

A1

Recherchieren Sie, welche Probleme bereits mittels Computer- bzw. Robotereinsatz gelöst werden können und welche aktuell noch ungelöst sind.

Gelöste Probleme:

- Sprachübersetzung
- Schnelle Kommunikation über weite Entfernung
- Bildung (Zugriff auf Wissen)
- Wissenssammlung
- Rechen-/mathematische Probleme
- Reinigung (Staubsauger, Rasenmäher)
- Digitale Zahlungen (Transaktionen)
- Unterhaltung
- Fachkräftemangel abfedern
- Fabriken in der Produktion (Industrie)

Ungelöste Probleme:

- Mangel an emotionaler Intelligenz (Wichtig in Gesundheitsfürsorge/Kundenservice)
- Fake News (Lässt sich nicht endgültig lösen)
- Datenschutz
- Sicherheit (Kein Computersystem ist zu einhundert Prozent vor Fehlern geschützt)(Cyber-Angriffe)
- Roboter können sich nicht selbst reparieren
- Kreativität und Fantasie (keine neuen oder originellen Ideen, nur innerhalb der Grenzen der Daten kreativ)
- Starke Abhängigkeit von Daten
- Ressourcen
- Menschenverstand

A1

Recherchieren und diskutieren Sie Auswirkungen auf die Gesellschaft durch die KI, etwa durch autonomes Fahren oder durch Large Language Models (LLM).

- Kognitiver Abbau
 - > Schüler/Studenten nutzen KI zur Erzeugung von Lösungen für ihre Aufgaben, anstatt selbst nachdenken und diese zu lösen
- Zunahme der Verwendung von Wörtern, die bevorzugt von ChatGPT erzeugt werden in der menschlichen Kommunikation
- KI ersetzt und vereinfacht Arbeit
 - > Arbeitsplatzverlust bzw. weniger Jobangebot
 - > Hohe Leistungserwartungen an Arbeitnehmer
- Desinformation (Fake-News) werden leichter verbreitet durch KI
 - > KI-Technologie erstellt in kürzester Zeit und mit geringstem Aufwand überzeugende Fakes
- Selbstfahrende Autos werden einen höheren Energieverbrauch, Verkehrsstaus und Zersiedelung verursachen
- Verbesserte Mobilität für ältere, junge und behinderte Menschen durch autonome Autos
- KI-generierte Ratschläge beeinflussen das moralische Urteilsvermögen von Menschen, wenn sie moralische Entscheidungen mit KI treffen
- Reduzierung von Autounfällen
- Reduzierung der Kfz-Versicherung

A2 Search.01

- = Elben
 - = Orks
 - = Pfend
- (so abstrakt wie möglich halten)

Gegeben: 3x Elben und 3x Orks die einen Fluss überqueren wollen

Ziel: Alle Elben und Orks über den Fluss bringen

Bedingungen:

↳ Pfend kann den Fluss **nicht alleine** überqueren & kann **maximal zwei Wesen** gleichzeitig tragen

↳ Orks > Elben zu keiner Zeit an keinem Ufer, da es sonst zu Konflikten zwischen beiden Gruppen kommt

Aktionen:

- Pick Orks (POS)
- Pick Elben (PES)
- Pick OrkElbe (POE)
- Pick Ork (PO)
- Pick Elbe (PE)

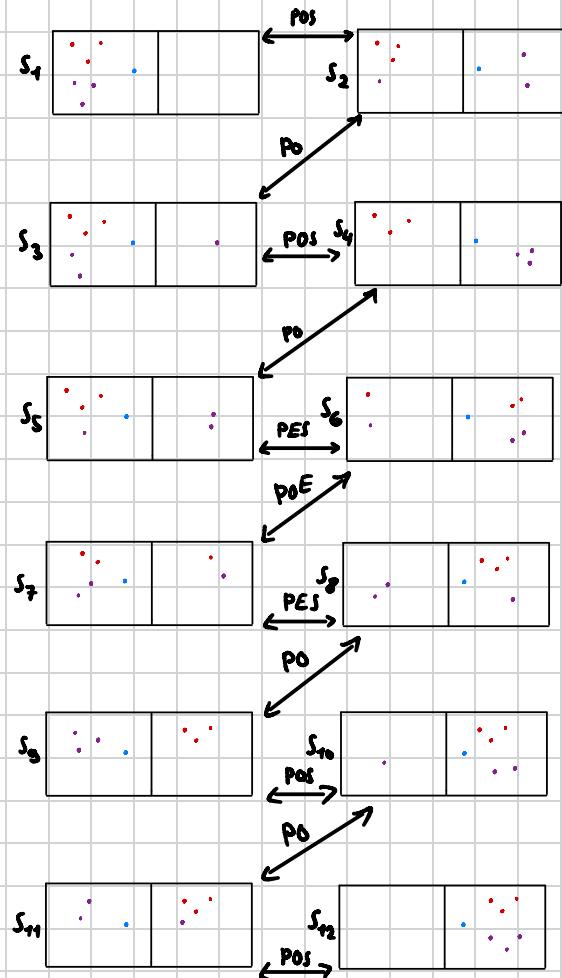
(Pick inkl. Bewegung zur anderen Uferseite)

