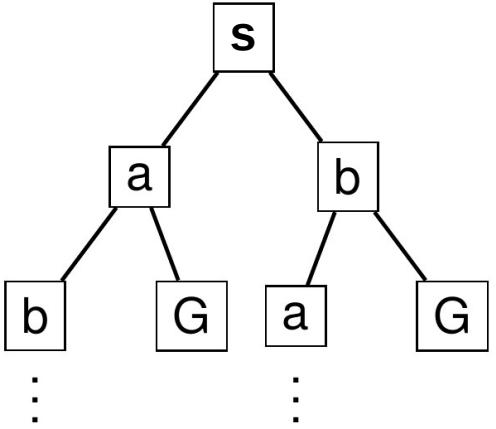


Quiz: State Space Graphs vs. Suchbaum

Betrachten Sie diesen State Space Graph:



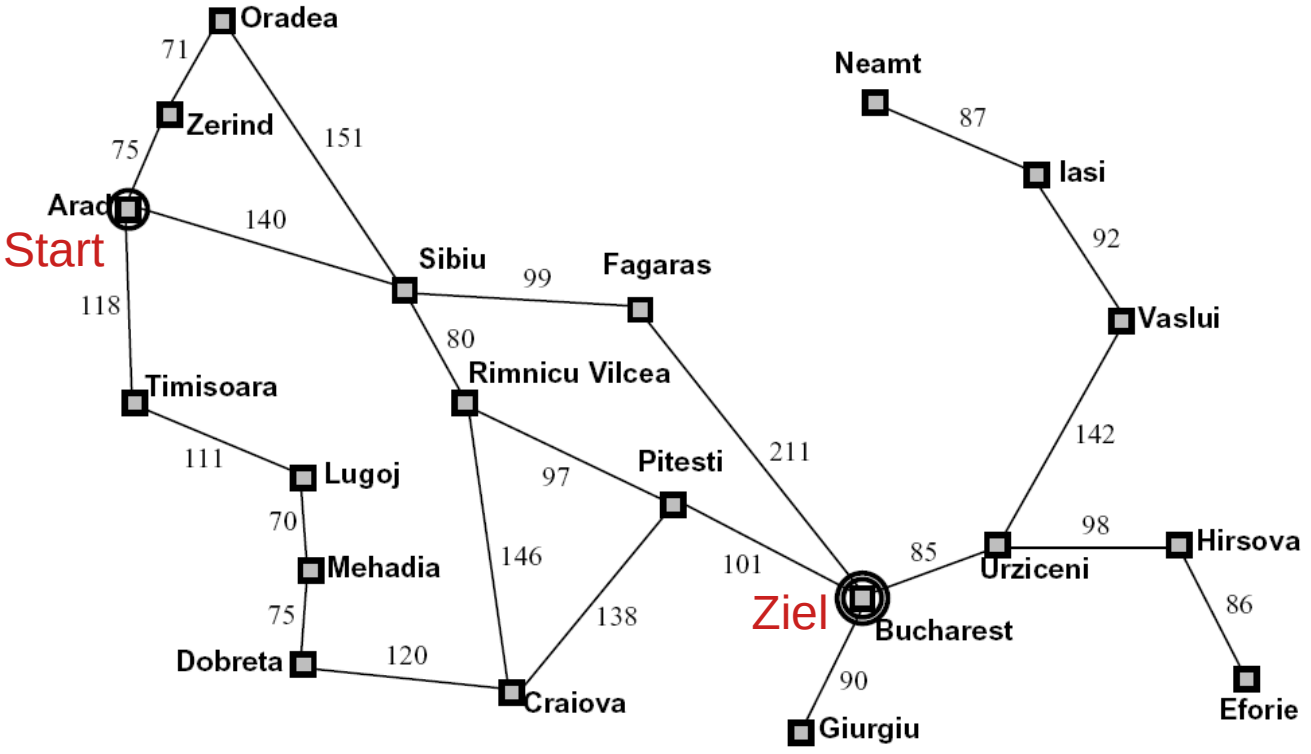
Wie groß ist der zugehörige Suchbaum?



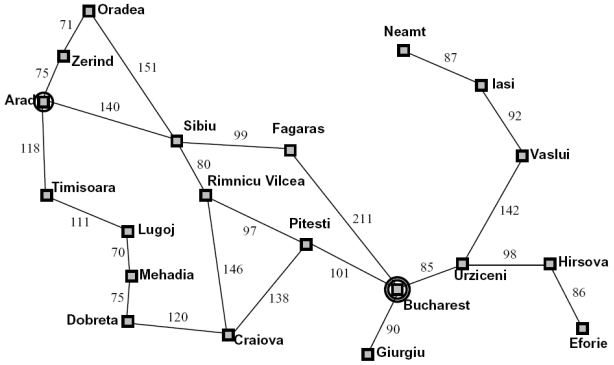
... Unendlich!

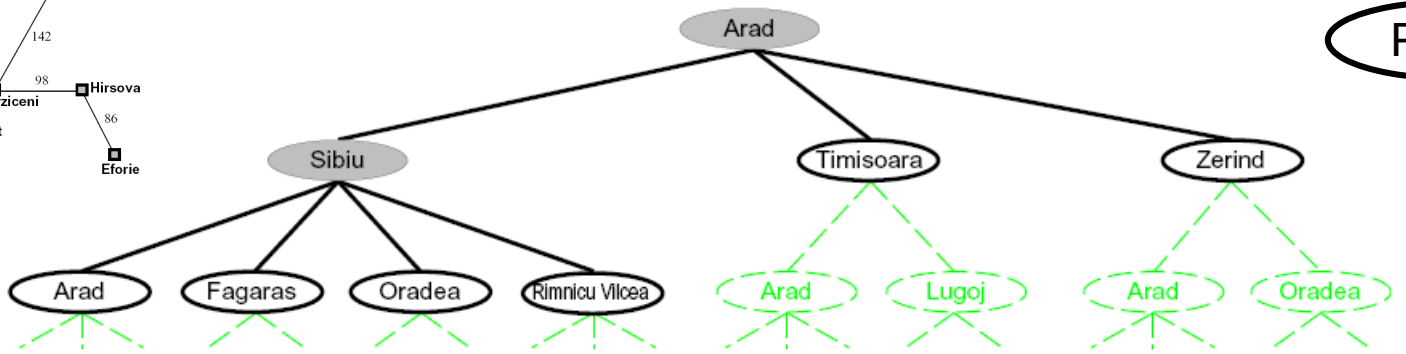
Wichtig! Viele sich wiederholende Strukturen im Suchbaum!

Beispiel: Zugfahrt in Rumänien



Algorithmus: Suche mit einem Suchbaum





Suche

- Expandiere all möglichen Knoten (also Pfade)
- Führe **eine Prioritätenliste („Fringe“)** von Teilpfaden, die in Erwägung gezogen werden können
- Aufgabe: So wenige Knoten des Fringe wie möglich zu erweitern, um das Ziel zu finden

Allgemeine Baumsuche

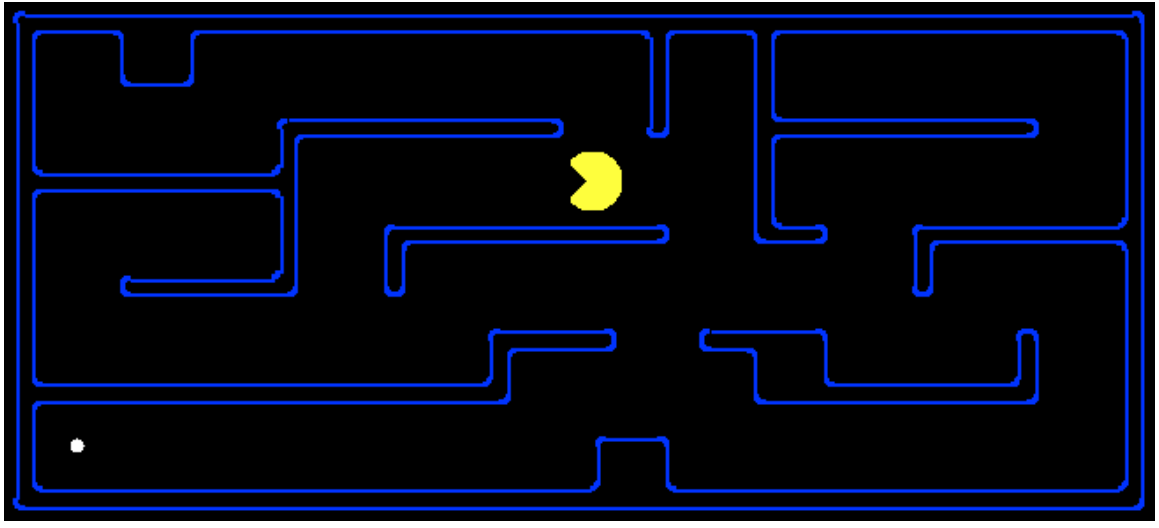
function TREE-SEARCH(*problem*, *strategy*) **returns** a **solution**, or **failure**
    initialize fringe using the initial state of the *problem*
    loop do
        if there are no candidates in fringe **then return failure**
        choose a fringe node for expansion according to *strategy* (remove node from fringe list)
        if the node is a goal state **then return** corresponding **solution**
        else expand the node: add all possible successor nodes to fringe list
    end

Wichtige Komponenten
- Fringe
- Erweiterung
- Erkundungsstrategie

Hauptfrage: Strategie - Welche Fringe-Knoten sollen erforscht werden?

Anmerkung: Eventuell andere Formulierung als in anderen Vorlesungen. Hier: Allgemeine Sichtweise.

Wie finde ich das Ziel?
Was ist der kürzeste Weg?



Kennt ihr Strategien/Algorithmen?

Teil 3

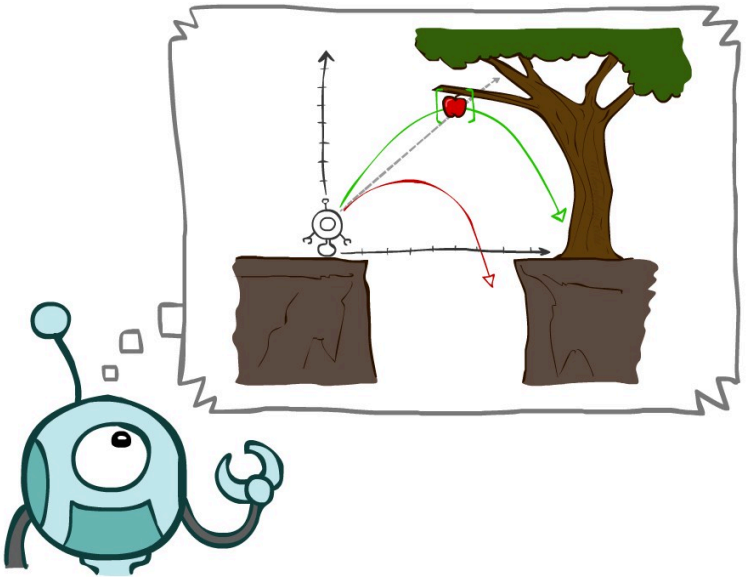
Agenten die planen und handeln
- Rationalität
- Agentendesign

Suchprobleme
- Umgebungen mit Zustandsräumen
- Zustandsräume definieren

Uninformierte Suchverfahren
- Tiefensuche („Depth-First Search“)
- Breitensuche (“Breadth-First Search“)
- Uniform-Cost Suche (“Uniform-Cost Search“)

Heuristische Suchverfahren
- A* Tree Suche

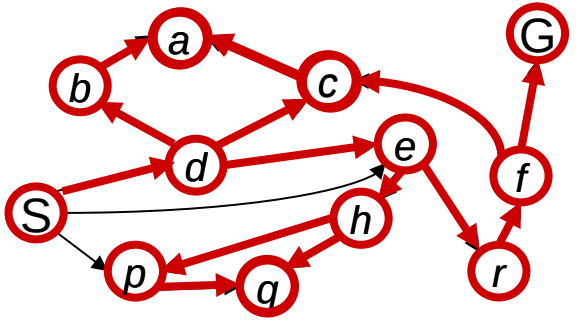
Logik- und Wissens-Repräsentationen



Depth-First Suche (Tiefensuche)

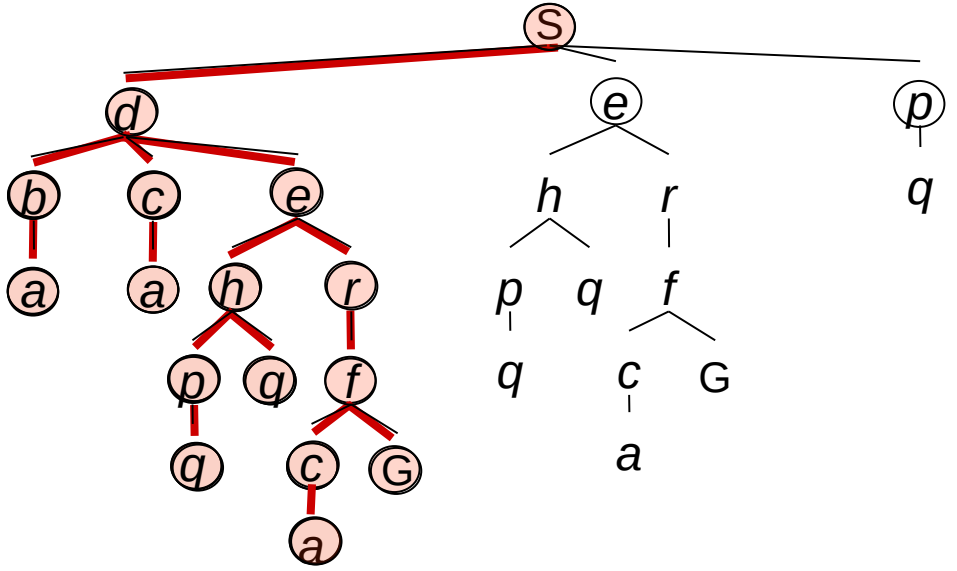
Strategie

- Zuerst den tiefsten und neusten Knoten erweitern



Umsetzung

- Fringe ist ein LIFO-Stack (Last-In First-Out)

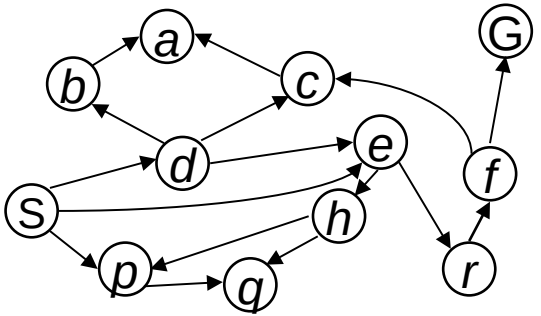


Anmerkung

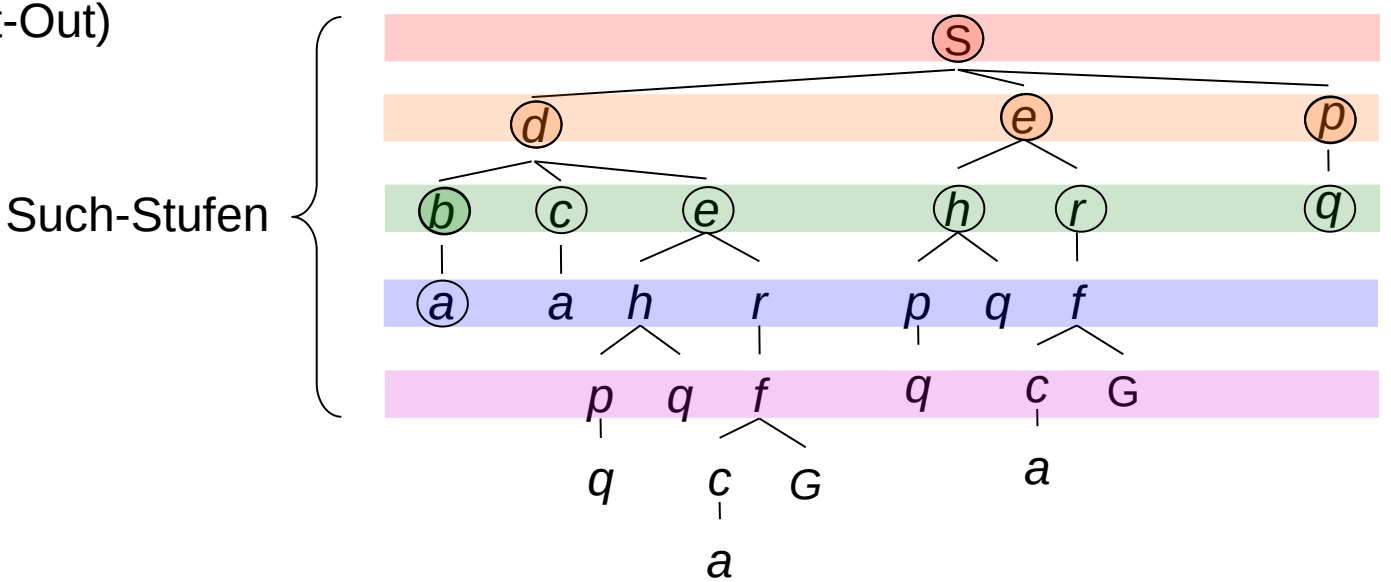
- Knoten in umgekehrter alphabetischer Reihenfolge zum Fringe hinzufügen:
- → "p", "e", "d" in der ersten Runde
- LIFO-Stapel: Expandiere dann zunächst "d"

