Intelligente Systeme
- Problemlösen –
Suchen, Spielen, Planen

Hochschule für Angewandte Wissenschaften Hamburg
Department Informatik

Dr.-Ing. Sabine Schumann

Roadmap

1. Einleitung
2. Logik: Aussagenlogik, Prädikatenlogik erster Stufe
3. Prolog
(4. DCG)
5. Problemlösen: Suchen, Spielen, Planen
6. Constraints
7. Soft Computing
8. Neuronale Netze
9. Semantische Netze & Frames

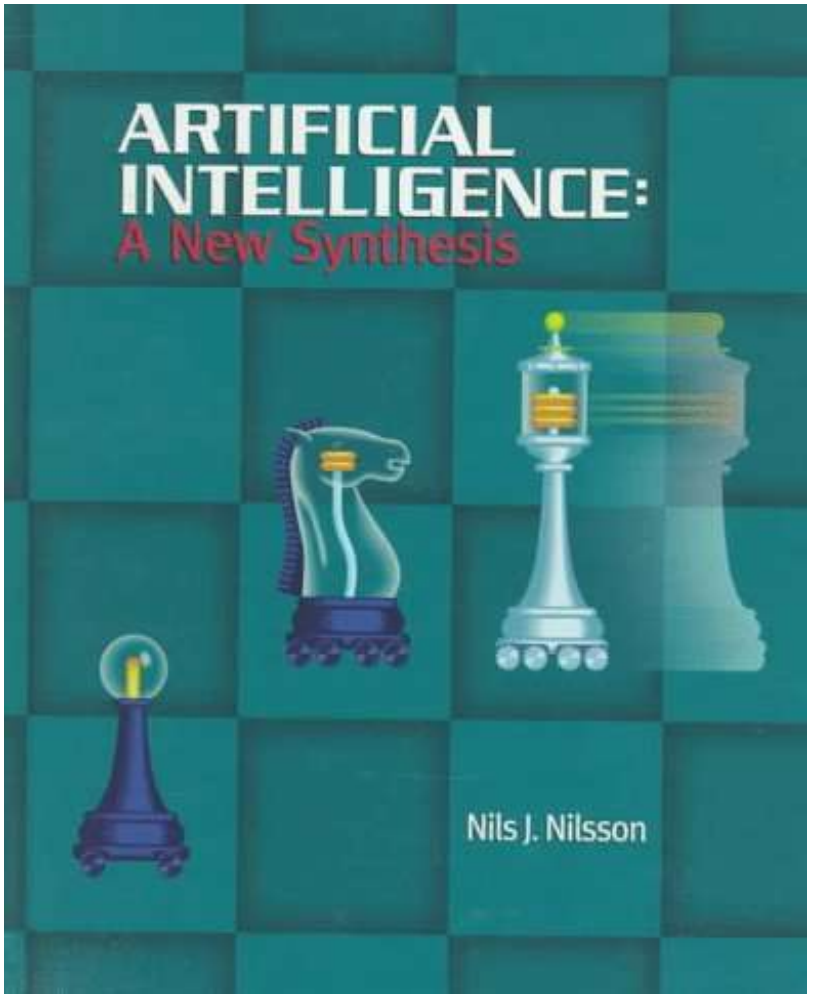
Roadmap

Problemlösen und Suche

- **Modellierung, Wissensrepräsentation**
- **Suche**
  - **uninformierte,**
  - **heuristische,**
  - **Optimierungsprobleme.**
- Spiele
  - MiniMax, α/β Schnitt
- Regelbasierte Systeme
- Planungsprobleme

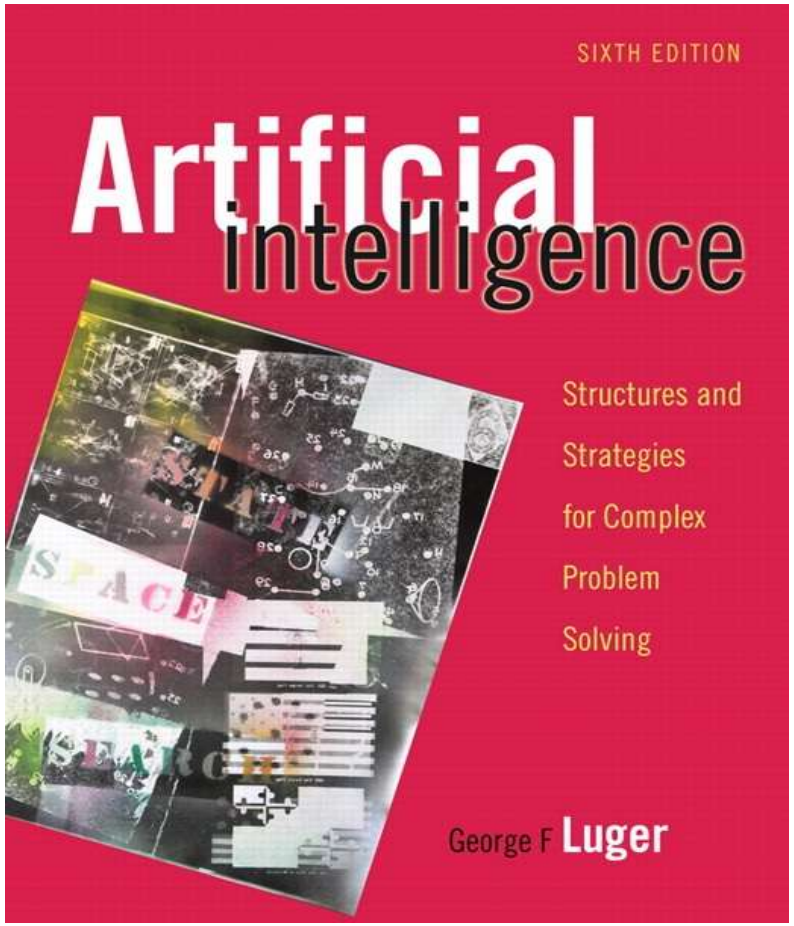
Empfohlene weiterführende Literatur (1)

Stuart Russell, Peter Norvig:
Künstliche Intelligenz – Ein moderner Ansatz.
2. Aufl., Pearson Studium, 2004



Nils J. Nilsson:
Artificial Intelligence: a new synthesis.
Morgan Kaufmann Publisher, 1998.

Empfohlene weiterführende Literatur (2)



George F. Luger:
Künstliche Intelligenz – Structures and Strategies for Complex Problem Solving.
6. Aufl., Pearson Studium, 2009

Ingo Boersch, Jochen Heinsohn, Rolf Socher:
Wissensverarbeitung: Eine Einführung in die Künstliche Intelligenz für Informatiker und Ingenieure.
2. Aufl., Spektrum Akademischer Verlag, 2007

Empfohlene weiterführende Literatur (3)

Christoph Beierle, Gabriele Kern-Isberner:
Methoden wissensbasierter Systeme
4. Auflage, Vieweg+Teubner, 2008

Günther Görz (Hrsg.):
Einführung in die Künstliche Intelligenz.
Addison-Wesley, 2. Aufl. 1995

Ein richtiges Problem hat keine Lösung.

Smith‘s Gesetz,
Arthur Bloch: Murphy‘s Gesetze in einem Band, 1985

Motivation

Lassen sich komplexe Probleme durch (geschickte) Suche lösen?

Was ist Problemlösen?

Unter Problemlösen versteht man das Bestreben, einen gegebenen Zustand (Anfangs- bzw. Startzustand) in einen anderen, gewünschten Zustand (Zielzustand) zu überführen, wobei es gilt, eine Barriere zu überwinden, die die unmittelbare Überführung des Startzustandes in den Zielzustand verhindert.