(Deterministisch)

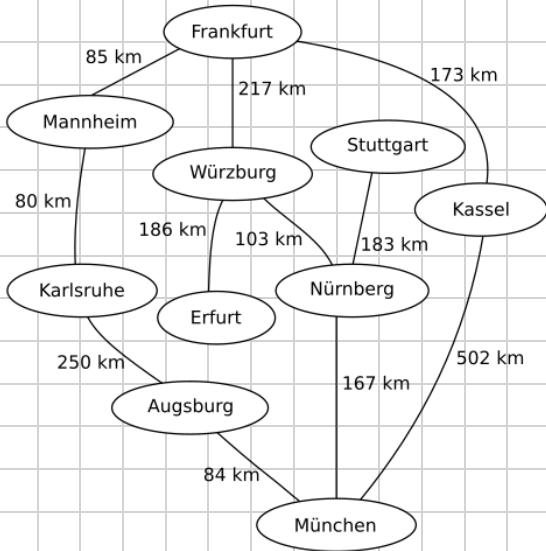
$$S_A = \{S_1\}$$

$$S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12}\}$$

$$S_E = \{S_{12}\}$$



A3 Search.02



Stadt	$h(n)$
Augsburg	0 km
Erfurt	400 km
Frankfurt	100 km
Karlsruhe	10 km
Kassel	460 km
Mannheim	200 km
München	0 km
Nürnberg	537 km
Stuttgart	300 km
Würzburg	170 km

↵ AG
 ↵ EF
 ↵ FF
 ↵ KR
 ↵ KS
 ↵ MA
 ↵ MÜ
 ↵ NU
 ↵ ST
 ↵ WÜ

1. Finden Sie nacheinander mit Tiefensuche und Breitensuche (jeweils in den Graph-Search-Variante) sowie A* (in der Tree-Search-Variante mit der Verbesserung "keine Zyklen") jeweils einen Weg von Würzburg nach München.

Tiefensuche (Graph-Search) (Stack)

[WÜ]

[WÜEF, WÜFF, WÜNU]

[WÜFF, WÜNU]

[WÜFFKS, WÜFFMA, WÜNU]

[WÜFFKSMÜ, WÜFFMA, WÜNU]

[WÜ]

[WÜ, EF]

[WÜ, EF, FF]

[WÜ, EF, FF, KS]

+ Welche Knoten/Zustände sind bereits in der Datenstruktur?

Breitensuche (Graph-Search) (Queue)

[WÜ]

[WÜEF, WÜFF, WÜNÜ]

[WÜFF, WÜNÜ]

[WÜNÜ, WÜFFKS, WÜFFMA]

[WÜFFKS, WÜFFMA, WÜNÜMÜ, WÜNST]

[WÜFFMA, WÜNÜMÜ, WÜNST]

[WÜNÜMÜ, WÜNST, WÜFFMAKR]

[WÜ]

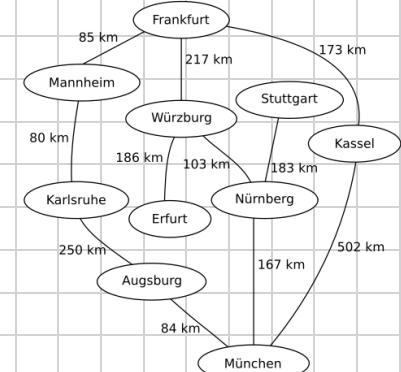
[WÜ, EF]

[WÜ, EF, FF]

[WÜ, EF, FF, NÜ]

[WÜ, EF, FF, NÜ, KS]