Breadth-First Suche (Breitensuche)

Strategie
- Zuerst den flachsten Knoten erweitern



Umsetzung
- Fringe ist ein FIFO-Stack (First-In-First-Out)

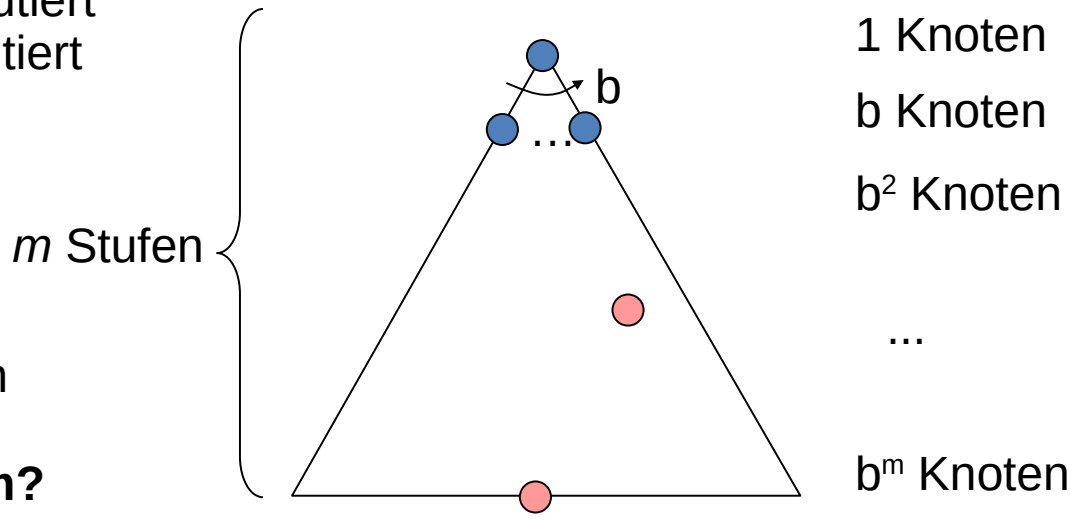


Eigenschaften von Suchalgorithmen

Vollständigkeit: Garantiert *eine* Lösung finden (wenn es eine gibt)
Optimalität: Garantiert *die beste* Lösung finden
Zeitliche Komplexität: Heute nicht diskutiert
Speicherkomplexität: Heute nicht diskutiert

Illustration des Suchbaums

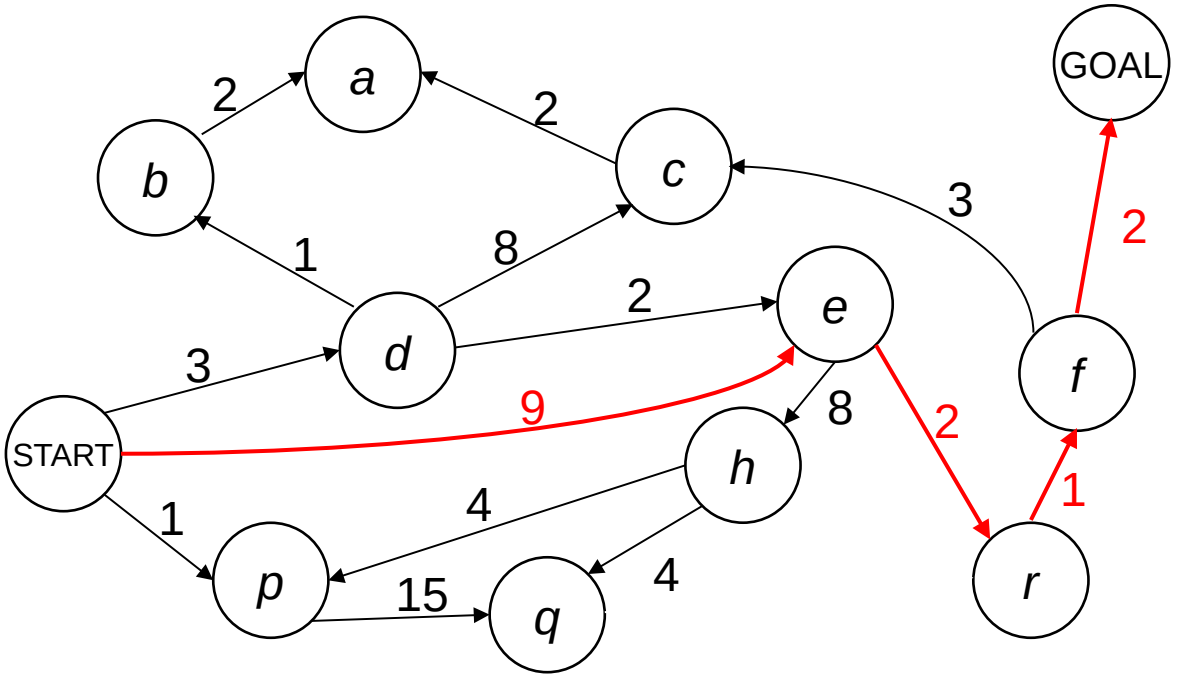
- *b* ist der Verzweigungsfaktor („branching“)
- *m* ist die maximale Tiefe
- Lösungen ● in verschiedenen Tiefen



Anzahl der Knoten im gesamten Baum?
$1 + b + b^2 + ... b^m = O(b^m)$

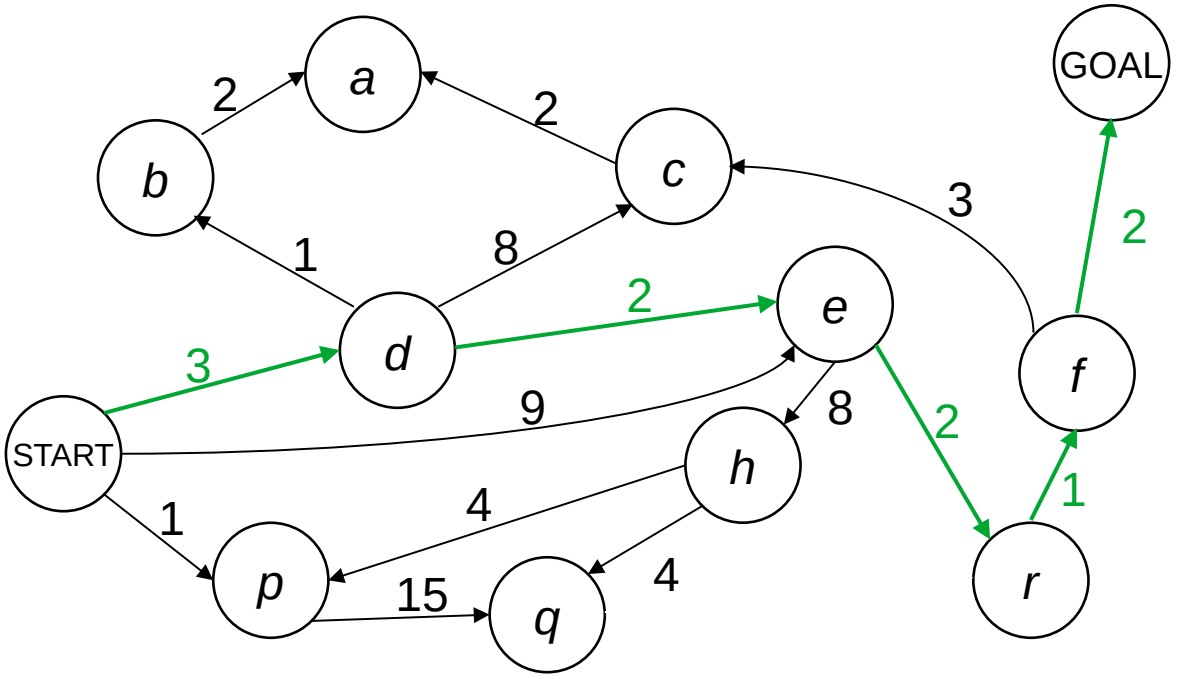
Probleme von DFS und BFS
Nicht optimal und (bei Zyklen) nicht vollständig

Kostenbewusste Suche?



BFS findet den **kürzesten** Weg (kurz = wenige Schritte)

Kostenbewusste Suche?

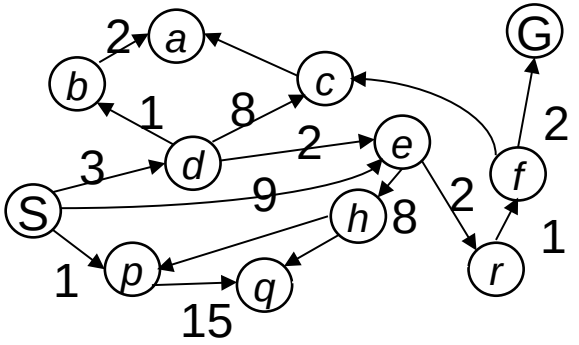


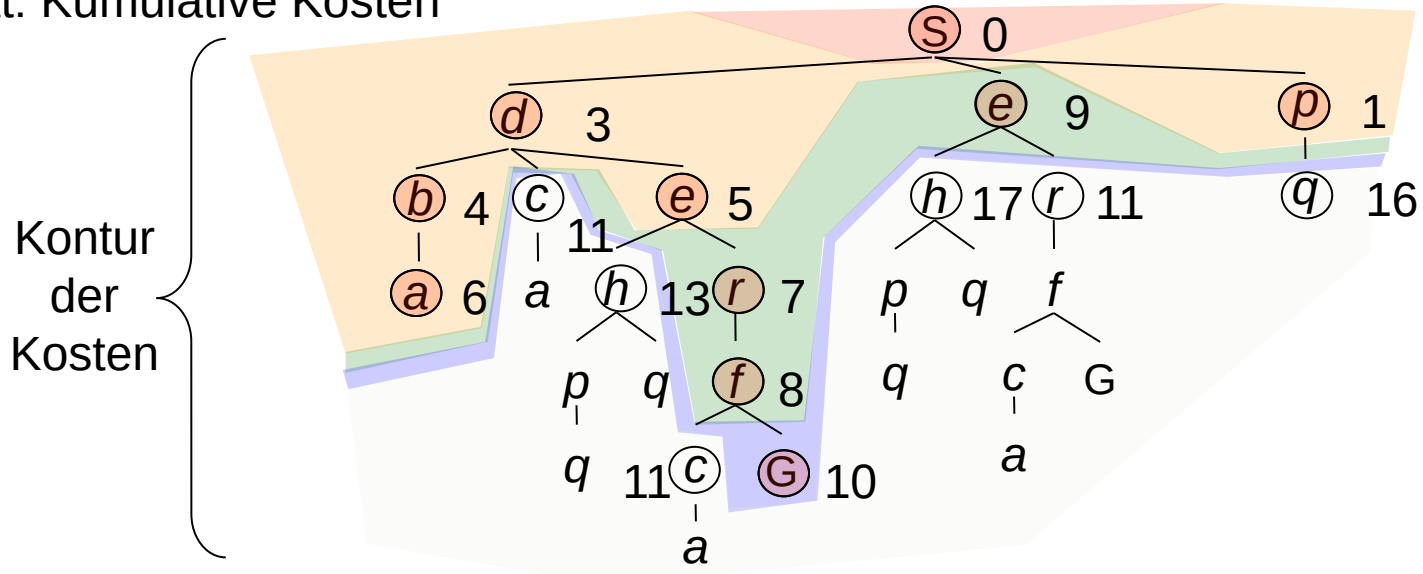
BFS findet den **kürzesten** Weg (kurz = wenige Schritte)
→ Es wird **nicht der kostengünstigste** Weg gefunden.

Uniforme Kostensuche (Uniform Cost Search, UCS)

Strategie
- Zuerst einen Knoten mit niedrigsten Kosten erweitern

Umsetzung
- Fringe ist ein Prioritäts-Stack
- Priorität: Kumulative Kosten





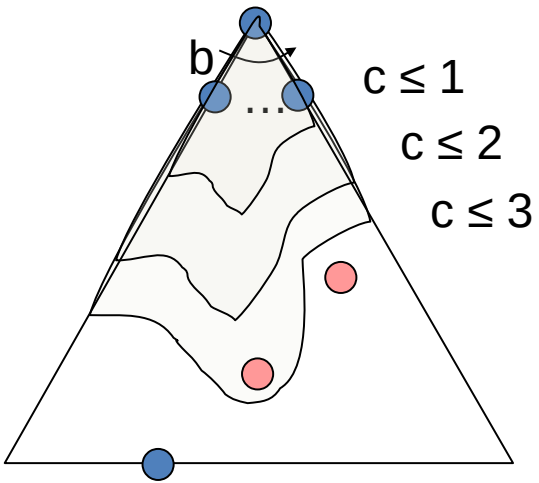
Eigenschaften der Uniform Cost Search (UCS)

Welche Knotenpunkte werden von UCS erweitert?
- Alle Knotenpunkte mit weniger Kosten als die günstigste Lösung!

Ist sie vollständig?
- Ja. Annahme: Die beste Lösung hat endliche Kosten und die minimalen Kosten pro Aktion sind positiv.

Ist sie optimal?
- Ja! (Nachweis später über A*)

Anmerkung: Ähnlich wie der Dijkstra-Algorithmus
- Findet den kürzesten Weg vom Start zu jedem anderen Knoten



Zusammenfassung bisher

Suchproblemdefinition
- Zustände (Konfigurationen der Umgebung)
- Aktionen und Kosten
- Nachfolgefunktion (Dynamik der Umgebung)
- Startzustand und Zielprüfung

Darstellung über Suchbäume
- Knotenpunkte: Pfade für das Erreichen von Zuständen
- Pfade haben Kosten (Summe der Kosten aller Aktionen entlang des Pfads)

Suchalgorithmus
- Baut systematisch einen Suchbaum auf
- Strategie: Sortierung des Fringe (also der unerforschten Knoten)
- Optimale Strategie: Finde den kostengünstigsten Pfad

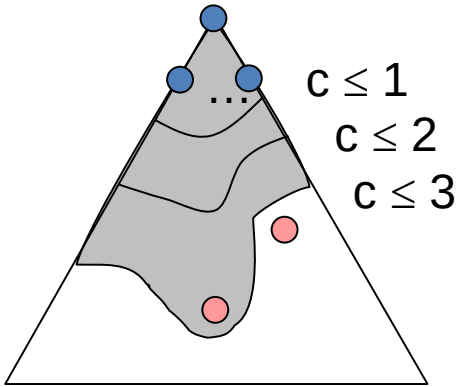
Ist Uniform Cost Search perfekt? Was könnte man verbessern?