Modellierung

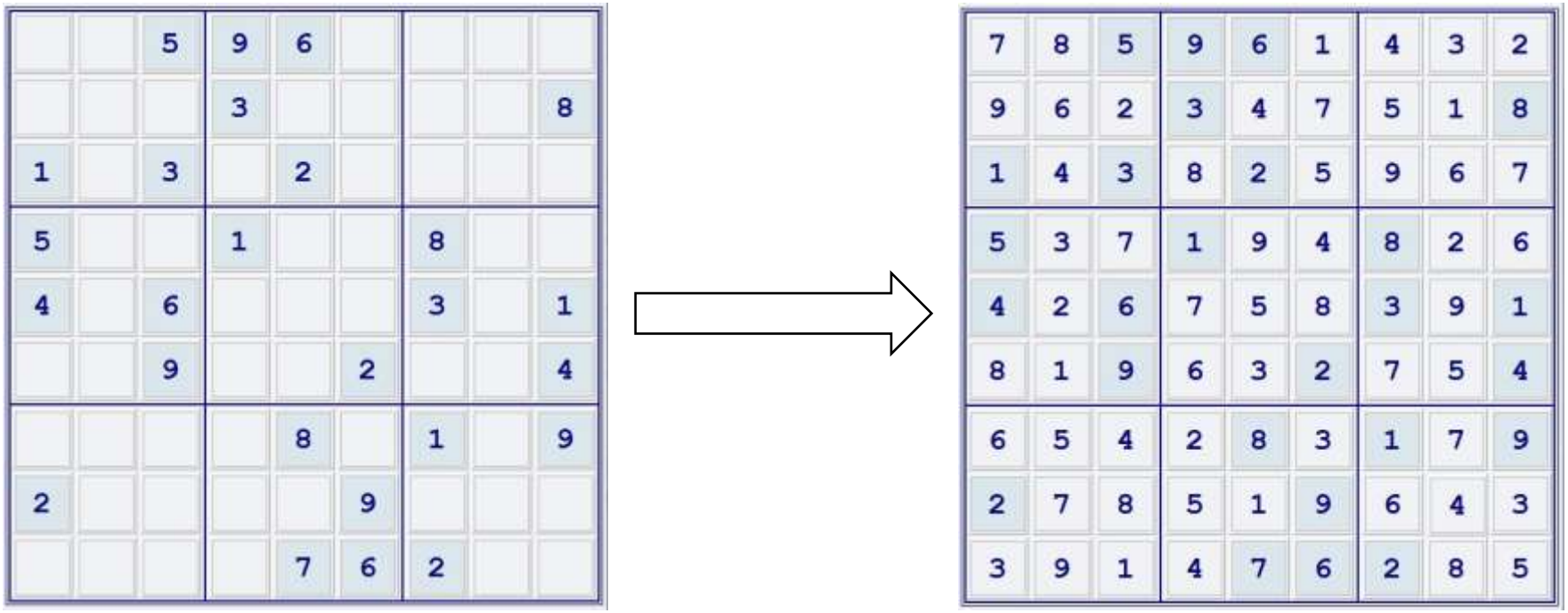
Repräsentation von Situationen (Zuständen)

Beschreiben von zeitlich/räumliche bestimmten Eigenschaften der (realen/angenommenen) Welt.

Beispiele:

- Stellung auf einem Schachbrett,
- Verkabelung eines Computers,
- Medizinischer Laborbefund,
- Steuererklärung (tax report),
- …

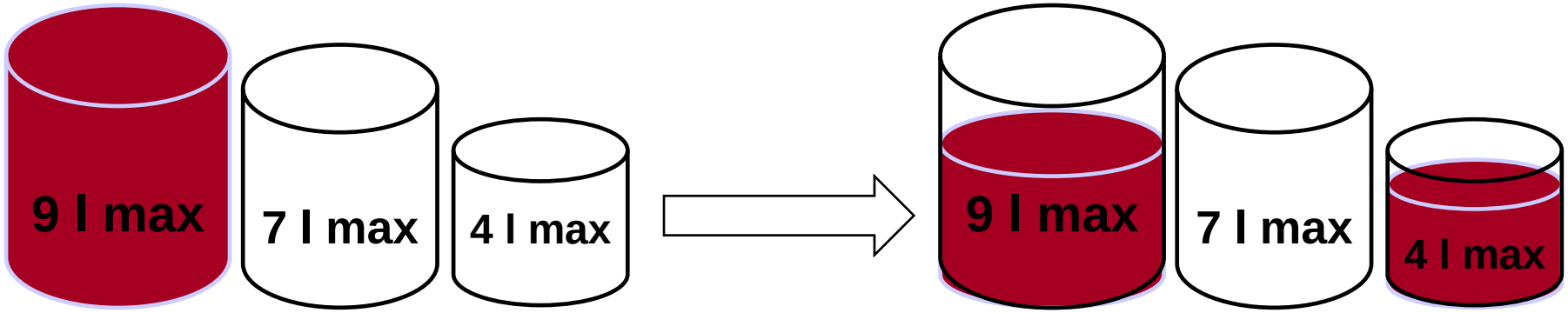
Beispiel für Problemstellung (1)



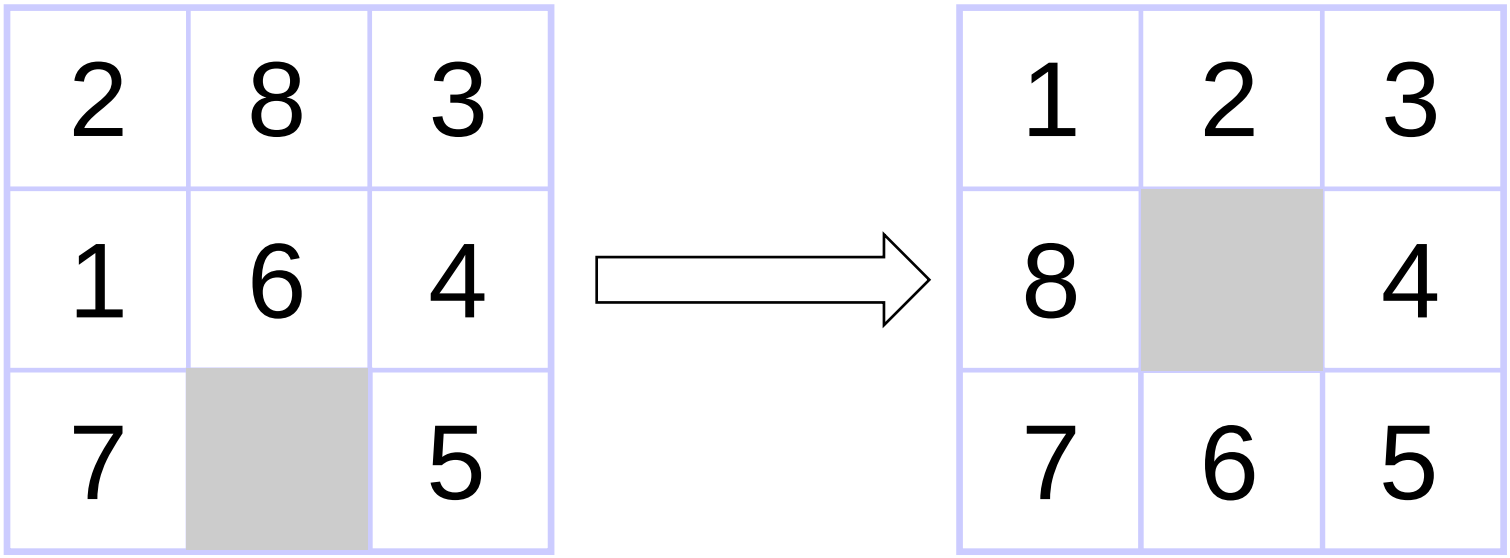
Beispiel für Problemstellung (2)

Ein Weinhändler hat 3 Fässer, 9l, 7l und 4l Inhalt. Auf den Fässern sind keine Markierungen angebracht. Das 9l Fass ist mit Wein gefüllt, die anderen beiden sind leer. Aufgabe: das 9l-Fass soll 6l Wein enthalten und das 4l-Fass 3Liter.

(aus *Boersch, Heinsohn, Socher: „Wissensverarbeitung: Eine Einführung in die Künstliche Intelligenz für Informatiker und Ingenieure.“*)



Beispiel für Problemstellung (3) 8-Puzzle-Problem



Beispiel für Problemstellung (4)

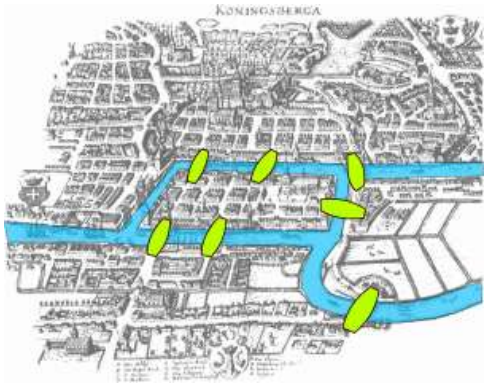
Zahlen-Puzzle

```
   SEND
+  MORE
------
  MONEY
```

```
   DONALD
+  GERALD
--------
   ROBERT
```

```
   TWO
+  TWO
-----
  FOUR
```

Beispiel für Problemstellung (5) optimaler Weg



Königsberger Brückenproblem

Quelle: http://www.brg-landeck.tsn.at/~gerd/pvd/pub/raetsel5.gif

Handlungsreisender (Travelling Salesman Problem)

Repräsentation (1)

Suchprobleme sind gegeben durch:

- Beschreibung eines **Startzustandes**,
- Beschreibung eines **Zielzustandes**,
- Menge von **Operatoren**, die jeweils einen Zustand in einen anderen überführen.