[WÜ, EF, FF, NÜ, KS, MA]



Stadt	$h(n)$
Augsburg	0 km
Erfurt	400 km
Frankfurt	100 km
Karlsruhe	10 km
Kassel	460 km
Mannheim	200 km
München	0 km
Nürnberg	537 km
Stuttgart	300 km
Würzburg	170 km

←AG
←EF
←FF
←KR
←KS
←MA
←HÜ
←NÜ
←ST
←WÜ

A* (Tree-Search mit der Verbesserung keine Zyklen) (Präd. wo Kunden mehrfach vorkommt)
 (sortierte/prioritäre-Queue)

[WÜ 0 + 170 = 170]

[WÜEF 186 + 400 = 586, WÜFF 317, WÜNU 640]

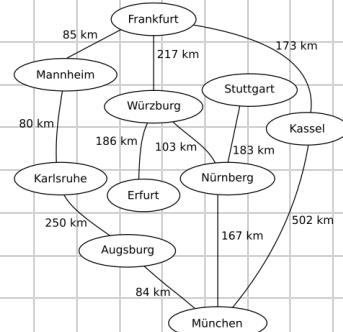
[WÜEF 586, WÜNU 640, WÜFFKS 850, WÜFFMAK 502]

[WÜEF 586, WÜNU 640, WÜFFKS 850, WÜFFMAK R 392]

[WÜEF 586, WÜNU 640, WÜFFKS 850, WÜFFMAK RAG 632]

[WÜNU 640, WÜFFKS 850, WÜFFMAK RAG 632]

[WÜNU 640, WÜFFKS 850, WÜFFMAK RAG MÜ 716]



Verbesserung:

Präd., deren Endknoten bereits früher im Präd. vorkommt, werden in Schritt 2 in die Queue nicht aufgenommen.

→
 Grund: Übersetzung in Nürnberg

Stadt	$h(n)$	
Augsburg	0 km	← AG
Erfurt	400 km	← EF
Frankfurt	100 km	← FF
Karlsruhe	10 km	← KR
Kassel	460 km	← KS
Mannheim	200 km	← MA
München	0 km	← HÜ
Nürnberg	537 km	← NÜ
Stuttgart	300 km	← ST
Würzburg	170 km	← WÜ

Vergleichen Sie die drei Algorithmen: Wie viele Einfüge gibt es in der Datenstruktur maximal, wie oft wird die Hauptschleife durchlaufen (also ein Element aus der Datenstruktur entnommen, untersucht und weiterentwickelt)?

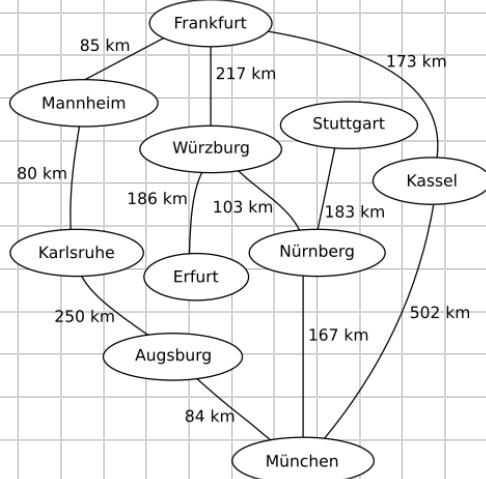
Materialien Einübung	Wie oft wird expandiert? (erktivie Zielzustand)
- Tiefensuche	3
- Breitensuche	4
- A*	4

Fazit:

Von der Zeitskomplexität hat die Tiefensuche am effizientesten gearbeitet. Alle Algorithmen haben auch eine Lösung gefunden, wobei die ^{Lösung} der Breitensuche am besten ist. Die Breitensuche und A* sind von der Zeitskomplexität gleich schnell gewesen, wobei beachtet werden muss, dass A* mit einer unzulässigen Heuristik und Tree-Search-Variante gearbeitet hat. A* hätte am schnellsten und optimalsten gearbeitet mit einer zulässigen Heuristik. Da die Tiefensuche Pfade in die Tiefe verfolgt hat sie schnell aufgrund der Graphstrukture eine Lösung gefunden. Demgegenüber hat die Breitensuche sich ebenenweise entwickelt, den Weg mit den geringsten Schritte gesucht und gefunden. In der Tree-Search-Variante wäre die Breitensuche mit Abstand am aufwendigsten gewesen.