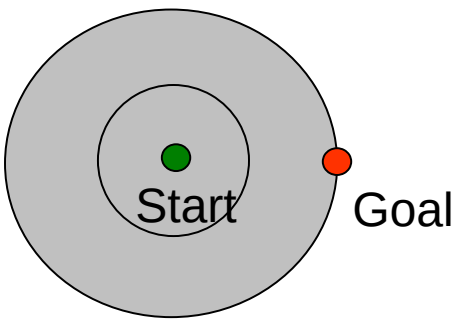
Erinnerung
- UCS erforscht steigende Kostenkonturen

Pro
- UCS ist vollständig und optimal!

Contra
UCS erkundet Optionen in jeder "Richtung"
→ Keine Informationen über den Zielort

→ **Das werden wir bald ändern!**





Teil 4

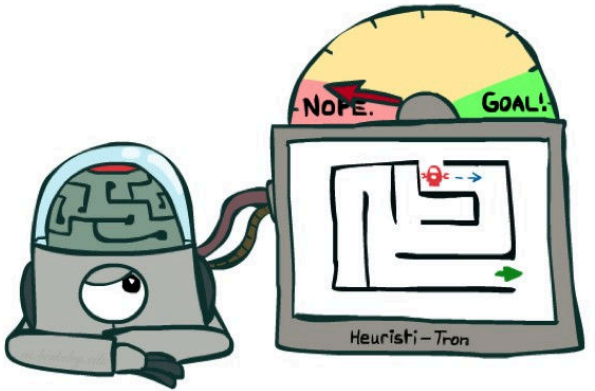
Agenten die planen und handeln
- Rationalität
- Agentendesign

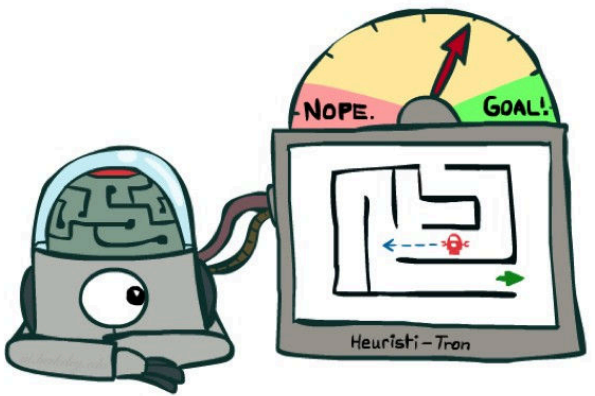
Suchprobleme
- Umgebungen mit Zustandsräumen
- Zustandsräume definieren

Uninformierte Suchverfahren
- Tiefensuche („Depth-First Search“)
- Breitensuche (“Breadth-First Search“)
- Uniform-Cost Suche (“Uniform-Cost Search“)

Heuristische Suchverfahren
- A* Tree Suche

Logik- und Wissens-Repräsentationen



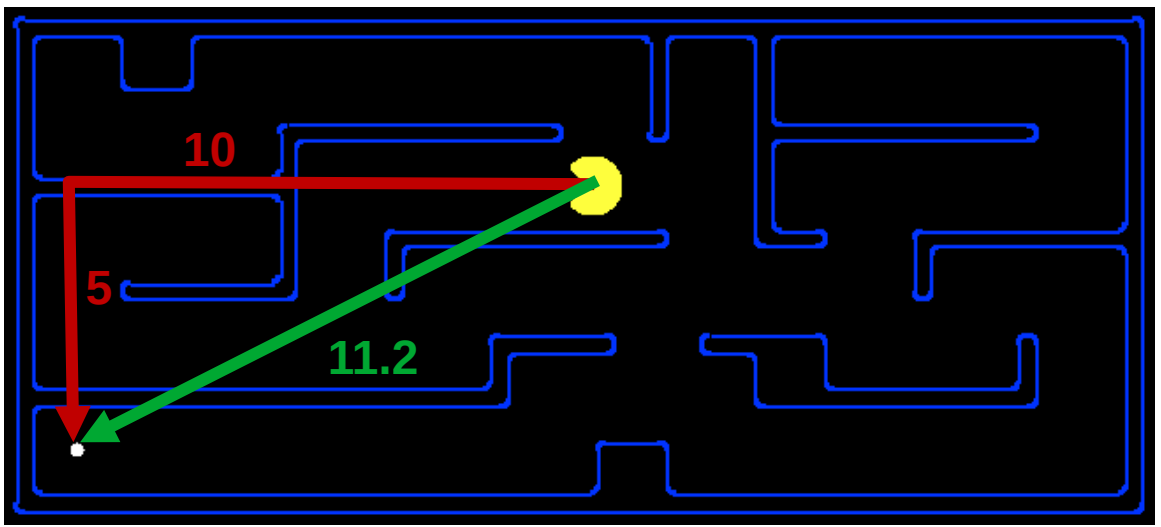


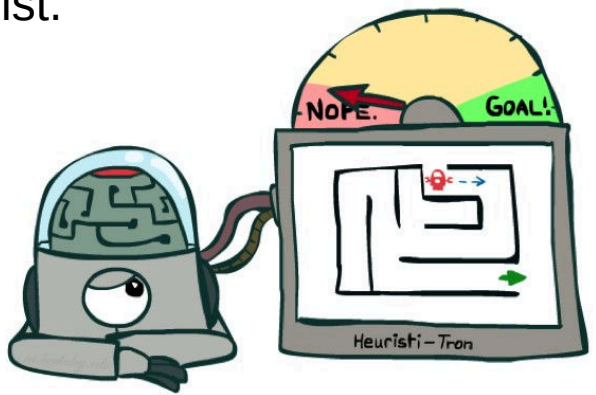
Such-Heuristiken

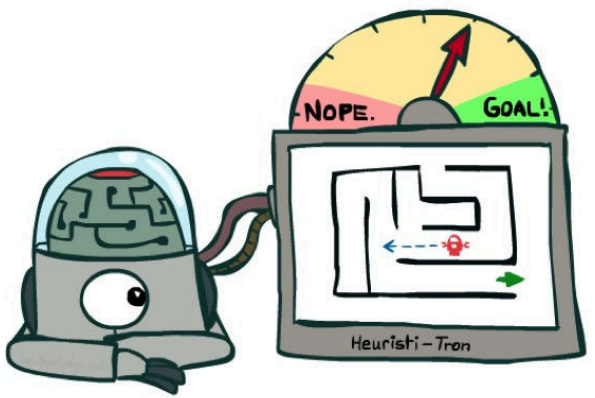
Eine Heuristik h(x) ist …

- … eine Funktion, die schätzt, wie nahe ein Zustand am Ziel ist.
- … spezifisch für ein bestimmtes Suchproblem.

- Beispiel Wegfindung:
  - Manhattan-Abstand zum Ziel
  - Euklidischer Abstand zum Ziel

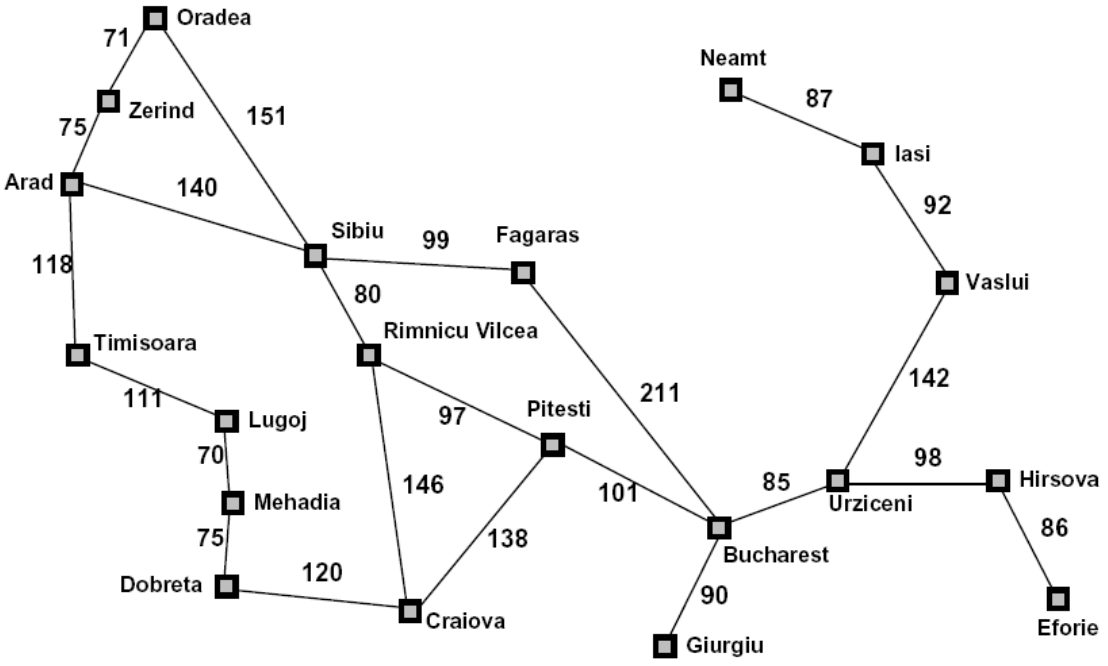






Heuristik für Bahnfahrt in Rumänien

Heuristik: Luftlinie nach Bukarest



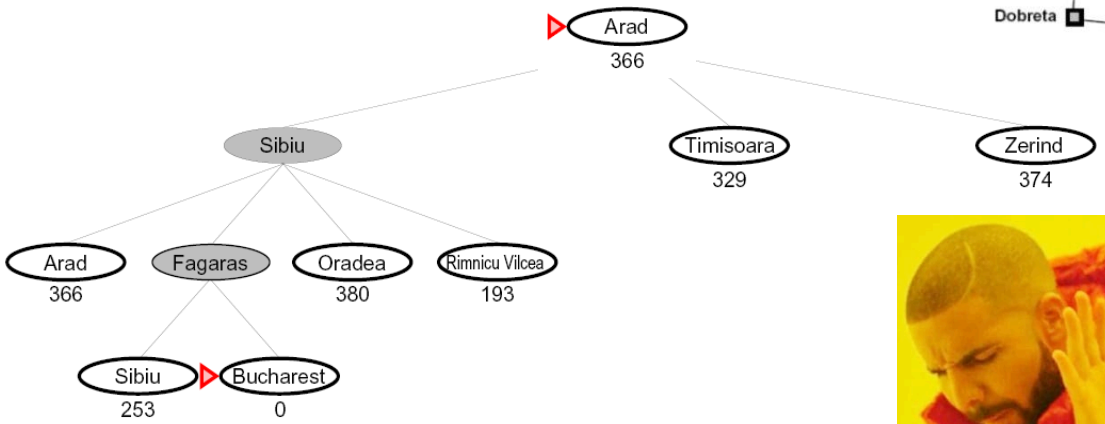
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

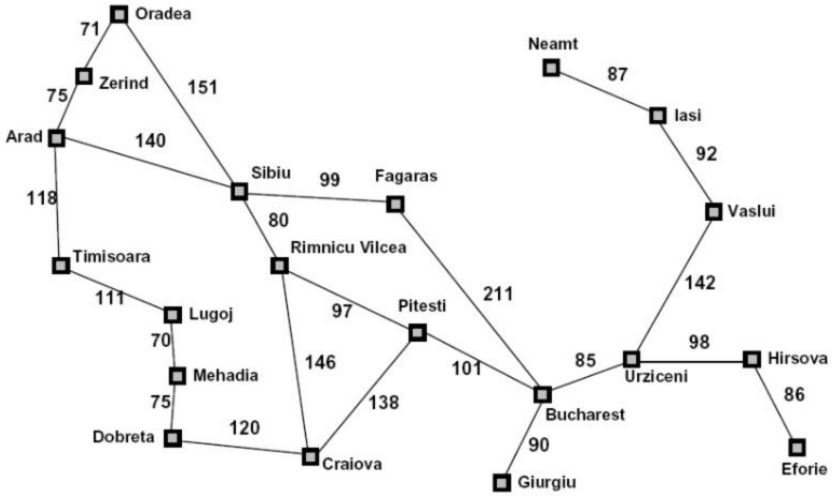
h(x)

Greedy Search („gierige Suche“)

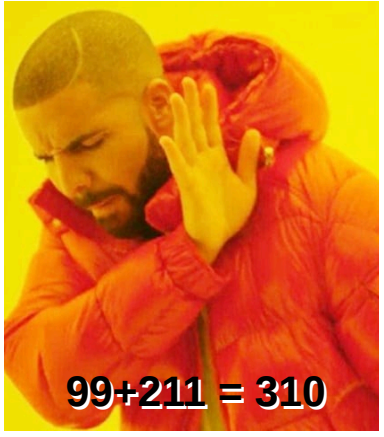
Erweitere den Knoten, der am nächsten am Ziel ist (Heuristik ist minimal)

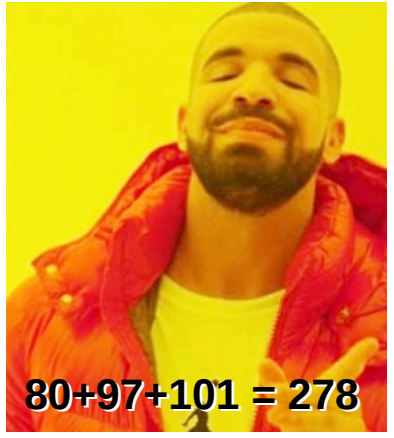
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374





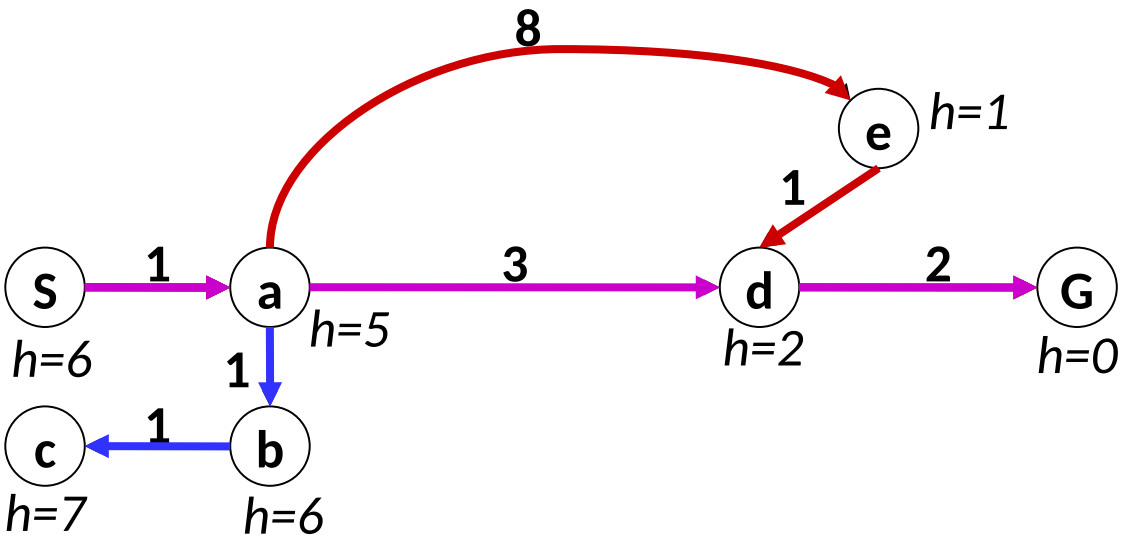
→ Was ist schief gegangen?