Lösung eines Suchproblems:

- konkreter Zustand, der der Beschreibung eines gewünschten Zielzustandes genügt, oder
- Folge von Operatoren, deren Anwendung vom Start- zum Zielzustand führt (**Pfad** bzw. Weg).

Repräsentation (2)

Wie kann eine Darstellung erfolgen?

Symbolisch

- Diskrete Darstellung durch geeignete Symbolstrukturen
- z.Bsp. logische Formeln

Subsymbolisch

- z.Bsp. Gewichte in Neuronalen Netzen

Analog

- z.Bsp. Graphische Repräsentation

Repräsentation des 8-er Puzzle

Beispiel: 8-er Puzzle

logische Repräsentation:

pos(2,lo),pos(8,mo),pos(3,ro),
pos(1,lm),pos(6,mm),pos(4,rm),
pos(7,lu),pos(h,mu),pos(5,ru)

2	8	3
1	6	4
7		5

symbolische Repräsentation (z.Bsp.: 3x3 Array):

<<2 8 3> <1 6 4> <7 h 5>>

[2,8,3,1,6,4,7,h,5]

[[2,8,3], [1,6,4], [7,h,5]]

analoge Repräsentation (z.Bsp.: als Bitmap)

2	8	3
1	6	4
7		5

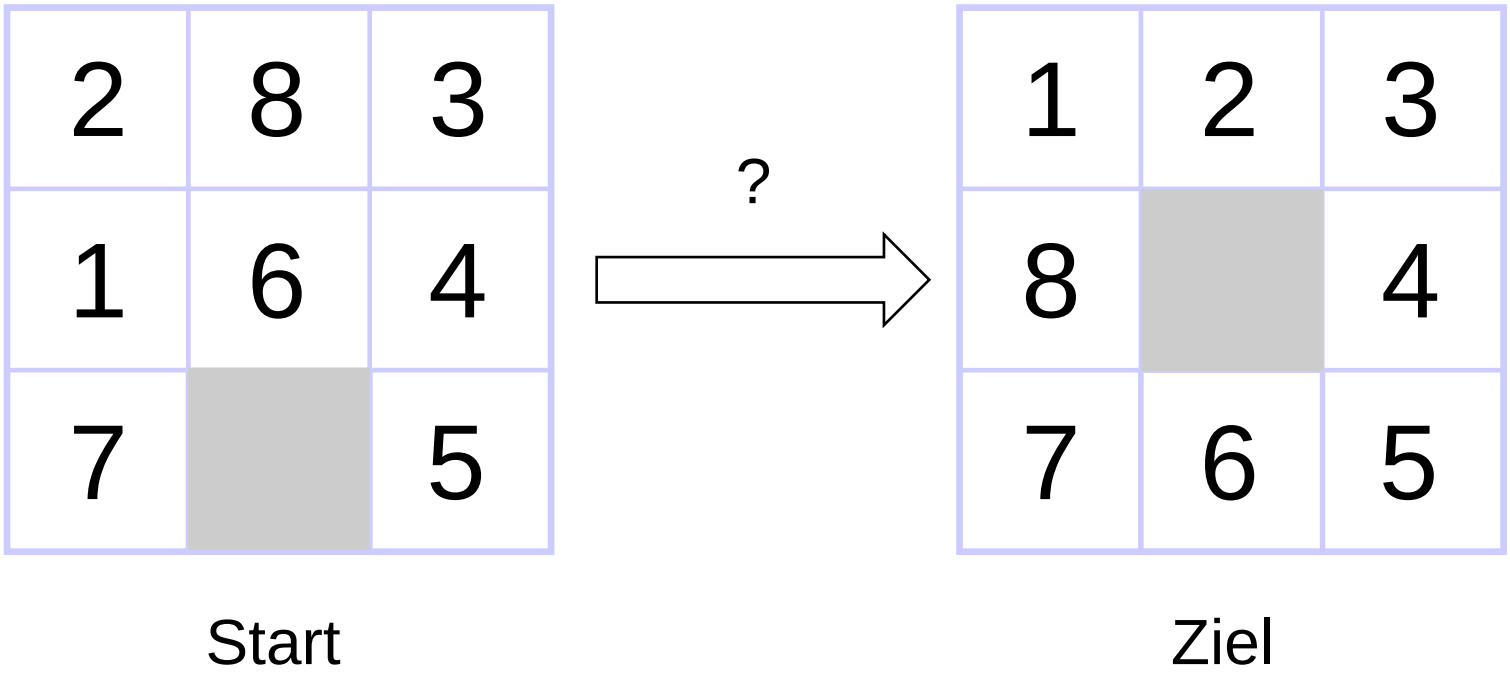
Kriterien zur Wahl der Repräsentation

Vergleich … Schnittstelle eines ADT

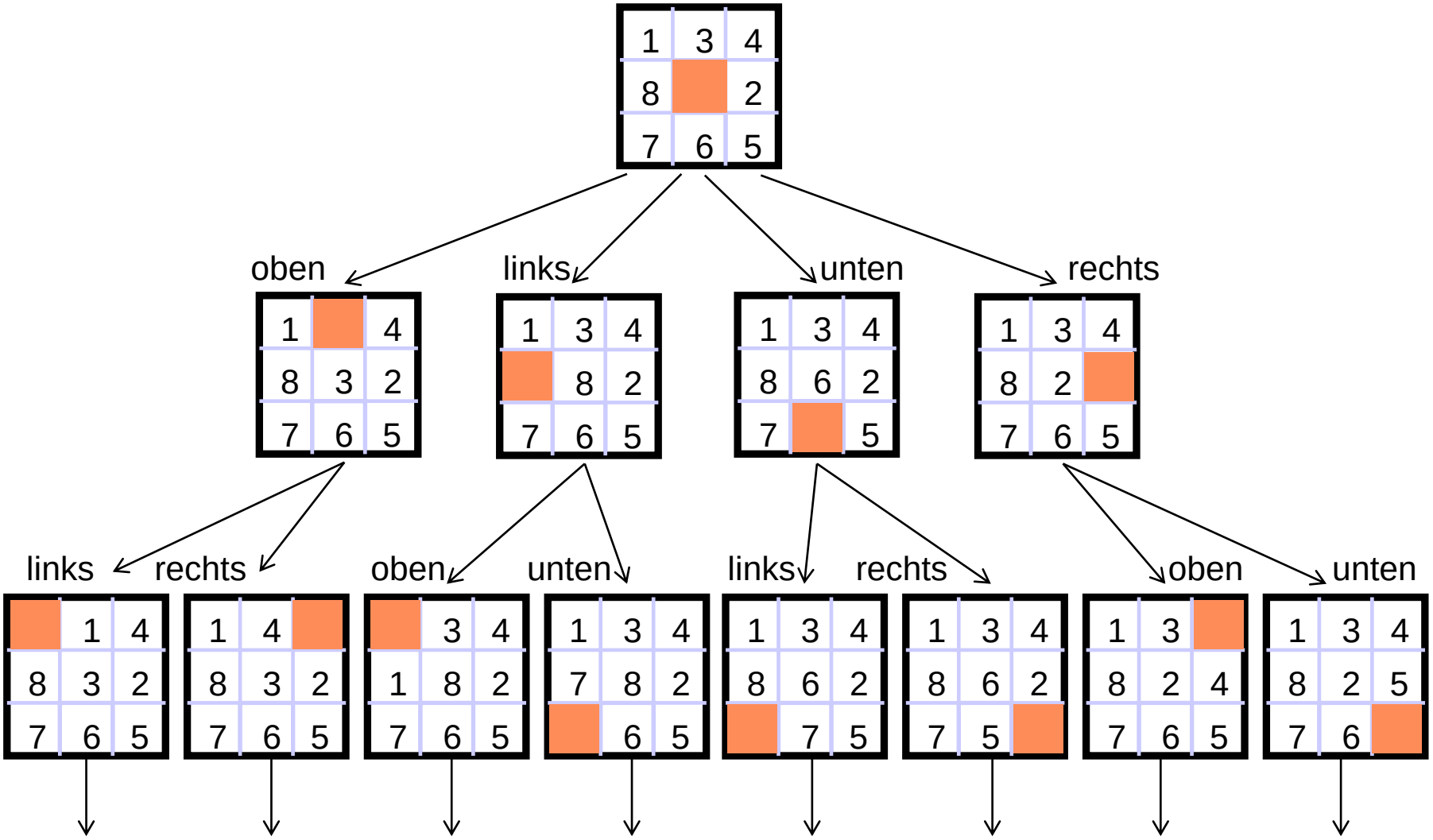
Was will man ein 8-erPuzzle fragen?