Stadt	$h(n)$
Augsburg	0 km
Erfurt	400 km
Frankfurt	100 km
Karlsruhe	10 km
Kassel	460 km
Mannheim	200 km
München	0 km
Nürnberg	537 km
Stuttgart	300 km
Würzburg	170 km

↙ AG
 ↙ EF
 ↙ FF
 ↙ KR
 ↙ KS
 ↙ MA
 ↙ Mü
 ↙ NÜ
 ↙ ST
 ↙ WÜ



2. Dürfen die oben gegebenen Restkostenschätzungen in λ^* verwendet werden?

Tree-Search

Die Bedingung für die λ^* Suche an die gegebene Heuristik lautet $h(u) \leq h^*(u)$, wobei h^* die tatsächlichen optimalen Restkosten von einem Knoten u zum nächsten Ziel sind. Zusätzlich gilt für jeden Ziellknoten u $h(u)=0$ und jeden Knoten $h(u) \geq 0$.

$h(u) \geq 0$ ✓ Negative Schätzungen sind verboten.

$h(u)=0$ München ✓

$$h^*(u) \text{ Augsburg} = 84 \quad \checkmark$$

$$h(u) \leq h^*(u)$$

$$h^*(u) \text{ Erfurt} = 186 + 103 + 167 = 456 \quad \checkmark$$

$$h^*(u) \text{ Frankfurt} = 173 + 502 = 675 \quad \checkmark$$

$$h^*(u) \text{ Karlsruhe} = 250 + 84 = 334 \quad \checkmark$$

$$h^*(u) \text{ Kassel} = 502 \quad \checkmark$$

$$h^*(u) \text{ Mannheim} = 80 + 250 + 84 = 414 \quad \checkmark$$

$$h^*(u) \text{ München} = 0 \quad \checkmark$$

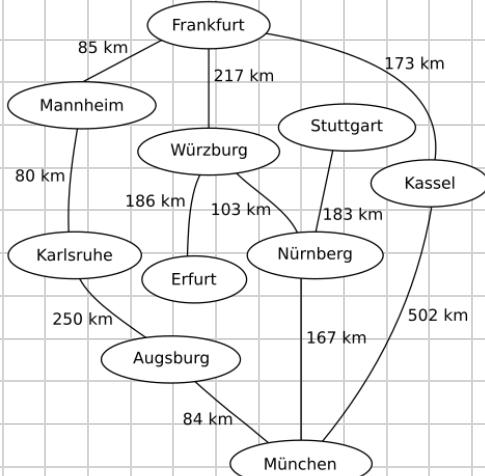
$$h^*(u) \text{ Nürnberg} = 167 \quad \times \quad \leftarrow \text{Hier gibt es eine Überschätzung in } h(u) \text{ und dies}$$

$$h^*(u) \text{ Stuttgart} = 183 + 167 = 350 \quad \checkmark \quad \text{bricht die Anforderung für } \lambda^* \text{ in der Tree-Search-Variante}$$

$$h^*(u) \text{ Würzburg} = 103 + 167 = 270 \quad \checkmark$$

Stadt	$h(n)$
Augsburg	0 km
Erfurt	400 km
Frankfurt	100 km
Karlsruhe	10 km
Kassel	460 km
Mannheim	200 km
München	0 km
Nürnberg	257 km
Stuttgart	300 km
Würzburg	170 km

↪ AG
 ↪ EF
 ↪ FF
 ↪ KR
 ↪ KS
 ↪ MA
 ↪ HÜ
 ↪ NÜ = 167 km
 ↪ ST (Korrigierung der
 ↪ WÜ Kostätzung)



A^* (Tree-Search mit der Verbesserung keine Zyklen und Korrigierung Abschätzungen)
 (sortierte/prioritär-Queue)

[WÜ 0 + 170 = 170]

[WÜEF 186 + 400 = 586, WÜFF 217 + 100 = 317, WÜNÜ 103 + 167 = 270]

[WÜEF 586, WÜFF 317, WÜNÜ 270]

A4 Search.03

Quelle: Buch Modern Approach S.99 (Rechts steht auch Domination in Türkis)

$h(n)$ Geschätzte Restkosten für den Weg von Knoten n zum Ziel

Optimalität: Sind gefundene Lösungen garantiert optimal?