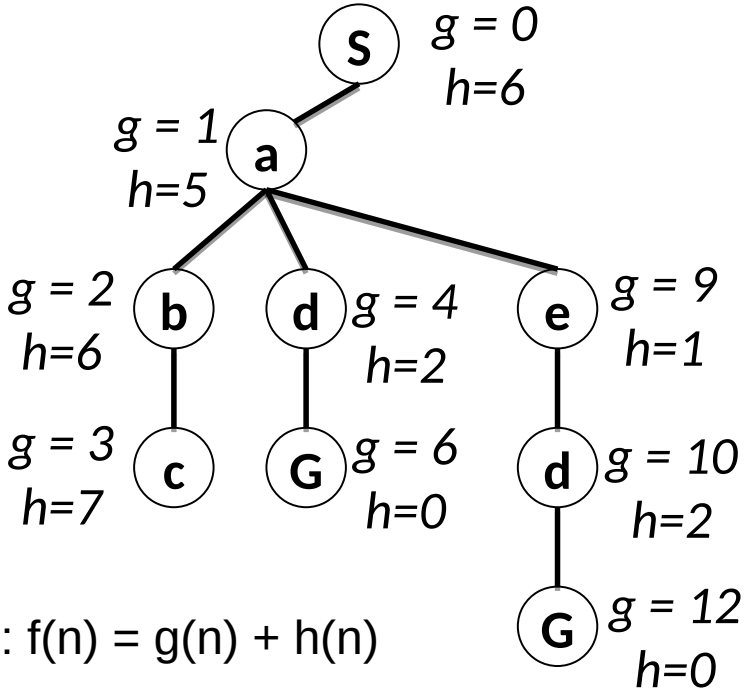




Idee: Kombiniere UCS und Heuristik/Greedy

- Uniform-Cost-Search: Sortiere nach Pfadkosten ab Start g(n) („Rückwärtskosten“)
- Greedy Search: Sortiere nach Zielnähe h(n) („Vorwärtskosten“)

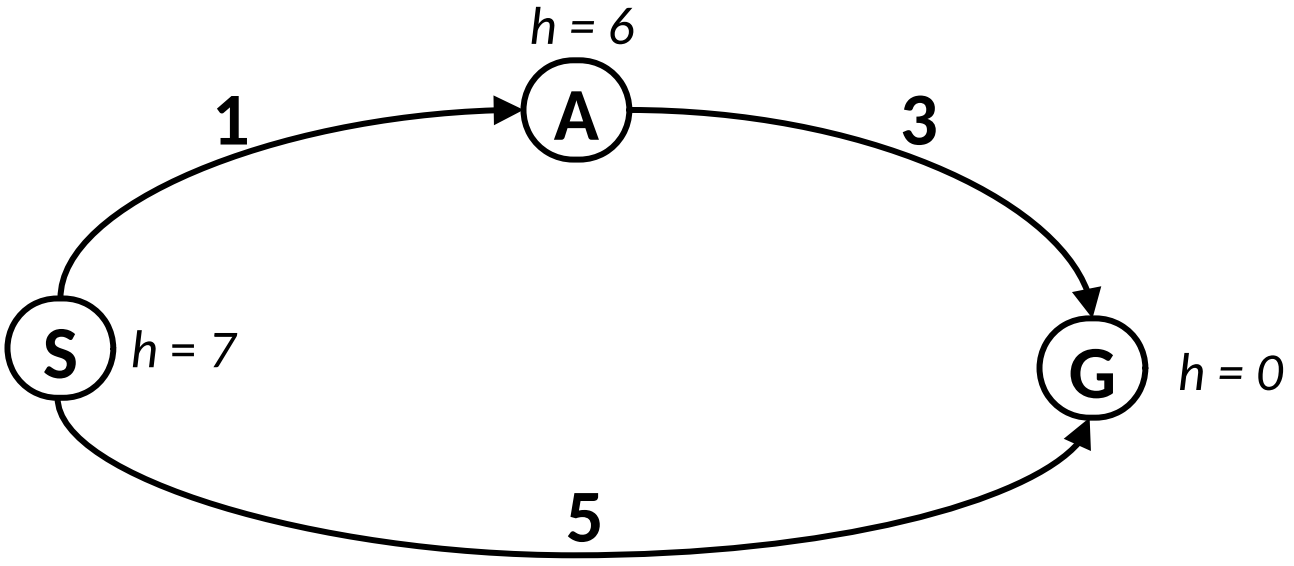




- A* Suche: Sortiere nach der Summe: f(n) = g(n) + h(n)

Ist A* optimal?

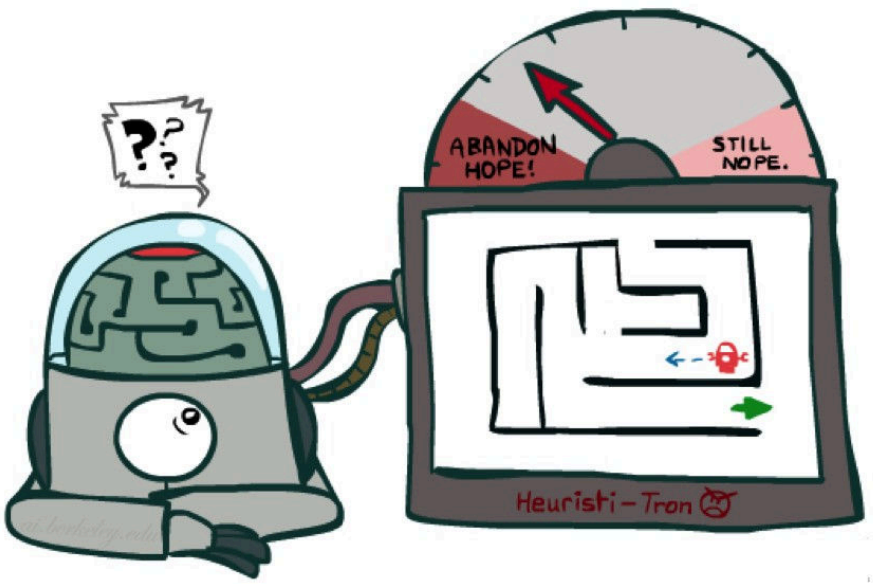
1 Minute



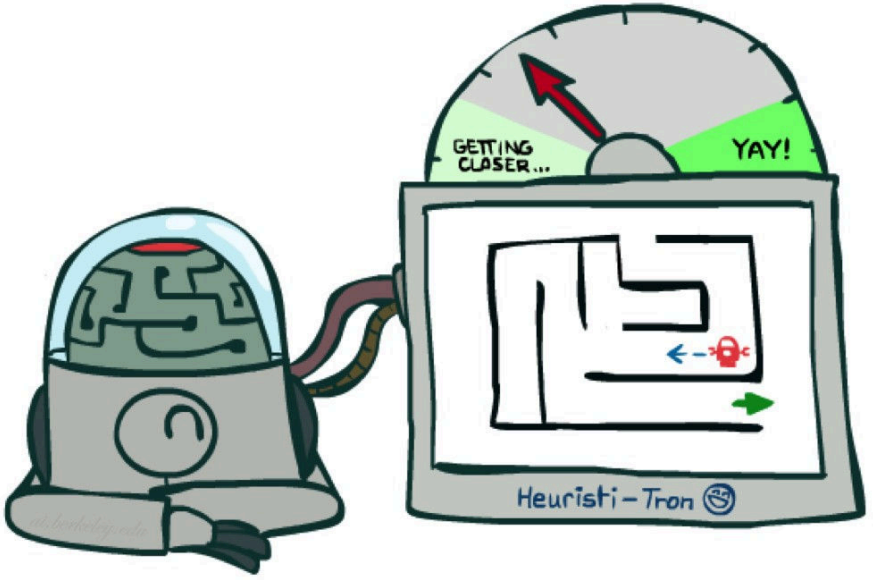
Was geht hier schief?

Geschätzte Kosten des guten Pfads > Kosten des schlechten Pfads
→ Wir brauchen Kostenabschätzungen (Heuristiken), die unter den tatsächlichen Kosten liegen!
→ Wer optimistisch ist, gewinnt!

Idee: Zulässigkeit von Heuristiken (Admissibility)



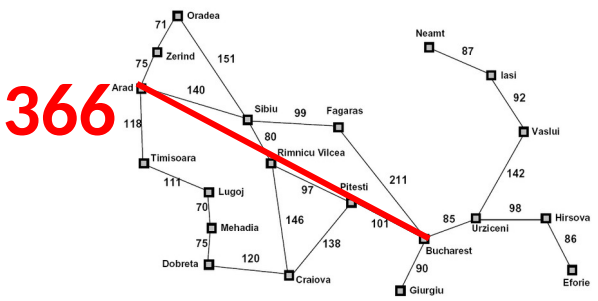
Unzulässige (pessimistische) Heuristiken
brechen die Optimalität, indem sie gute Pläne aus dem Fringe ausschließen

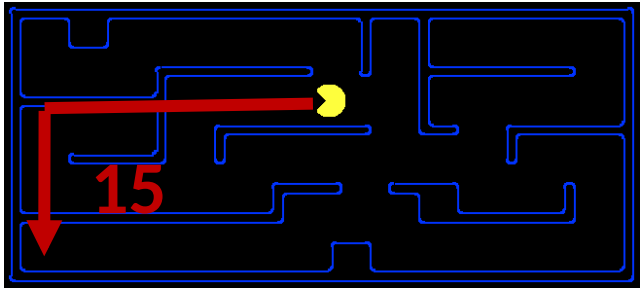


Zulässige (optimistische) Heuristiken
verhindern schlechte Pläne, unterschätzen aber die tatsächlichen Kosten

Zulässige Heuristiken

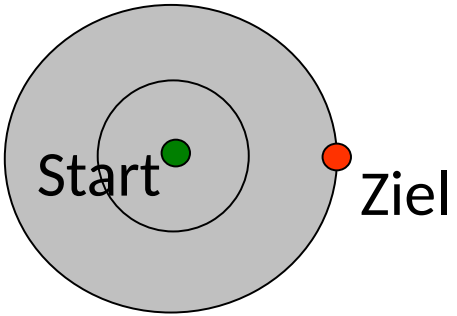
- Eine Heuristik h ist zulässig (optimistisch), wenn: $0 \leq h(n) \leq h^*(n)$
- Hier sind $h^*(n)$ die wahren Kosten zum (nächstgelegenen) Ziel.
- Die meiste Arbeit bei der optimalen Lösung schwieriger Suchprobleme besteht darin, zulässige Heuristiken zu entwickeln
- Häufig sind zulässige Heuristiken Lösungen für **einfachere Probleme**, bei denen zusätzliche Aktionen verfügbar sind (direkte Wege nehmen, durch Wände gehen, ...)



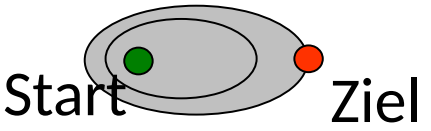


Konturen in UCS vs. A*

- Konturen gleicher Kosten dehnen sich gleichmäßig in alle Richtungen aus



- A* dehnt die Konturen in Richtung Ziel aus
- A* traut aber nicht nur der Heuristik, um Optimalität zu gewährleisten (z.B. um Sackgassen zu vermeiden)



A*: Zusammenfassung

- **A*** verwendet sowohl **Rückwärts-Kosten** als auch **(geschätzte) Vorwärts-Kosten**
- **A* ist optimal** mit **zulässiger** (optimistischer) Heuristik
- **Design der Heuristik** ist der Schlüssel

  → Häufig werden vereinfachte Probleme verwendet

A* Anwendungen

- Videospiele
- Routenplanung
- Scheduling
- Bewegungsplanung Robotik
- Maschinelle Übersetzung
- Spracherkennung
- ...

Teil 5

Agenten die planen und handeln
- Rationalität
- Agentendesign

Suchprobleme
- Umgebungen mit Zustandsräumen
- Zustandsräume definieren

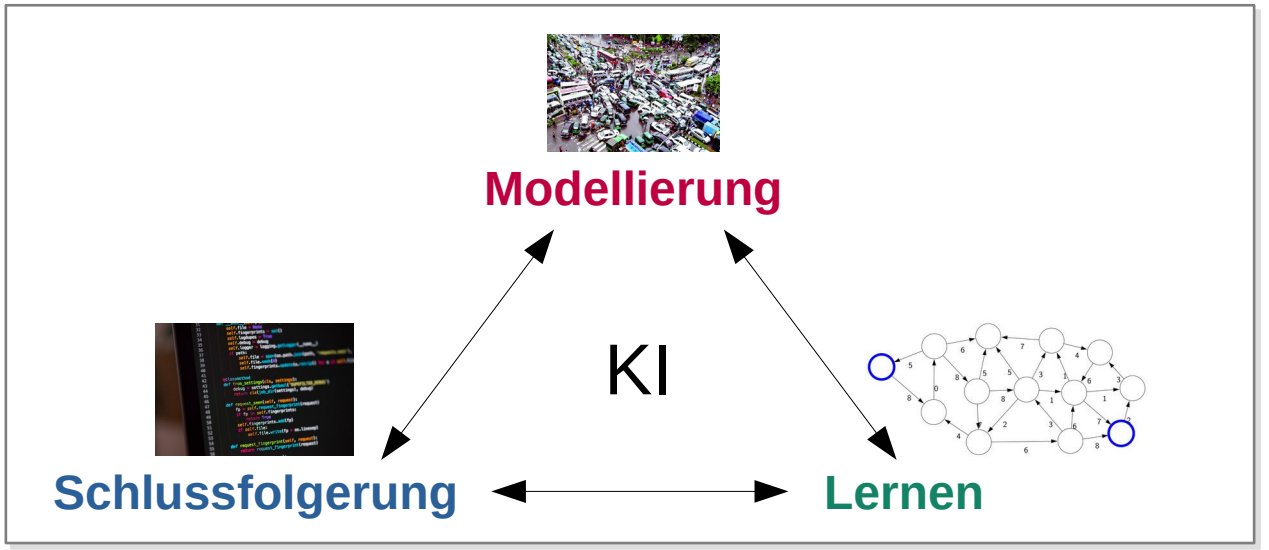
Uninformierte Suchverfahren
- Tiefensuche („Depth-First Search“)
- Breitensuche (“Breadth-First Search“)
- Uniform-Cost Suche (“Uniform-Cost Search“)

Heuristische Suchverfahren
- A* Tree Suche

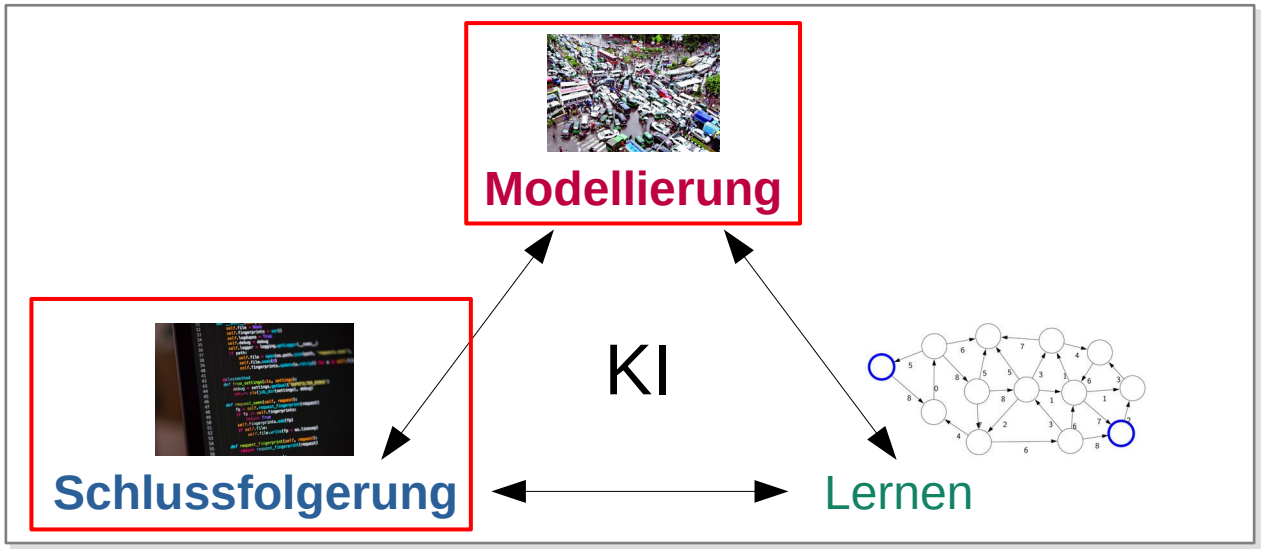
Logik- und Wissensrepräsentationen

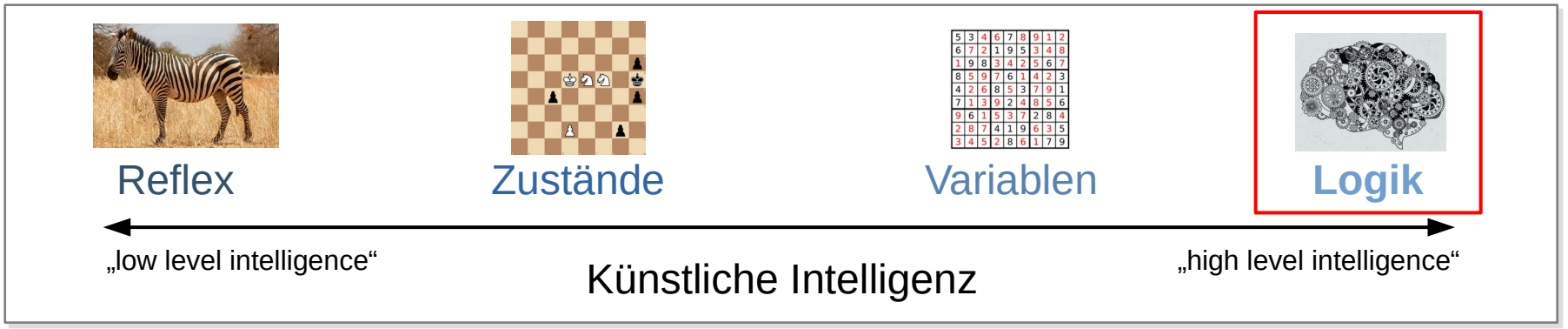
```
>> Hello! I am your personal assistant!
>> How can I help you?

> All students like GKI.
>> I learned something.

> Bob does not like GKI.
>> I learned something.

> Is Bob a student?
>> No.
```