- Wo ist das Loch?
  - logisch:
    dasjenige **x** mit pos(h, **x**)
  - symbolisch:
    durch einen geeigneten Algorithmus
  - analog:
    ???
- Welche Züge sind in der aktuellen Stellung möglich?
- Wie sieht das Puzzle nach einem solchen Zug aus?

Problemlösung

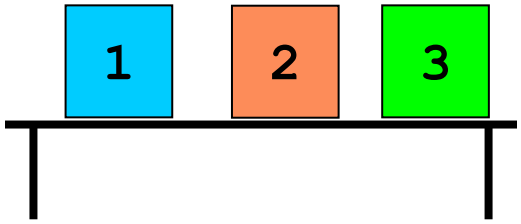


Zustandsübergänge des 8-Puzzle durch Verschieben des leeren Feldes



Übungsaufgabe I

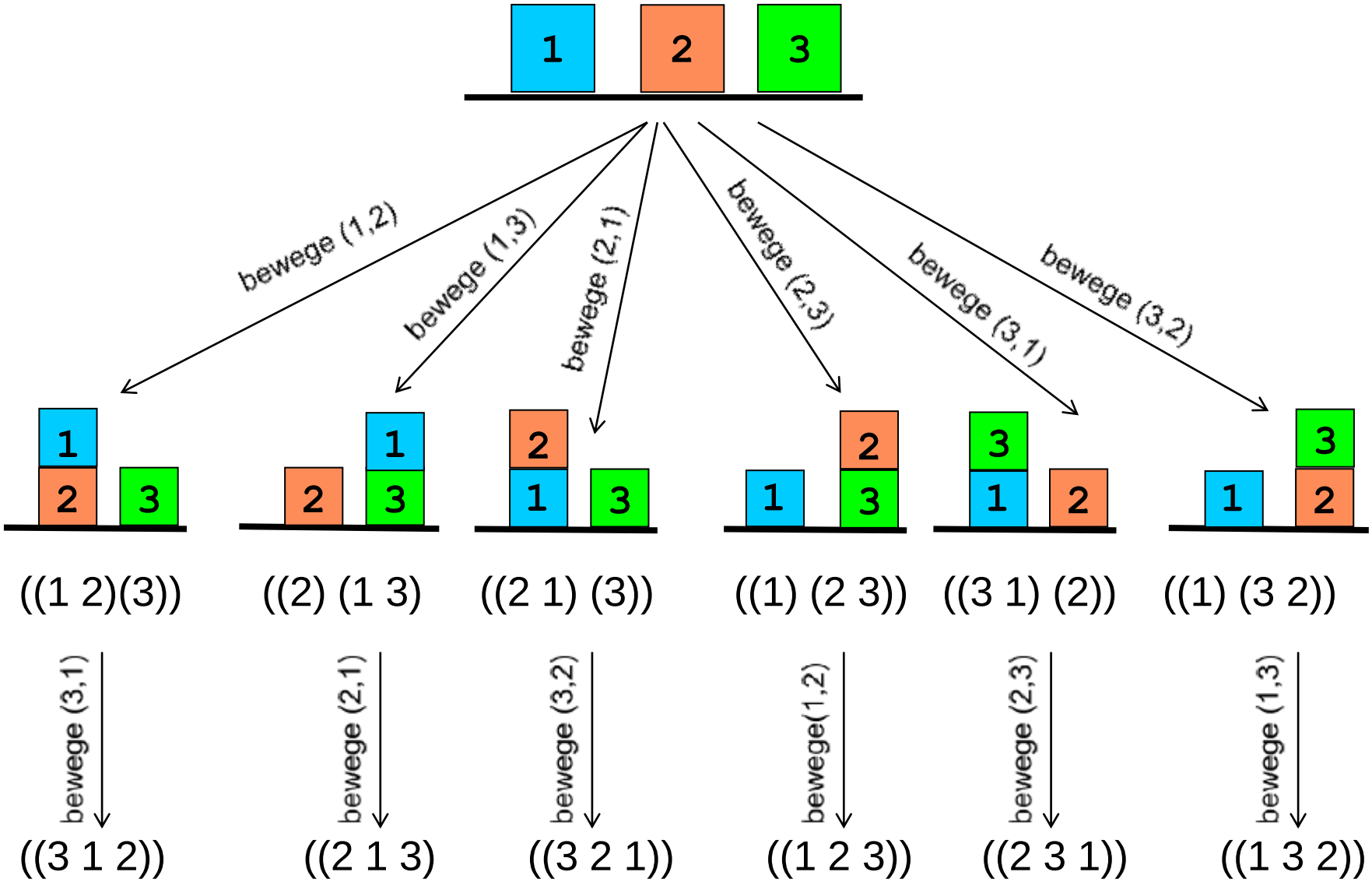
Ermitteln Sie die Zustandsübergänge für die folgende Blocksworld:



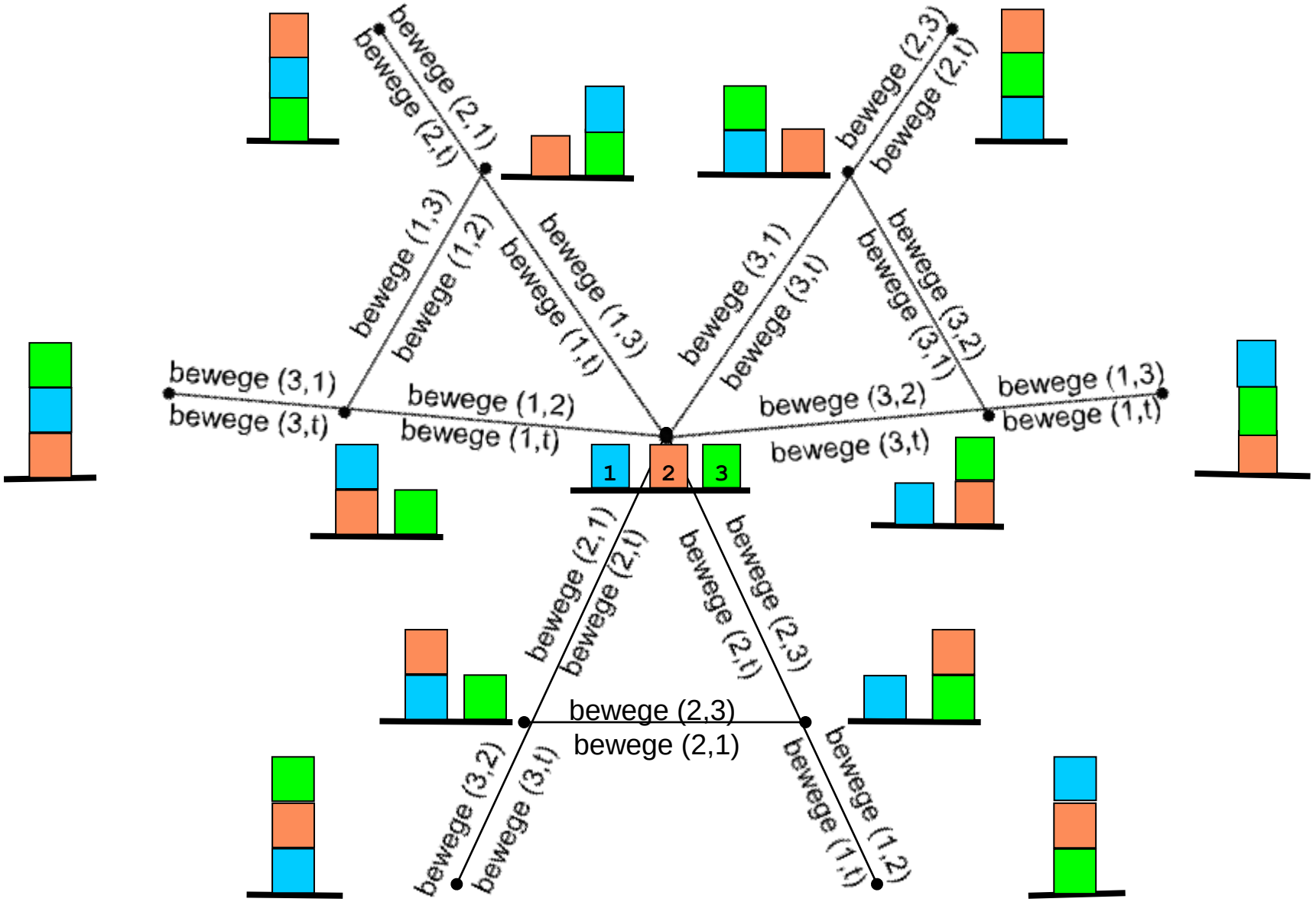
((1) (2) (3))