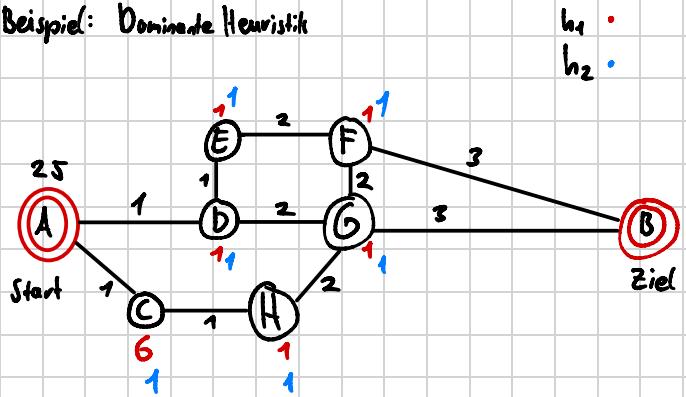
Der optimale / beste Pfad ist der mit den geringsten tatsächlichen Kosten.

Was bedeutet: „Eine Heuristik $h_1(n)$ dominiert eine Heuristik $h_2(n)$ “?

Dies bedeutet, dass für jeden Knoten (Zustand) n in den beiden Heuristiken gilt $h_1(n) \geq h_2(n)$. (Kürzer: $\forall n: h_1(n) \geq h_2(n)$)

Wie wirkt sich die Nutzung einer dominierenden Heuristik $h_1(n)$ in A^* aus (im Vergleich zur Nutzung einer Heuristik h_2 , die von h_1 dominiert wird)?

Beispiel: Dominante Heuristik



Optimale L \ddot{a} sung:

A \rightarrow D \rightarrow G \rightarrow B

Mit den tats \ddot{a} chlichen Kosten 6

$$h_1 : f(c) = 1 + 6 = 7 \quad f(c) > C^* \quad \text{Der wird sicher nicht expandiert. (1 Knoten wird gespart)}$$

$$h_2 : f(c) = 1 + 1 = 2 \quad f(c) < C^* \quad \text{Der Knoten wird sicher expandiert. (Der selbe Knoten wird expandiert)}$$

(h₂ expandiert mehr, heißt mehr Aufwand)

Auswirkung wenn $h_1(n) \geq h_2(n)$:

- Suchaufwand wird gespart

\hookrightarrow Präzisere Schätzungen

- Optimalit \ddot{a} g \ddot{a} ntiert nur wenn Heuristik zulässig bzw. konsistent (zu große Werte Optimalit \ddot{a} nicht garantiert)

Da $h(n)$ immer kleiner oder gleich der tatsächlichen optimalen Bestkosten von einem Knoten n zum nächsten Ziel sein muss, bedeutet es je größer n desto präziser an den tatsächlichen optimalen Bestkosten. Das heißt, die dominante Heuristik schätzt immer gleich oder besser die Distanz zum Ziel ein von einem gegebenen Zustand.

Weitere gültige Bedingungen: (Quelle: Beck Modern Approach S.88 - 90 für A*)

Wenn C^* optimale Pfadkosten bei der A^* Suche ist, dann gilt, dass wenn $f(n)$ kleiner als C^* ist, diese Knoten sicher expandiert werden. Wenn $f(n)$ größer als C^* ist, dann werden diese Knoten sicher nicht expandiert. Die expandierten Knoten in h_1 sind somit eine Teilmenge oder dieselben expandierten Knoten von $h_2(n)$.

C^* Pfadkosten des günstigsten / besten Zielpfads

($f(n) = C^*$ optimales Pfad)

A5 Search 0.4

Beweisen Sie, dass A* in der Tree-Search-Variante bei Nutzung einer zulässigen Heuristik optimal ist.

(Widerspruchsbeweis)

Angenommen A* in der Tree-Search-Variante bei Nutzung einer zulässigen Heuristik ist nicht optimal, dann müsste der Algorithmus einen Pfad zurückgeben der von den Kosten her teurer als der optimale Pfad ist. Damit das passiert muss mind. 1 Knoten auf dem optimalen Pfad ignoriert bzw. nicht expandiert werden. Nach der Definition wird $f(n) = g(n) + h(n)$ gerechnet. Damit ein Knoten ignoriert wird gilt, muss $f(n) >$ optimale Pfadkosten sein, um dies zu bewirken muss $h(n) > h^*(n)$ sein, das kann in einer zulässigen Heuristik nicht passieren, dementsprechend findet der Algorithmus immer den optimalsten Weg.