KI-Überblick



Jetzt

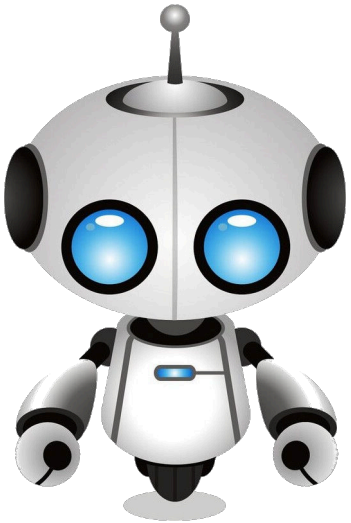

Modellierung
KI
Schlussfolgerung
Lernen


Reflex
Zustände
Variablen
Logik
„low level intelligence“
Künstliche Intelligenz
„high level intelligence“

Ziel: Smart Personal Assistant

Informationen mitteilen

```
> Es schneit.
>> Ich habe etwas gelernt.

> Schnee ist rutschig.
>> Ich habe etwas gelernt.
```



Fragen stellen

```
> Ist es rutschig?

>> Ja.
```

Voraussetzungen

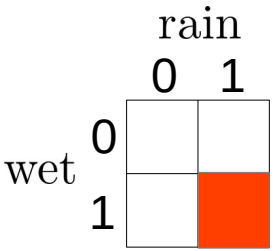
- Muss logisch schlussfolgern können.
- Interaktion: Sprache
- Logik: Erfordert formale Sprache

Zutaten der Logik

- **Syntax**: Definiert ein Set gültiger Ausdrücke/Formeln
  - Beispiel $\text{rain} \wedge \text{wet}$

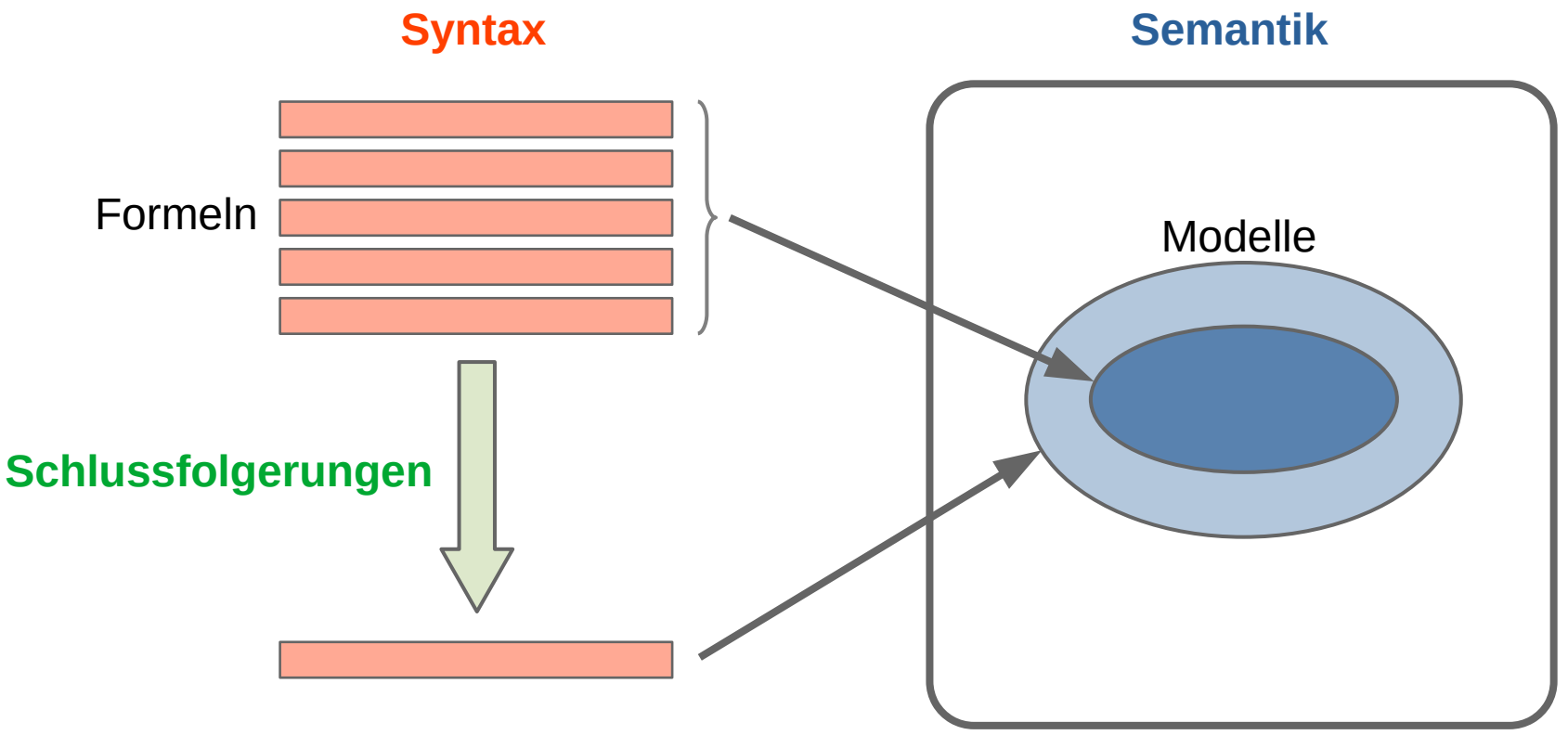
→ **Nur Symbole, keine Bedeutung!**

- **Semantik**: Bedeutung (Zuordnung von Werten → „Interpretation“/„Modell“)
  - Beispiel



- **Logisches Schlussfolgern**: Schlussregeln, gegeben Ausdruck *f*, welches *g* können wir ableiten?
  - Beispiel: $\dfrac{\text{rain} \wedge \text{wet}}{\text{wet}}$

Aussagenlogik



Interpretationsfunktion

Definition: Interpretationsfunktion

Gegeben: Formel f und Interpretation w

Die **Interpretationsfunktion** $\mathcal{I}(f, w)$ ergibt:

- Wahr: w erfüllt f („w satisfies f“)
- Falsch: w erfüllt f nicht („w does not satisfy f“)

f	g	$\mathcal{I}(\neg f, w)$	$\mathcal{I}(f \wedge g, w)$	$\mathcal{I}(f \vee g, w)$	$\mathcal{I}(f \rightarrow g, w)$	$\mathcal{I}(f \leftrightarrow g, w)$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

Gegeben
Interpretationen von f und g

Definition
Die Bedeutung der Junktoren

Knowledge Base

Definition: Knowledge Base

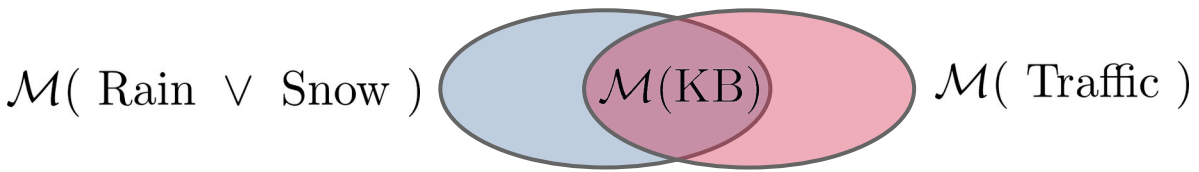
Syntax: Eine Knowledge Base KB ist ein Set von Formeln $\{f_i, \ldots, f_n\}$.

Semantik: $\mathcal{M}(\mathrm{KB})$ ist das Set aller Interpretationen, die alle Formeln erfüllen.

$$\mathcal{M}(\mathrm{KB}) = \bigcap_{f \in \mathrm{KB}} \mathcal{M}(f)$$

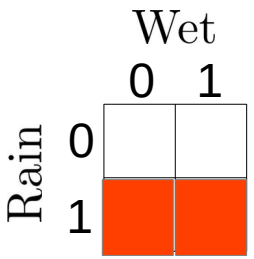
Intuition: Formeln spezifizieren Bedingungen an erlaubte Modelle.

$$\mathrm{KB} = \{\ \mathrm{Rain} \ \vee \ \mathrm{Snow}\ ,\ \mathrm{Traffic}\ \}$$

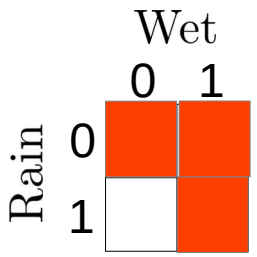


Knowledge Base: Beispiel

$\mathcal{M}(\text{Rain})$

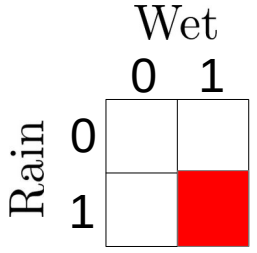


$\mathcal{M}(\text{ Rain } \rightarrow \text{ Wet })$

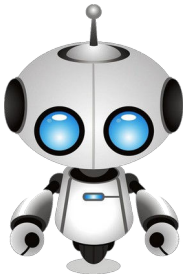


Schnitt:

$\mathcal{M}(\{\text{ Rain, Rain } \rightarrow \text{ Wet }\})$



Szenarien

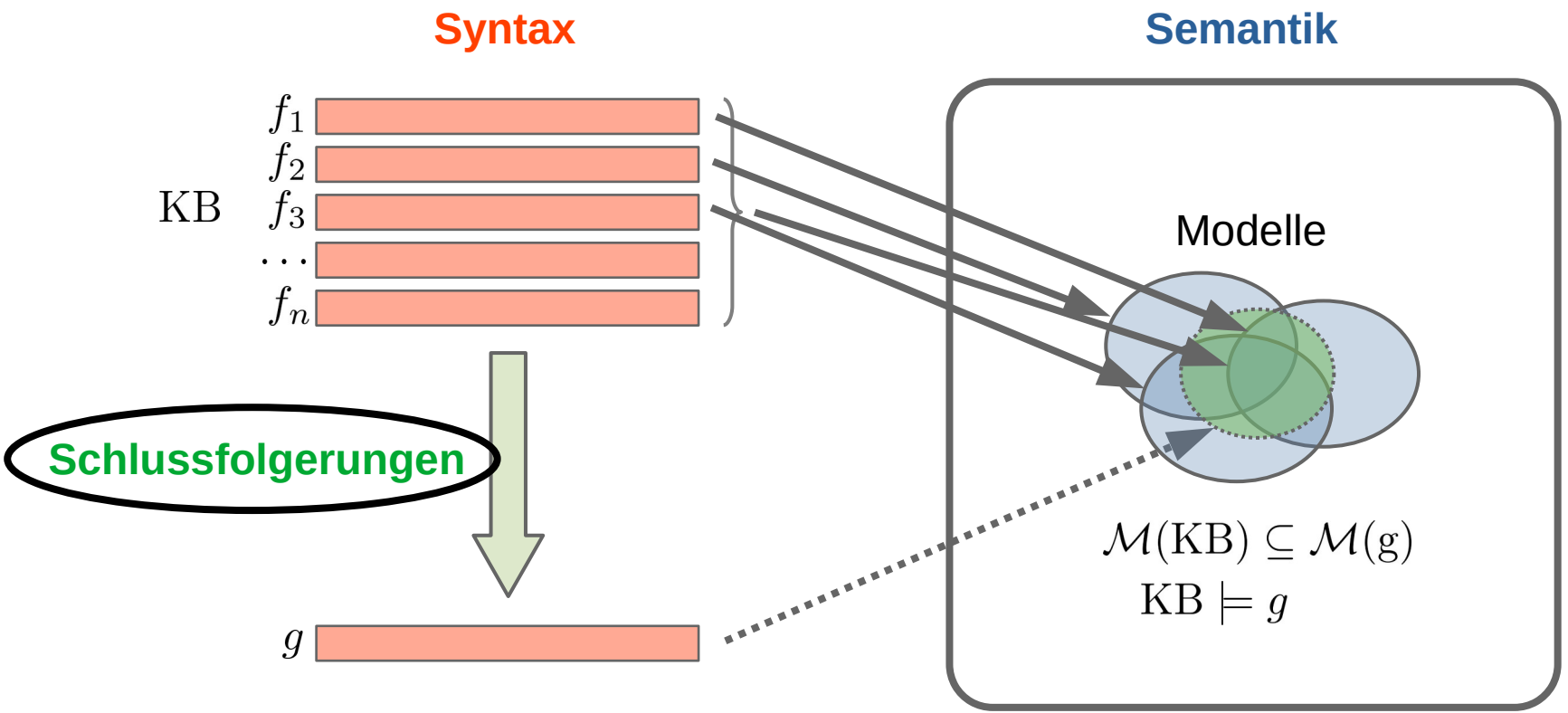
Informationen mitteilen →  ← **Fragen** stellen

$\text{Tell}\ [f]$ → KB → ?

Mögliche Reaktionen

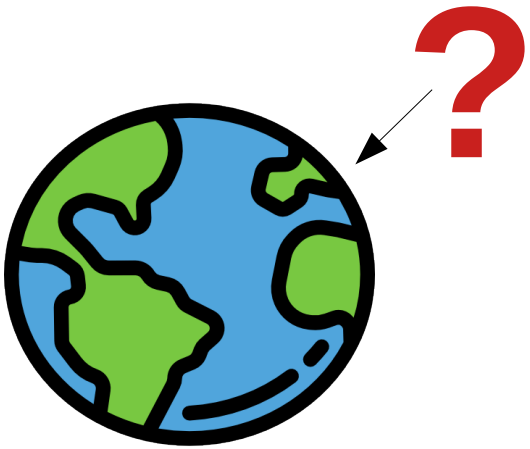
- „Das wusste ich schon.“ → Entailment/Folgerung: $\text{KB} \models f$
- „Das glaube ich nicht.“ → Widerspruch: $\text{KB} \models \neg f$
- „Ich habe etwas gelernt.“ → Contingency: Update KB

Schlussfolgerungen

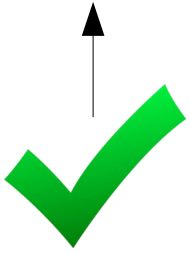


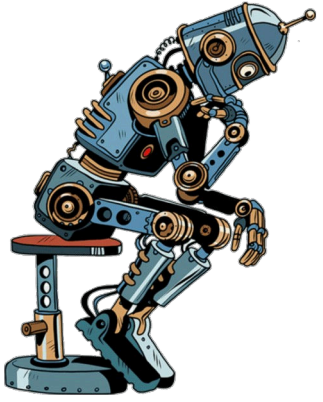
Ziele der Logik-basierten KI

- Das **Wissen** der Welt **repräsentieren**



- Mit dem Wissen **schlussfolgern**





Die folgenden Folien basieren auf der Vorlesung von **Prof. Dr. Harald Sack** (AIFB, KIT): **Information Service Engineering** https://www.youtube.com/channel/UCjkkhNSNuXrJpMYZoeSBw6Q

Formale Wissensrepräsentation

- **Formale Wissensrepräsentation …**
  - … ist ein Bereich der künstlichen Intelligenz
  - … erfasst **eindeutig** die **Semantik (Bedeutung)** von **Begriffen**, **Eigenschaften**, **Beziehungen** und **Individuen** spezifischer Wissensdomänen in Form von **strukturierten Daten**
- **Maschinen (Computer)** müssen in der Lage sein, formale Wissensrepräsentationen zu **verstehen**
- **„Verstehen**": Computer sind in der Lage, die Wissensrepräsentation **korrekt zu interpretieren**

- **Wie machen wir das? Durch eine Ontologie!**

Was ist Ontologie?