Übungsaufgabe I

Lösung



Situationsgraph



Suchverfahren (1)

- Symbolstrukturen können für Lösungen von Problemen stehen.
- Systeme können Symbolstrukturen für Alternativen konstruieren.
- Auf Symbolstrukturen für Kandidaten können Tests angewendet werden, um ihre Qualität zu ermitteln.
- Suchprozesse können algorithmisch beschrieben werden.
- Suchprozesse können durch Wissen über das Problem verbessert werden.

Suchverfahren (2)

Unterschiedliche Anforderungen:

- irgendwelche Lösungen finden,
- alle Lösungen finden,
- eine „optimale“ Lösung finden,
- bestätigen, dass keine Lösung existiert.

Ob ein Zielzustand oder ein Pfad („Plan, wie der Zielzustand erreichbar ist“) gesucht wird, ist von untergeordneter Bedeutung, da Zustandsbeschreibungen Pfade umfassen können.

Suchverfahren: typische Aufgabenstellungen

Diagnose:

Suche nach Fehlfunktionen, die beobachtete Symptome erklären und mit bisherigem Wissen kompatibel sind.

Interpretation:

Suche nach Fakten, die beobachtete Daten erklären und mit dem Vorwissen kompatibel sind.

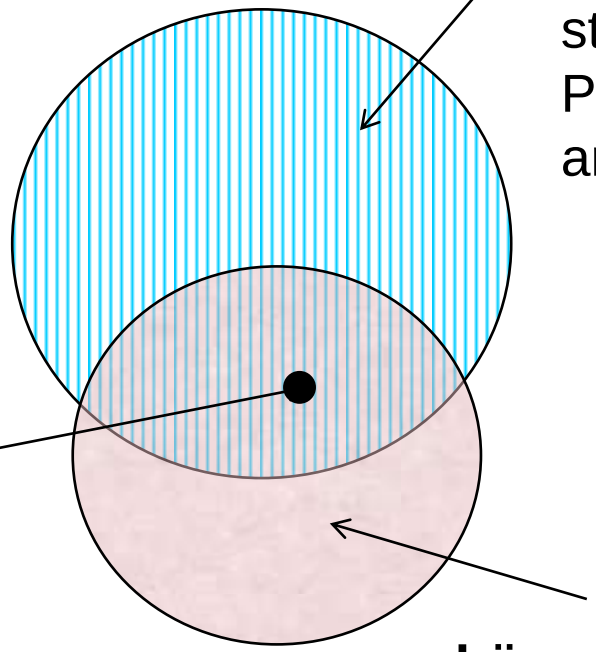
Konfiguration:

Suche nach einem durchführbaren Arrangement von Komponenten, die konsistent mit den Spezifikationen, teilweisen Konfigurationen und Einschränkungen der Komponenten sind.

Scheduling und Planung:

Suche nach der Reihenfolge von Aktionen, die zu einem gegebenen Ziel führen.

Such- und Lösungsraum



Suchraum (search space):
Die Menge der Symbol-strukturen, die von einem Programm als Kandidaten angesehen werden.

Lösung für ein spezielles Problem

Lösungsraum (solution space):
Die Menge der Lösungen eines Problems, unabhängig davon, ob ein Programm alle diese Lösungen erreichen kann.

Definition Zustandsraum / Suchraum

Ein Zustandsraum ist ein 4-Tupel **(N, A, S, G)**:

N: Menge der Knoten des Zustandsraumgraphen: Zustände

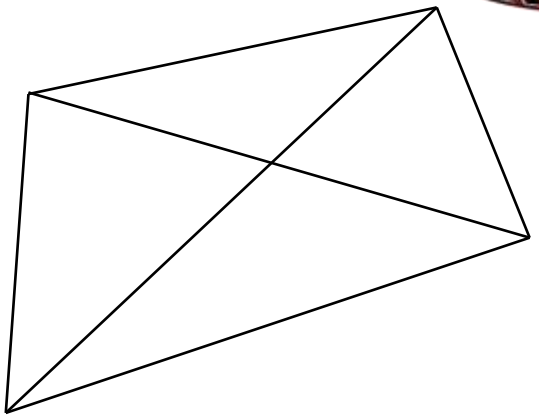
A: Menge der Kanten zwischen den Zuständen:
Zustandsübergänge, Schritte des Problemlösungsprozesses

S: nicht-leere Teilmenge von N: Startzustand bzw. Startzustände

G: nicht-leere Teilmenge von N: Zielzustand bzw. Zielzustände

Der Zustandsraum ist ein gerichteter Graph (ist der Graph baumartig, ist der Graph zyklenfrei, …).