Definiton in der Informatik

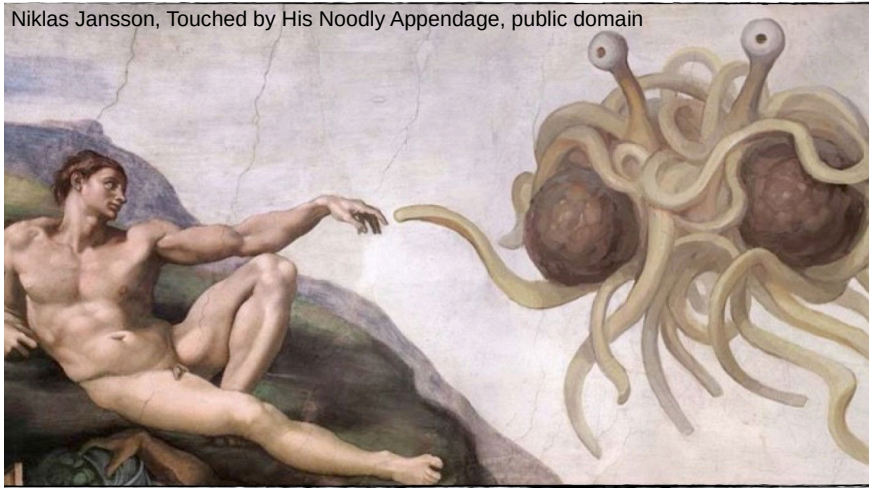
- Eine **Ontologie** ist eine **explizite**, **formale** Spezifikation einer **gemeinsamen Begriffsbildung („conceptualization“).**[1]

Begriffsbildung
- Abstraktes Modell
- Für einen Bereich relevante Begriffe, Beziehungen

Explizit
- Bedeutung aller enthaltenen Begriffe muss definiert sein

Formal
- Verständlich für Maschinen (nicht nur lesbar!)

Gemeinsam
- Konsens über die Ontologie


Niklas Jansson, Touched by His Noodly Appendage, public domain

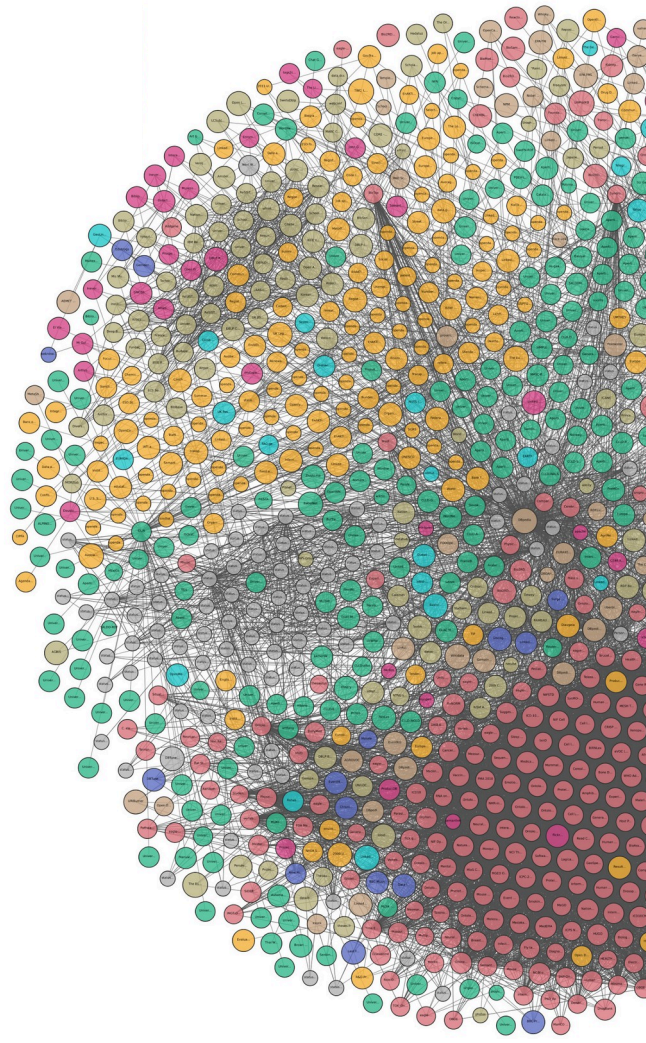
[1] Thomas R. Gruber: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2):199-220, 1993.

Wie kann man Ontologien darstellen?

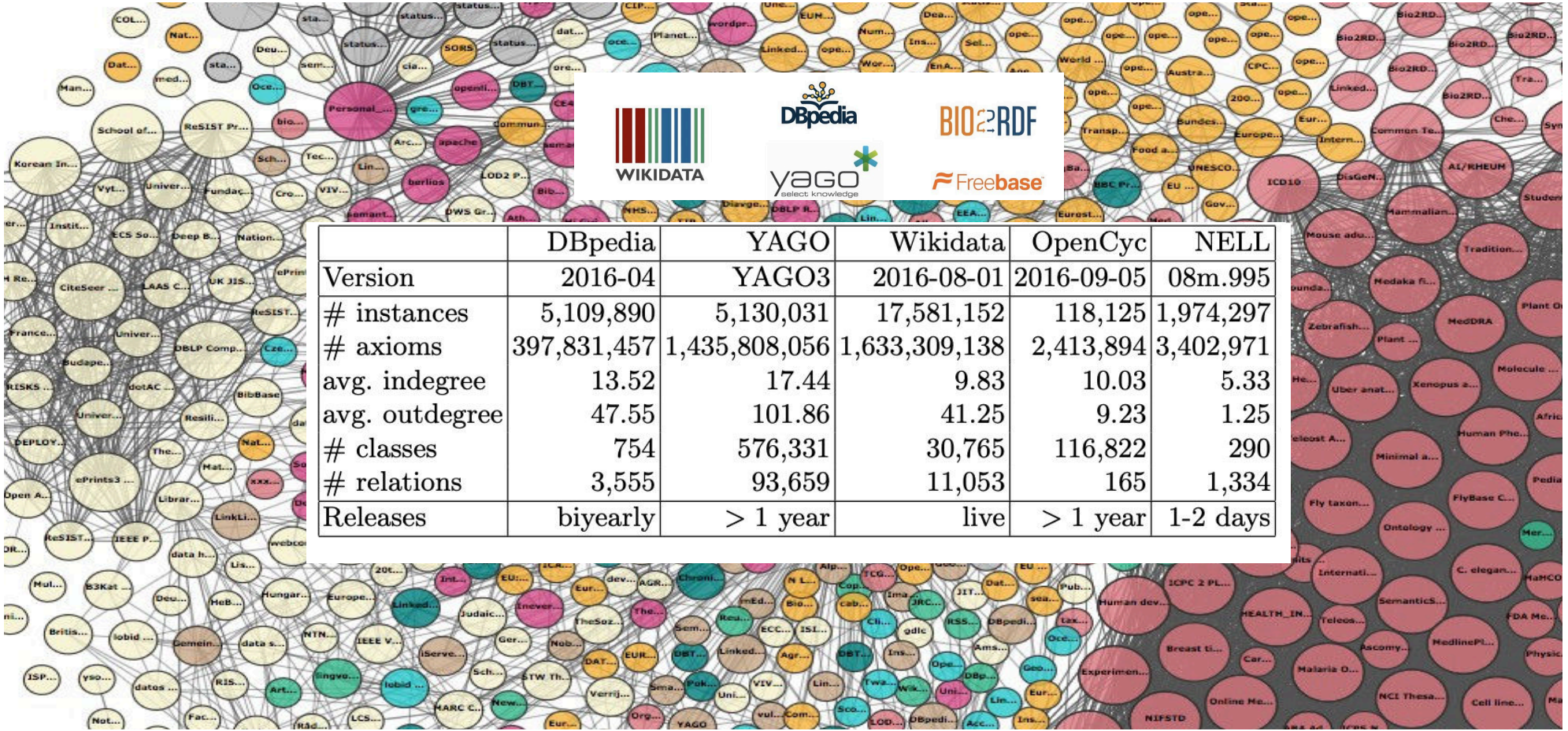
- Ontologien lassen sich durch ein abstraktes Modell darstellen
- Dieses Modell umfasst:
  - **Klassen** (um die allgemeinen Begriffe zu beschreiben)
  - **Attribute** (Eigenschaften von Dingen)
  - **Beziehungen** (zwischen Klassen)
  - Beschränkungen/“constraints“ (welche Beziehungen sind erlaubt/verboten)
  - **Individuen/Instanzen**
  - Regeln (preozedurales Wissen, if-then-else)
  - Axiome (alle Aussagen die wir über die Domäne machen wollen)

Wissensgraphen: Definition

- Ein **Knowledge Graph** besteht aus **Konzepten**, **Klassen**, **Eigenschaften**, **Beziehungen**, und **Entitätsbeschreibungen**
- Basiert auf **formalen Wissensrepräsentationen**
  z.B. Resource Description Framework (RDF(S)),
  Web Ontology Language (OWL)



Verbreitete offene Knowledge Graphen

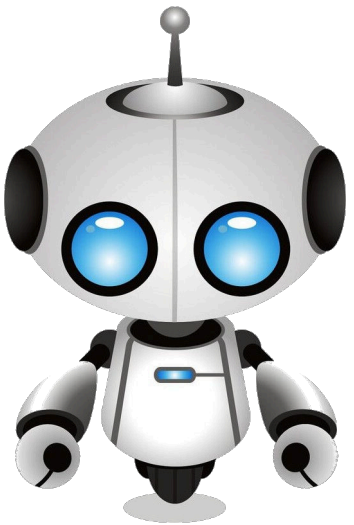


	DBpedia	YAGO	Wikidata	OpenCyc	NELL
Version	2016-04	YAGO3	2016-08-01	2016-09-05	08m.995
# instances	5,109,890	5,130,031	17,581,152	118,125	1,974,297
# axioms	397,831,457	1,435,808,056	1,633,309,138	2,413,894	3,402,971
avg. indegree	13.52	17.44	9.83	10.03	5.33
avg. outdegree	47.55	101.86	41.25	9.23	1.25
# classes	754	576,331	30,765	116,822	290
# relations	3,555	93,659	11,053	165	1,334
Releases	biyearly	> 1 year	live	> 1 year	1-2 days

Proprietäre Knowledge Graphen

	Data model	Size of the graph	Development stage
Microsoft	The types of entities, relations, and attributes in the graph are defined in an ontology.	~2 billion primary entities, ~55 billion facts	Actively used in products
Google	Strongly typed entities, relations with domain and range inference	1 billion entities, 70 billion assertions	Actively used in products
Facebook	All of the attributes and relations are structured and strongly typed, and optionally indexed to enable efficient retrieval, search, and traversal.	~50 million primary entities, ~500 million assertions	Actively used in products
eBay	Entities and relation, well-structured and strongly typed	Expect around 100 million products, >1 billion triples	Early stages of development and deployment
IBM	Entities and relations with evidence information associated with them.	Various sizes. Proven on scales documents >100 million, relationships >5 billion, entities >100 million	Actively used in products and by clients

→ Suchmaschinen, ...



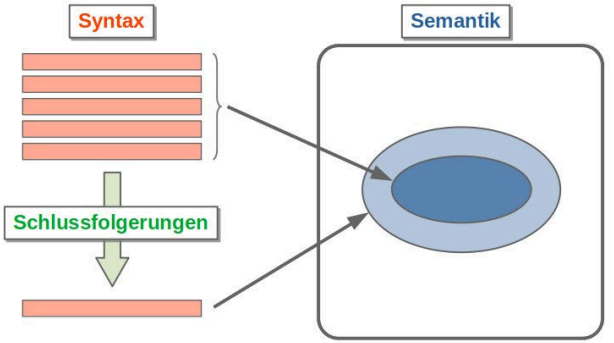
Siri, Alexa, ...

Noy *et al.*, Industry-scale Knowledge Graphs: Lessons and Challenges (2019)

Zusammenfassung

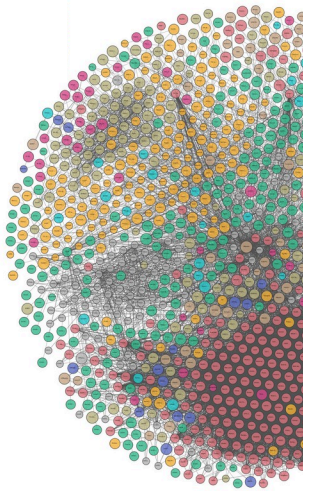
Wie können wir die Welt logisch repräsentieren?

- Syntax von Logik
- Semantik von Logik
- Logische Schlussfolgerungen
- Hier: Beschränkung auf Aussagenlogik



Wie können wir das Wissen der Welt repräsentieren?

- Was ist Wissen?
- Ontologien und ihre Darstellung
- Wissensgraphen („Knowledge Graphs“)



Vielen Dank für eure Aufmerksamkeit!

Selbst-Test Fragen

Das sollten Sie jetzt wissen

- Was bedeutet „rational“?
- Wie wird ein Suchproblem definiert?
- Was sind die "Fringe" Konten?
- Was sind die Unterschiede zwischen einem Suchbaum und einem Suchgraphen?
- Wie funktioniert der allgemeine Algorithmus für Suchalgorithmen?
- Was verstehen wir unter Vollständigkeit und Optimalität?
- Erklären Sie die Strategie von DFS, BFS, UCS und A*.
- Warum ist A* effizienter als UCS?
- Warum brauchen wir zulässige Heuristiken für A*?
- Warum sind A* und UCS optimale Suchalgorithmen und BFS und DFS nicht?

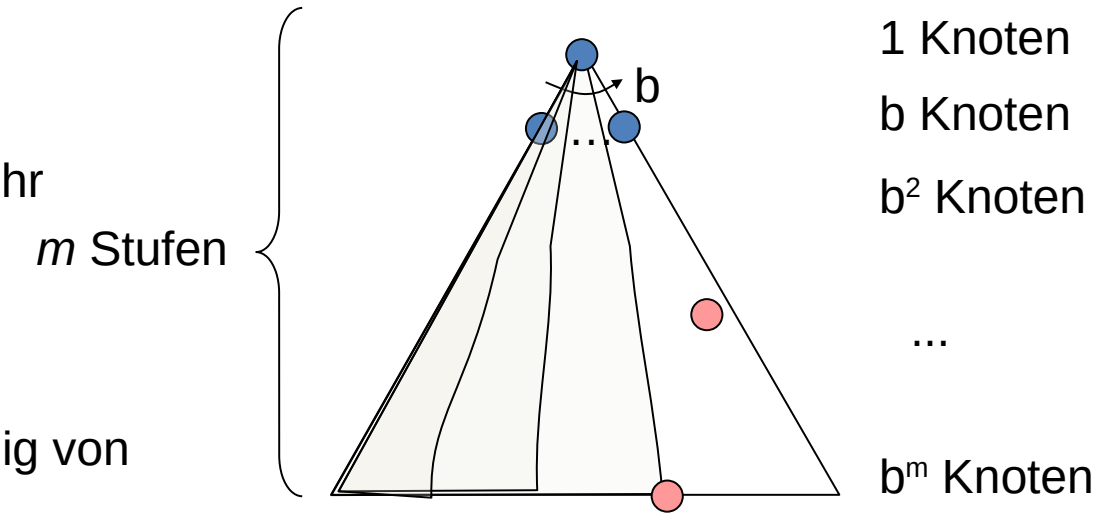
Zusatz-Folien
(nicht prüfungsrelevant)

Eigenschaften von Depth-First Suche (DFS)

Welche Knotenpunkte erweitert das DFS?
- Die tiefste linke Node im Fringe
- Könnte den ganzen Baum verarbeiten!

Ist sie vollständig?
- Jein, *m* könnte unendlich sein
  → Wir müssen Zyklen verhindern (mehr dazu später)

Ist sie optimal?
- Nein, es wird die "am weitesten links" liegende Lösung gefunden, unabhängig von Tiefe und Kosten



Eigenschaften von Breadth-First Suche (BFS)

Welche Knotenpunkte werden von BFS erweitert?

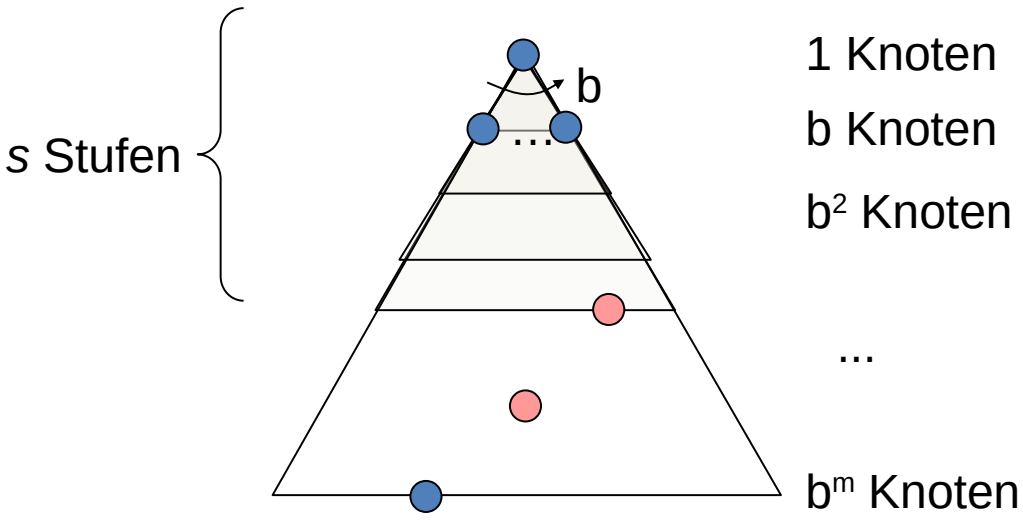
- Alle Knoten oberhalb der flachsten Lösung (Tiefe *s*)

Ist sie vollständig?

- *s* muss endlich sein, wenn es eine Lösung gibt, also ja!

Ist sie optimal?

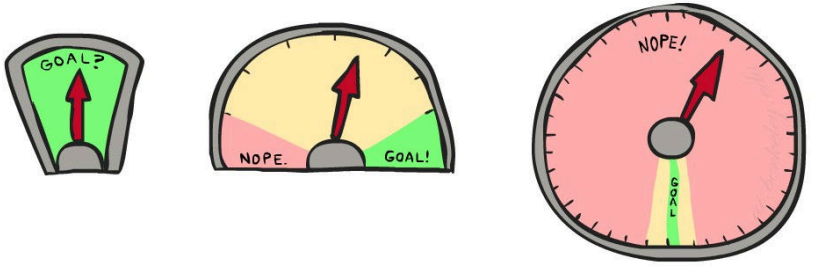
- Nur wenn alle Kosten 1 sind (mehr zu den Kosten später)



→ Warum kein Problem mit Zyklen?

Die „perfekte“ Heuristik

- Wie wäre es, die **tatsächlichen Kosten** als Heuristik zu verwenden?
  - Wäre das zulässig?
  - Würden wir an erweiterten Knotenpunkten sparen?
  - Was ist daran falsch?



- Mit A*: Heuristik ist Kompromiss zwischen Qualität der Schätzung und Arbeit pro Knoten
  → Je näher die Heuristik an die wahren Kosten herankommt, …
  - … desto weniger Knoten werden ausgebaut.
    Aber:
  - … desto mehr Arbeit wird pro Knoten für die Berechnung der Heuristik selbst benötigt.

Vergleich von Heuristiken, Dominanz

- Dominanz: $h_a \geq h_c$ wenn

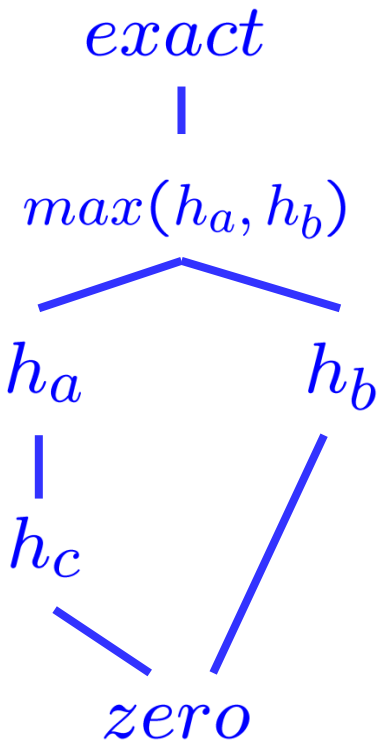
$$\forall n : h_a(n) \geq h_c(n)$$

- Kombination von Heuristiken
  - Maximal zulässige Heuristiken sind zulässig

$$h(n) = \max\left(h_a(n), h_b(n)\right)$$

- Triviale Heuristiken
  - Die Null-Heuristik wird von allen anderen Heuristiken dominiert (→ UCS)
  - Exakte Heuristiken dominieren alle anderen zulässigen Heuristiken
- Es gibt ein "Ranking" der Heuristiken

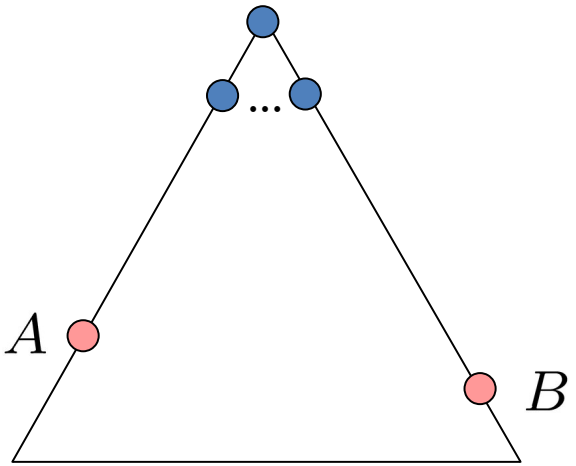
→ Verwende die dominanteste Heuristik, die das Rechenbudget zulässt.



Beweis: Optimalität der A* Suche

Annahmen
- A ist ein optimaler Zielknoten
- B ist ein suboptimaler Zielknoten
- h ist zulässig (optimistisch)

Behauptung
- A wird vor B aus dem Fringe ausgewählt werden



Optimalität der A* Suche

Beweis

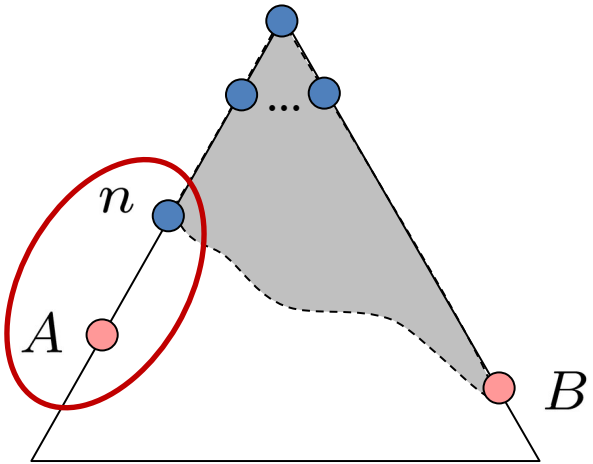
- Annahme: B ist im Fringe
- Irgendein Vorgänger n von A ist auch im Fringe (vielleicht A selbst)
- **Behauptung**: n wird vor B expandiert
  1. f(n) ist kleiner oder gleich f(A)

$f(n) = g(n) + h(n)$: Definition der f-Kosten

$f(n) \leq g(A)$: Zulässigkeit von h

$g(A) = f(A)$: h = 0 am Ziel

→ $f(n) \leq f(A)$



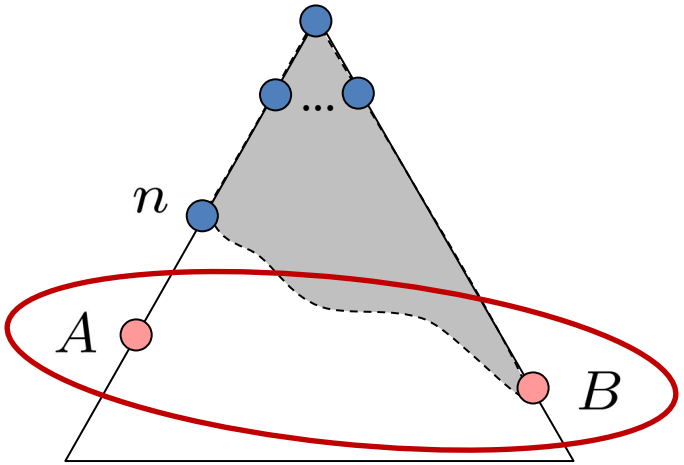
Optimalität der A* Suche

Beweis

- Annahme: B ist im Fringe
- Irgendein Vorgänger n von A ist auch im Fringe (vielleicht A selbst)
- **Behauptung**: n wird vor B expandiert
  1. f(n) ist kleiner oder gleich f(A)
  2. f(A) ist kleiner als f(B)

$g(A) < g(B)$: B ist suboptimal

$f(A) < f(B)$ da h = 0 am Ziel

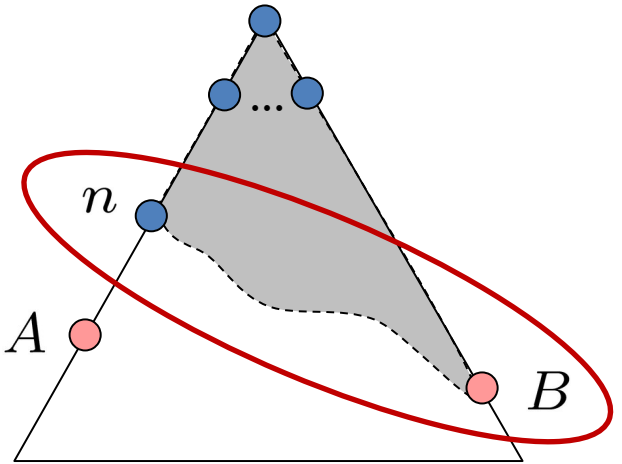


Optimalität der A* Suche

Beweis

- Annahme: B ist im Fringe
- Irgendein Vorgänger n von A ist auch im Fringe (vielleicht A selbst)
- **Behauptung**: n wird vor B expandiert
  1. f(n) ist kleiner oder gleich f(A)
  2. f(A) ist kleiner als f(B)
  3. n expandiert vor B

$$f(n) \leq f(A) < f(B)$$

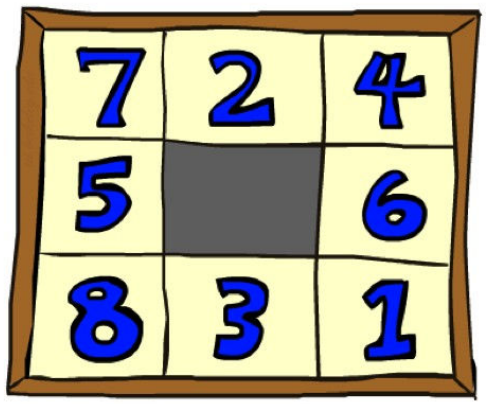


→ Alle Vorgänger von A expandieren vor B

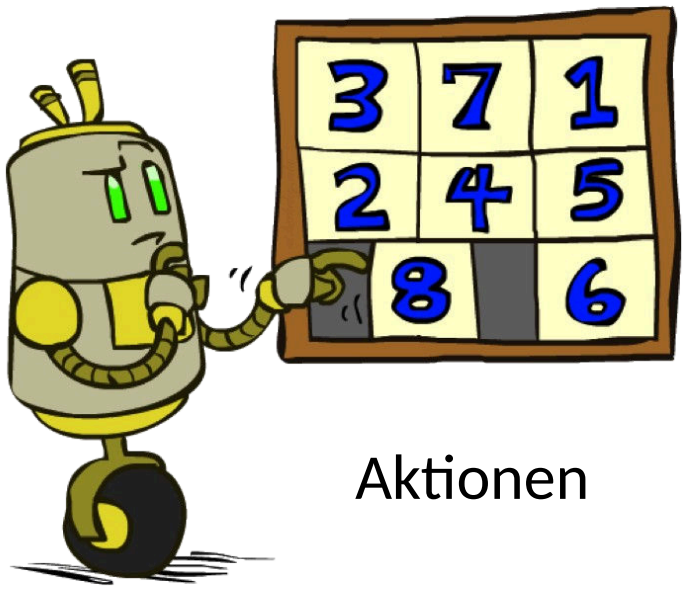
→ A expandiert vor B

→ A* Suche ist optimal

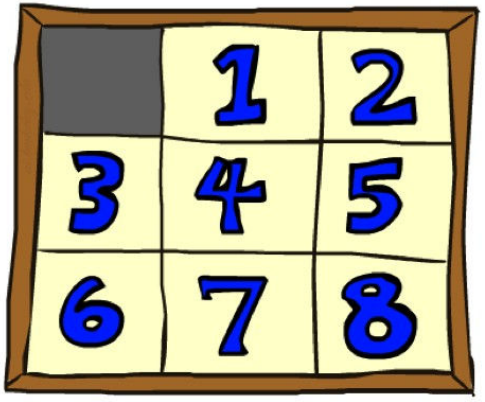
Beispiel: Sortier-Puzzle



Startzustand



Aktionen

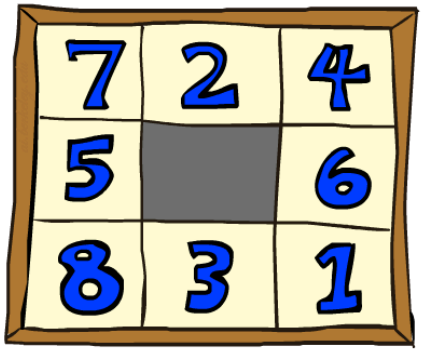


Zielzustand

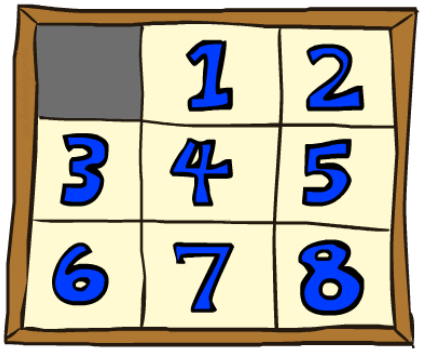
- Was sind die Zustände?
- Wie viele Zustände?
- Was sind die Aktionen?
- Wie viele Nachfolger hat der Anfangszustand?
- Wie hoch sollten die Kosten sein?