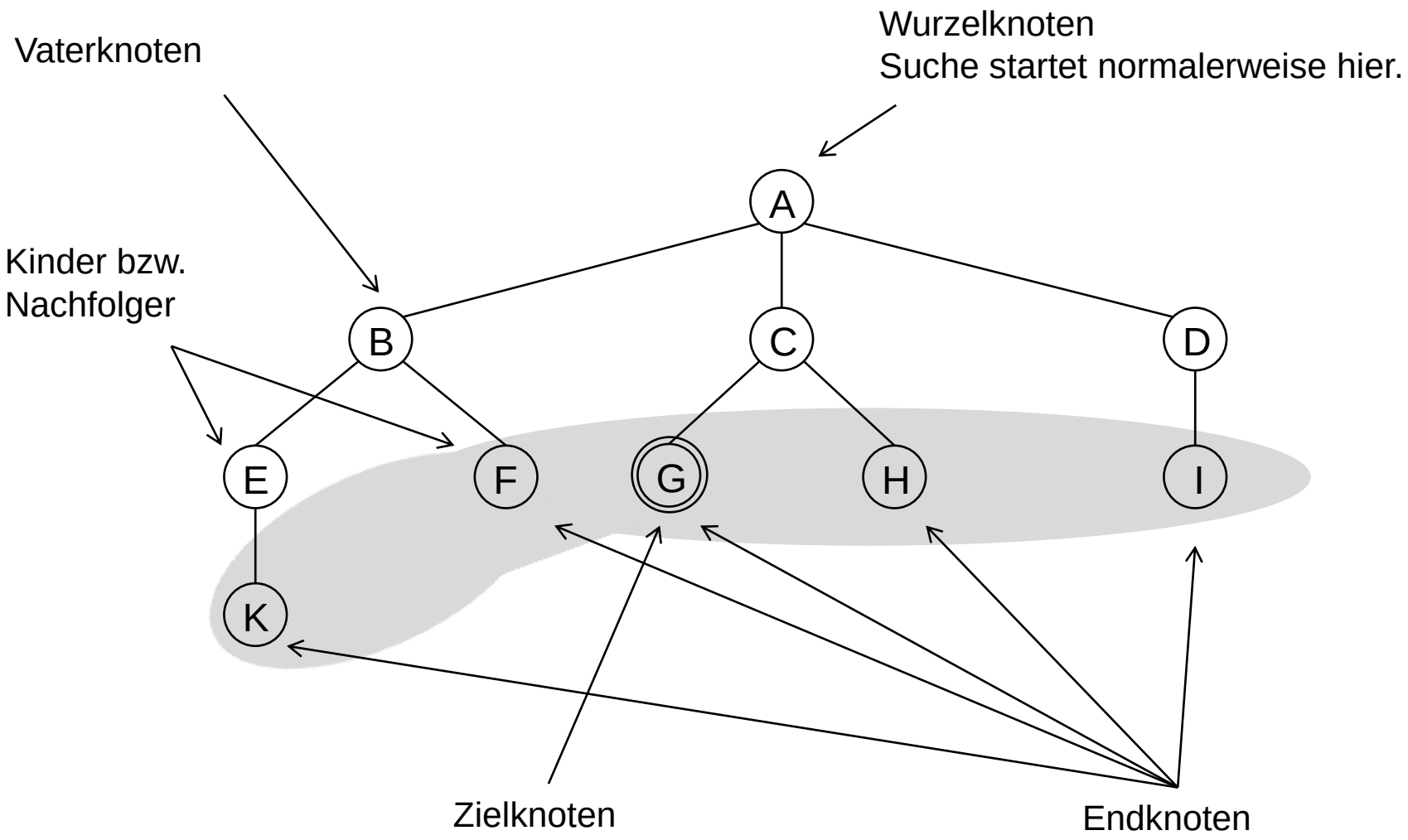
Erkundung des Suchraumes



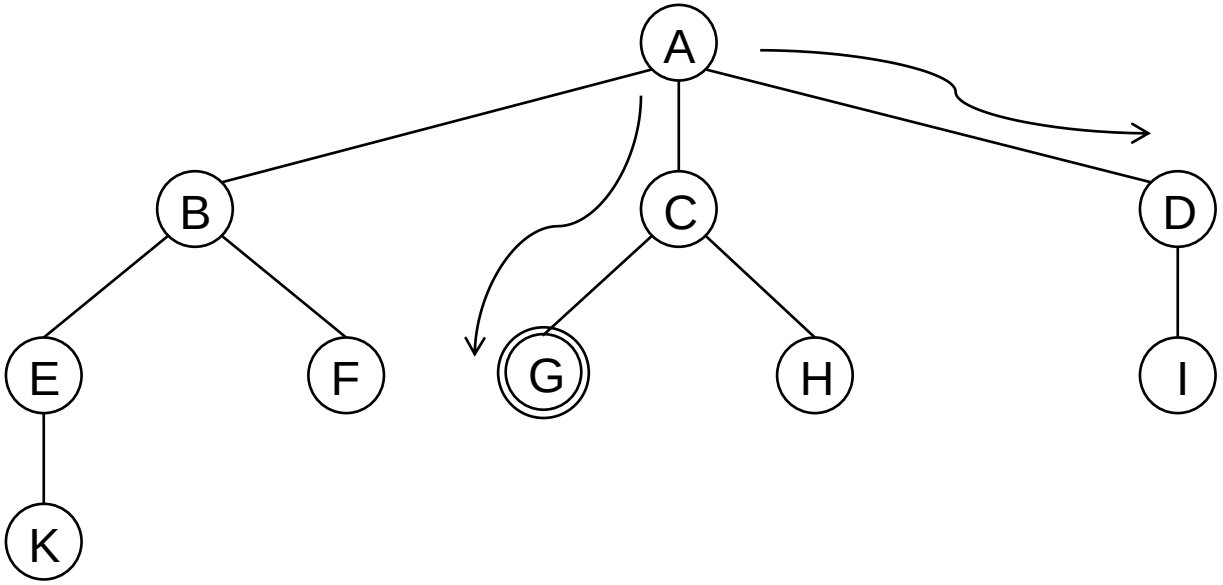
Haus Baum Auto Haus Schatz
Haus Baum Auto Schatz
Haus Auto Schatz
Haus Schatz

Möglichkeit unendlicher Pfade durch im-Kreis-laufen, solange man keine Möglichkeit hat, vorbesuchte Knoten zu erkennen.

Bäume als Suchraum

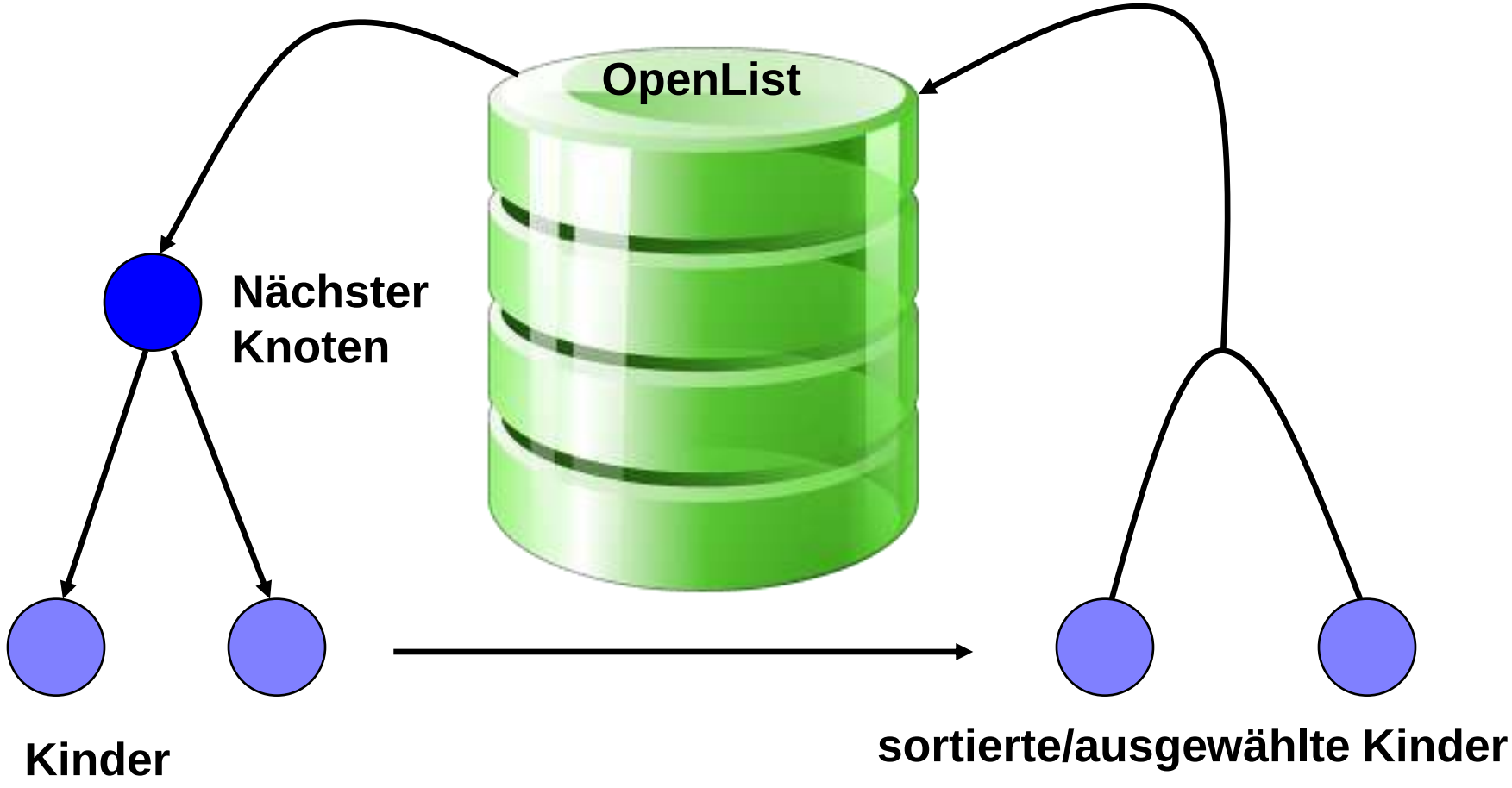


Generieren von Lösungs-Kandidaten



Suche durch systematische Generierung von Kandidaten (expandieren), d.h. zu einem Knoten werden alle Folgeknoten bestimmt, die in einer OpenList verwaltet werden.

Allgemeiner Suchalgorithmus



Eigenschaften von Suchprozessen

Erschöpfend (exhaustive):

Der Prozess liefert in jedem Fall alle Lösungen. Geschieht meistens (!) durch Betrachten des gesamten Suchraumes.

Vollständig (complete):

Der Prozess findet wenigstens eine Lösung, falls es eine gibt.

Befriedigend (satisfying):

Der Prozess stoppt bei der ersten Lösung.

Optimal (optimizing):

Der Prozess stoppt bei der besten Lösung (kürzester Weg).