Heuristik I

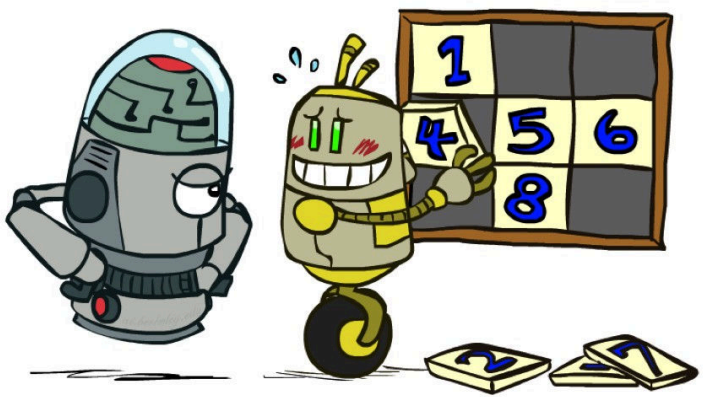
- Heuristik: **Anzahl** der falsch platzierten Zahlen
- h(Start) = 8
- **Warum ist diese Heuristik zulässig?**



Startzustand



Zielzustand

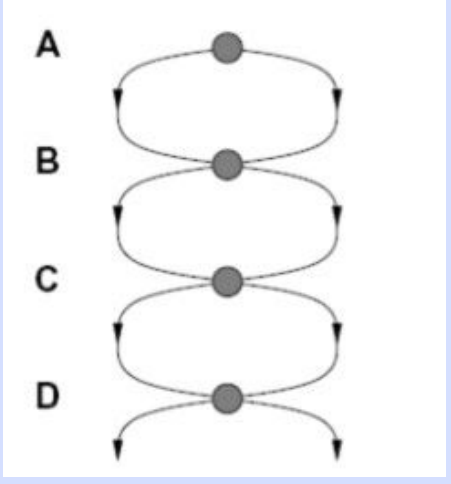


	Zahl der expandierten Nodes bei einem Abstand zum Ziel von …		
	…4 Schritten	…8 Schritten	…12 Schritten
UCS	112	6,300	3.6×10^6
A*: Anzahl	13	39	227

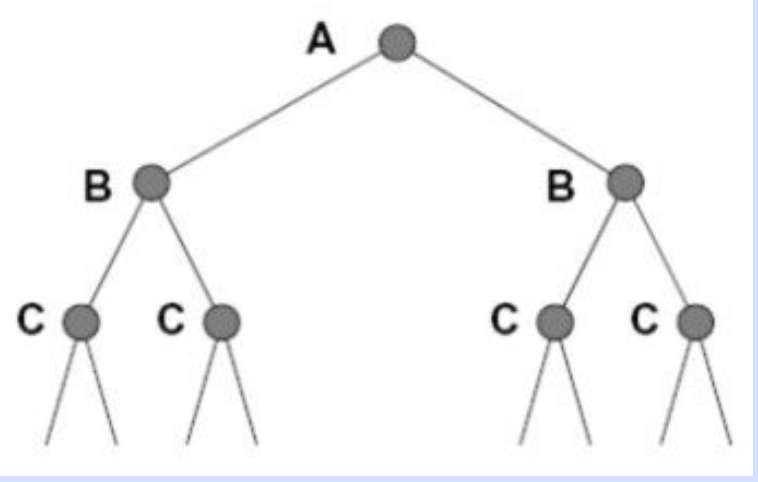
Baumsuche: Extra-Arbeit!

Wenn wiederholte Zustände nicht erkannt werden, kann dies exponentiell mehr Arbeit verursachen.

Zustandsgraph



Suchbaum

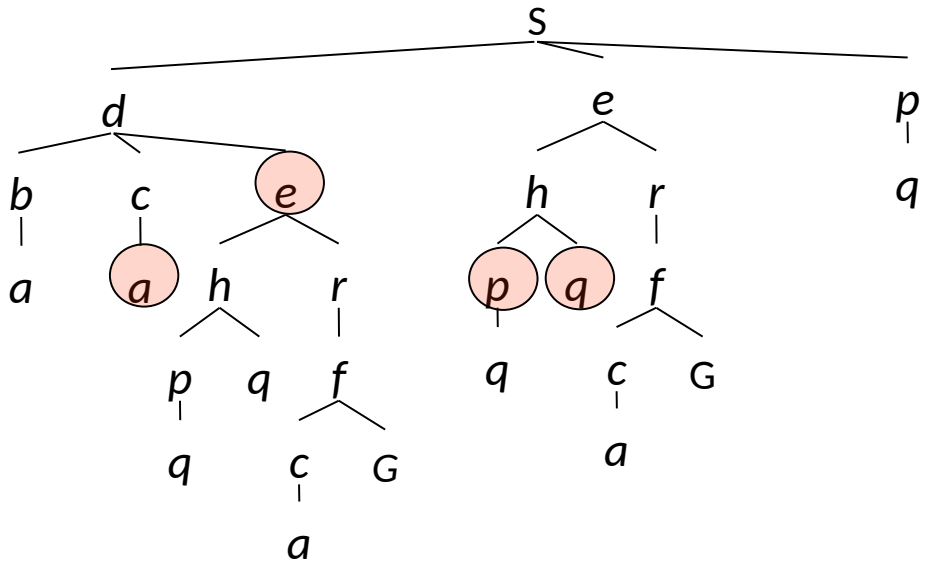


Beispiel

Wir sollten uns (z.B. in der Breitensuche) nicht die Mühe machen, die markierten Knoten zu expandieren!

Warum?

→ Weil diese bereits über einen anderen Pfad mit gleicher (a) oder kleinerer (e, p, q) Schrittzahl expandiert wurden.



Graph Suche

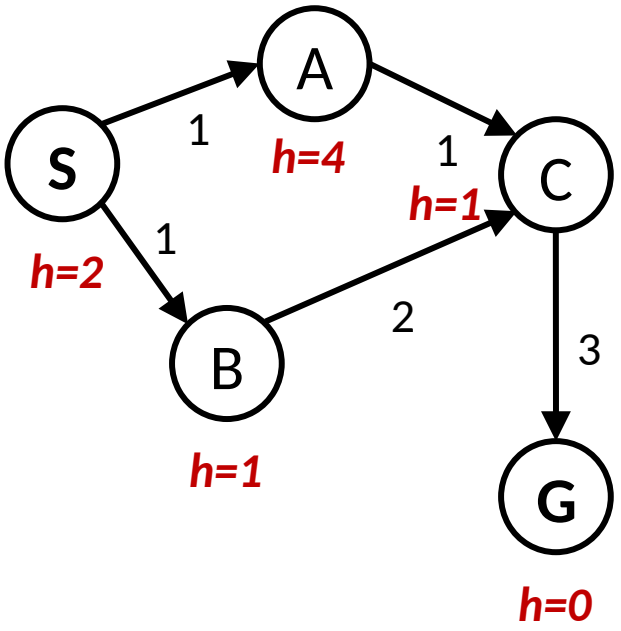
Idee
- Niemals einen Zustand zweimal expandieren

Umsetzung
- Baumsuche + Set der bereits erweiterten Zustände
- Erweitere den Suchbaum Knoten für Knoten, aber...
- … prüfe vor dem Expandieren eines Knotens, ob der Zustand bereits expandiert wurde
  → Wenn bereits expandiert, dann überspringen (aus dem Fringe entfernen)
  → Wenn nicht, zum Set hinzufügen

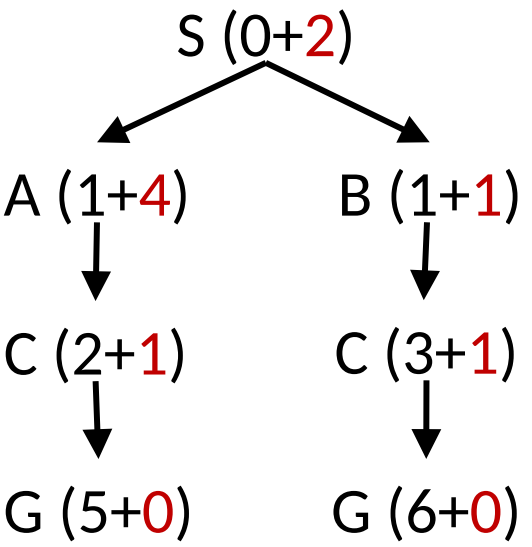
Vollständigkeit
- Kann die Graphensuche die Vollständigkeit zunichte machen? **Warum/warum nicht?**
- Wie sieht es mit der Optimalität aus? **Warum/warum nicht?**

A* Graph Suche versagt

Zustandsgraph

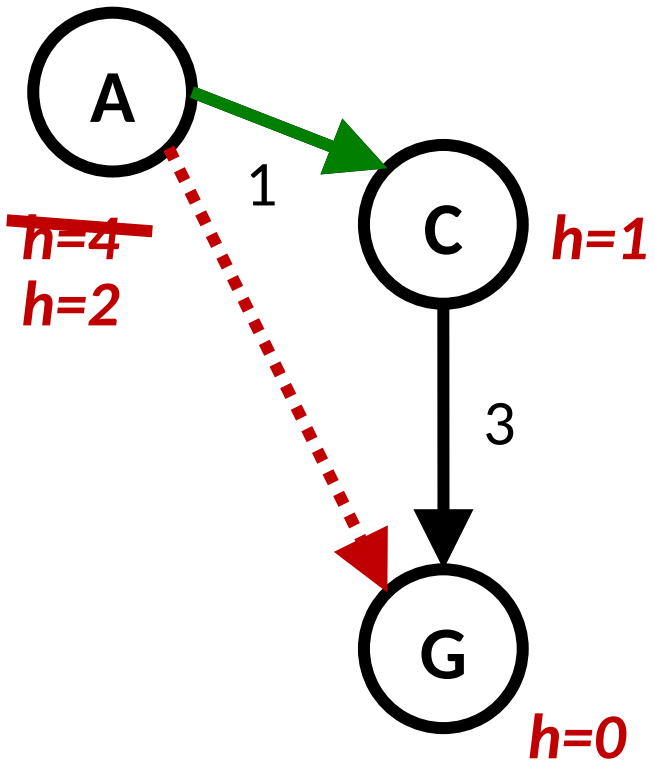


Suchbaum



Die Top-Route (über A) wird nie ausgewertet, da C bereits besucht wurde!

Konsistenz von Heuristiken



Hauptgedanke

Geschätzte heuristische Kosten ≤ tatsächliche Kosten

→ **Zulässigkeit**

- heuristische Kosten ≤ tatsächliche Kosten für das Ziel

$$h(A) \leq \text{tatsächliche Kosten von A nach G}$$

→ **Konsistenz**

- für jeden Pfad: heuristische "Pfad"-Kosten ≤ tatsächliche Kosten

$$h(A) - h(C) \leq \text{Kosten(A bis C)}$$

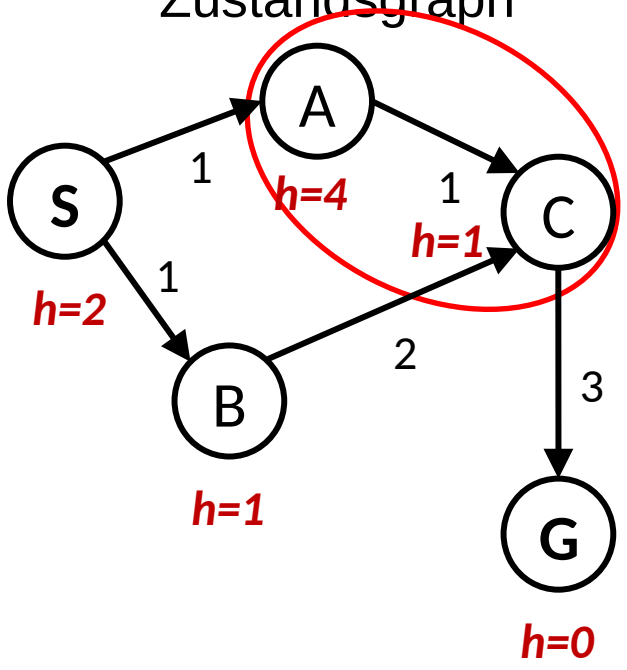
Die Folgen der Konsistenz

Der f-Wert entlang eines Pfades nimmt nie ab

$$h(A) \leq \text{Kosten(A bis C)} + h(C)$$
$$g(A) + h(A) \leq g(A) + \text{Kosten(A bis C)} + h(C)$$
$$f(A) \leq f(C)$$

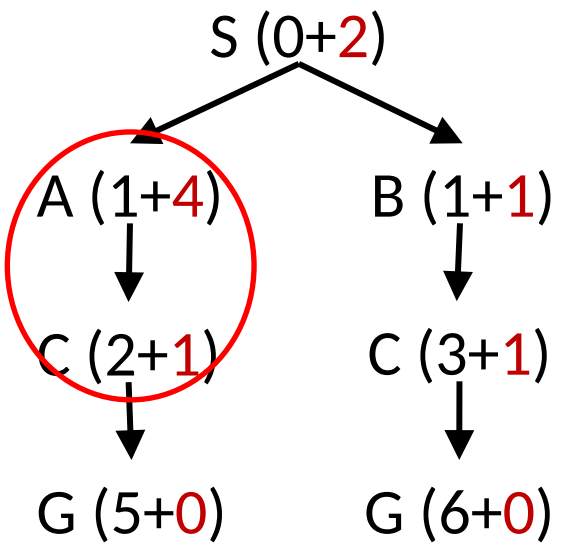
A* Graph Suche versagt

Zustandsgraph



Nicht konsistent

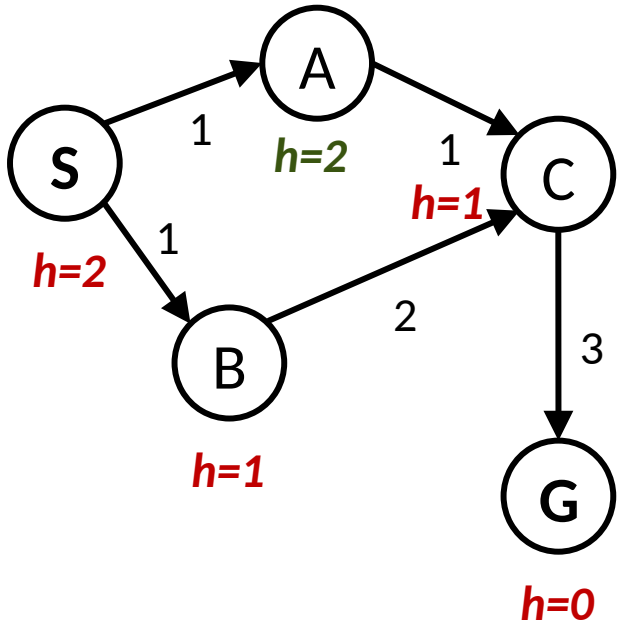
Suchbaum



Die Top-Route (über A) wird nie ausgewertet, da C bereits besucht wurde!

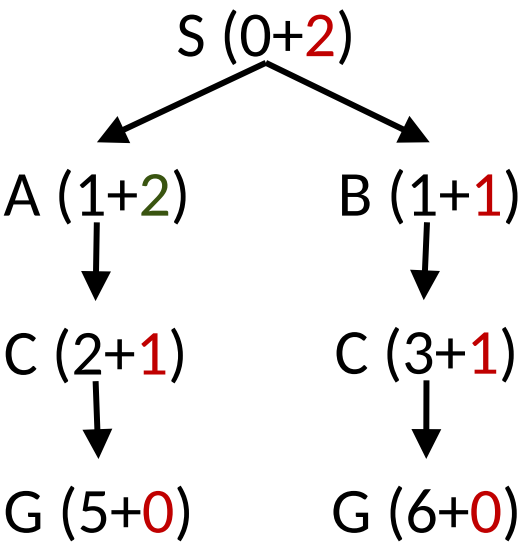
A* Graph Suche repariert

Zustandsgraph



Konsistent

Suchbaum



Die Top-Route (über A) wird **nach** dem Besuch von B erkundet, aber **vor** der Erkundung der suboptimalen unteren Route

Optimalität: Zusammenfassung

- **Baumsuche**
  - A* ist optimal, wenn die Heuristik **zulässig** ist
  - UCS ist ein Sonderfall (h = 0)

- **Graph Suche**
  - A* optimal, wenn die Heuristik **konsistent** ist
  - UCS ist optimal (da h = 0 konsistent ist)

- **Konsistenz impliziert Zulässigkeit**

- **Im Allgemeinen** neigen die meisten natürlichen zulässigen Heuristiken dazu, konsistent zu sein, insbesondere wenn sie von vereinfachten Problemen ausgehen