Ressourcen-beschränkt (resource limited):

Der Prozess stoppt, wenn die vorgegebenen Ressourcen (Zeit, Speicher, …) aufgebraucht sind.

Arten von Suchverfahren

Anytime-Algorithmen:

Liefern zu jedem Zeitpunkt eine Lösung. Die Qualität der Lösungen steigt mit wachsender Zeit.

Uninformiert (blind, uninformed):

Systematischer Aufbau des Suchraumes.

Gesteuert (directed):

Heuristiken steuern den Aufbau und die Untersuchung des Suchraumes.

Hierarchisch (hierarchical):

Abstrakte Ziele und Lösungen strukturieren den Aufbau des Suchraumes.

Uninformierte Suche

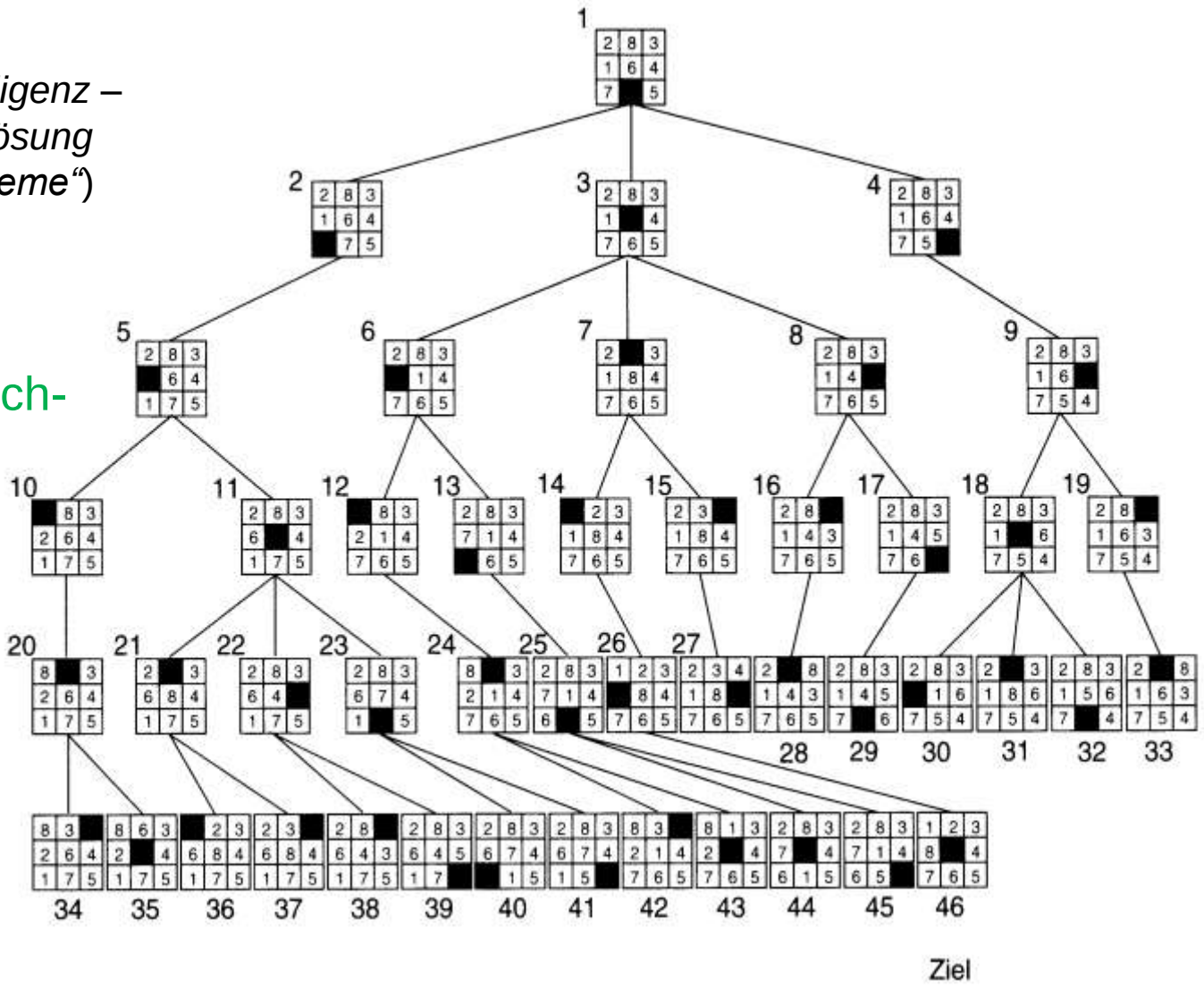
Uninformierte (blinde) Suche:

- Generieren und Prüfen von Lösungs-Kandidaten ohne Verwendung von zusätzlichem Domänenwissen.
- Ausgehend von der Wurzel kann man zu einem Knoten sukzessive die Nachfolgeknoten berechnen.
  Knoten werden in einer Liste (OpenList) vermerkt.
- Nachfolgend wird mit den Nachfolgeknoten weitergearbeitet, bis das (ein) Ziel gefunden wurde bzw. die Suche erfolglos war.
- zwei grundlegende Möglichkeiten der Suche:
  - Breitensuche und
  - Tiefensuche.

Uninformierte Suche

(aus *Luger:*
„Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“)

Beispiel:
8-Puzzle
Gesamter Such-raum bis zur am nächsten gelegenen Lösung

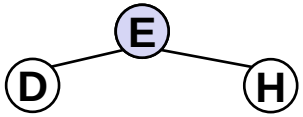


Breitensuche

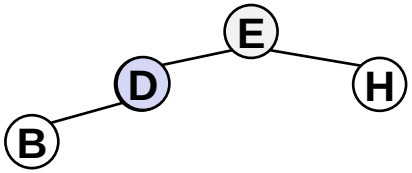
Schritt	OpenList
1	E
2	D, H
3	H, B
4	B, F
5	F, A, C
6	A, C, G
7	C, G
8	G

OpenList wird im FIFO- Prinzip abgearbeitet (Queue).

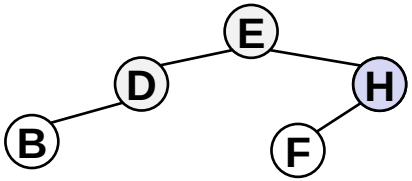
1)



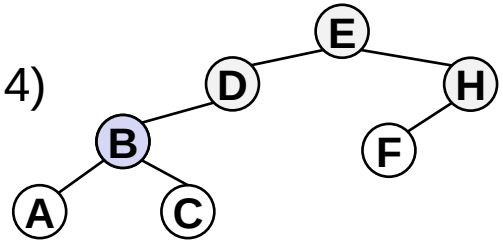
2)



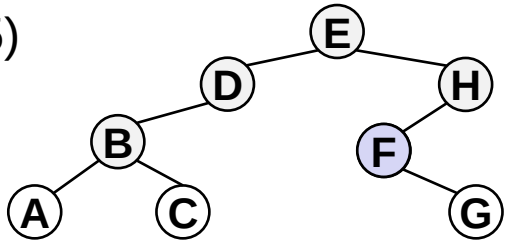
3)



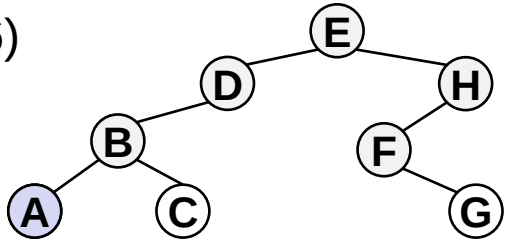
4)



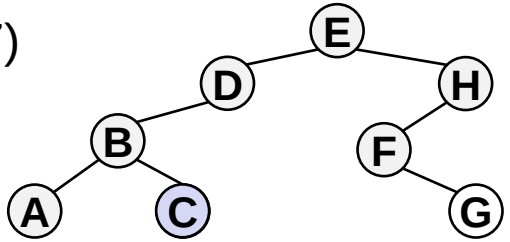
5)



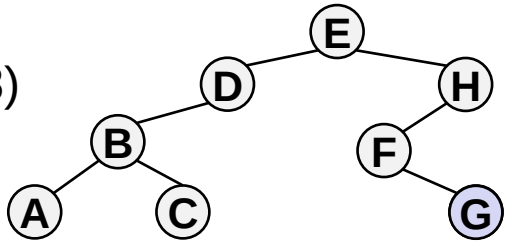
6)



7)



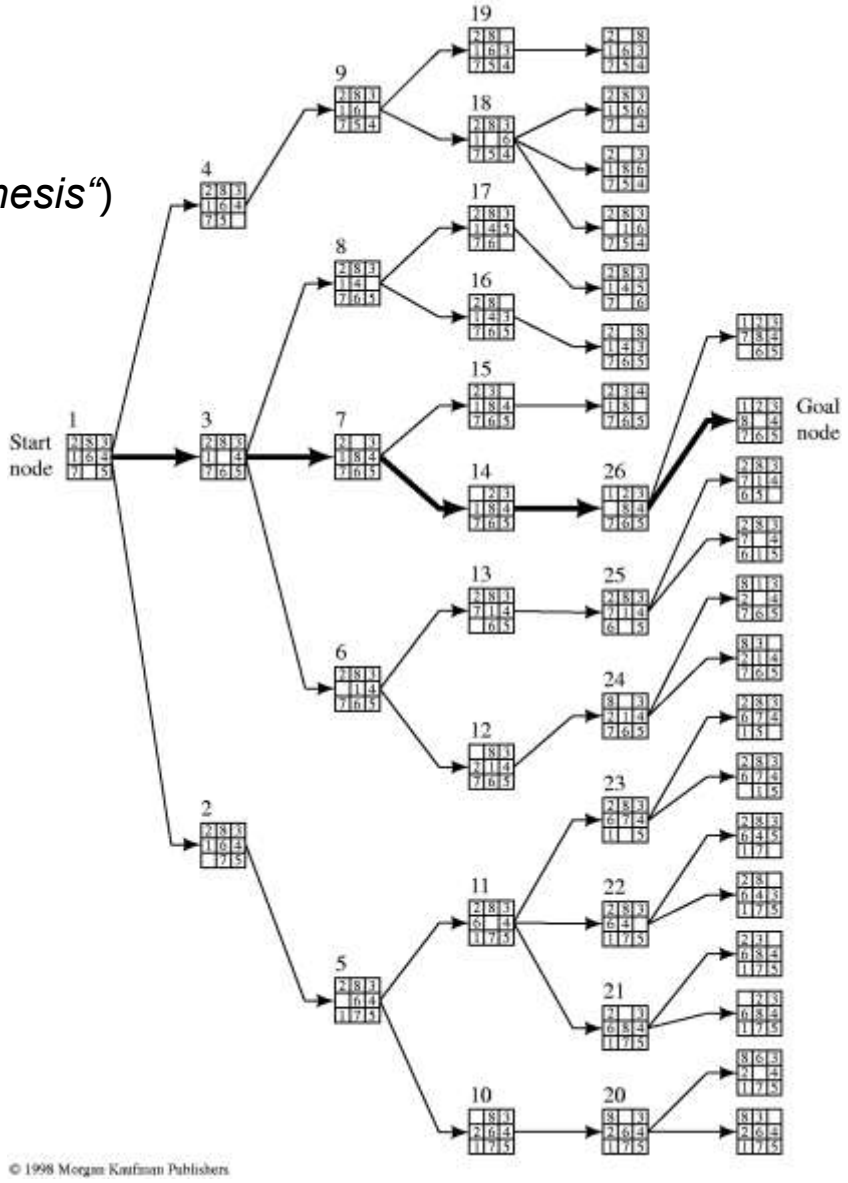
8)



(aus *Nilsson:*
„Artificial Intelligence – A New Synthesis“)

Beispiel: 8-Puzzle

Reihenfolge expandierter Knoten, bis Lösung gefunden wurde.



Bewertung Breitensuche

- Wenn Lösung existiert, wird sie gefunden,
- Kürzester Weg wird gefunden (Optimalität),
- Hoher Ressourcenbedarf:
  - Abspeichern der zahlreichen offenen Zustände.
  - Problematisch bei hoher Verzweigungsrate.
  - Problematisch bei hoher Tiefe des Suchraums.
  - **Daher häufig nicht einsetzbar!**

Komplexität Breitensuche

Anzahl erzeugter Knoten bei Verzweigungsgrad **g** und Tiefe des Zielknotens **t**:

$$g + g^2 + g^3 + \ldots + (g^{t+1} - g) = O(g^{t+1})$$

- Worst Case Annahme:
  Zielknoten befindet sich unten rechts.
- Startknoten wird nicht erzeugt.
- Startknoten auf Ebene 0.

Entspricht Speicher- und Zeitbedarf.

Zeit- und Speicherbedarf der Breitensuche

(nach *Russel, Norvig: „Künstliche Intelligenz – Ein moderner Ansatz“, 2012, korrigiert*)

Annahmen:

- Verzweigungrate 10
- Zeitbedarf 1.000.000 Knoten / Sekunde
- Speicherbedarf 1000 Byte / Knoten

Tiefe	Knoten	Zeit	Speicher
2	1100	1,1 ms	1 Megabyte
4	111100	111 ms	100 Megabyte
6	10^7	11 Sekunden	10 Gigabyte
8	10^9	19 Minuten	1 Terabyte
10	10^{11}	31 Stunden	100 Terabyte
12	10^{13}	128 Tage	10 Petabyte
14	10^{15}	35 Jahre	1 Exabyte
16	10^{17}	3490 Jahre	100 Exabyte

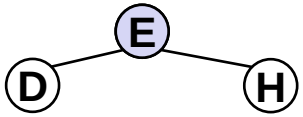
Tiefensuche

(Backtracking Suche)

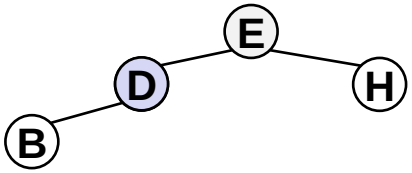
Schritt	OpenList
1	E
2	D, H
3	B, H
4	A, C, H
5	C, H
6	H
7	F
8	G

OpenList wird im LIFO- Prinzip abgearbeitet (Stack).

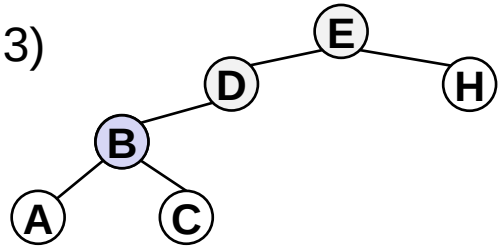
1)



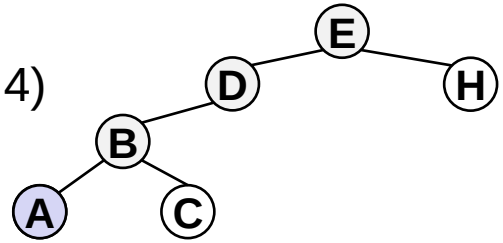
2)



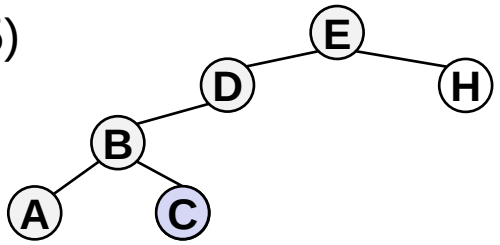
3)



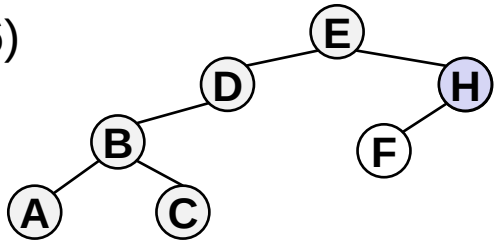
4)



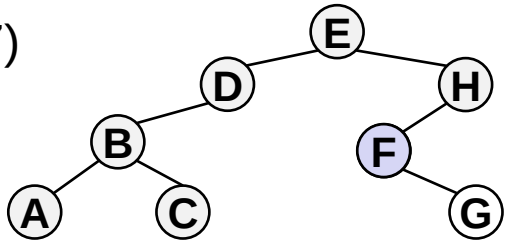
5)



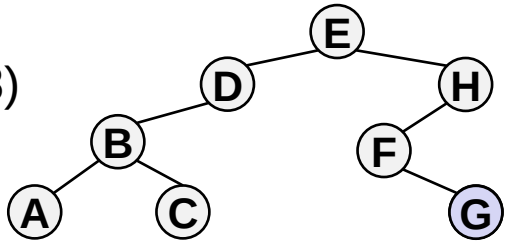
6)



7)

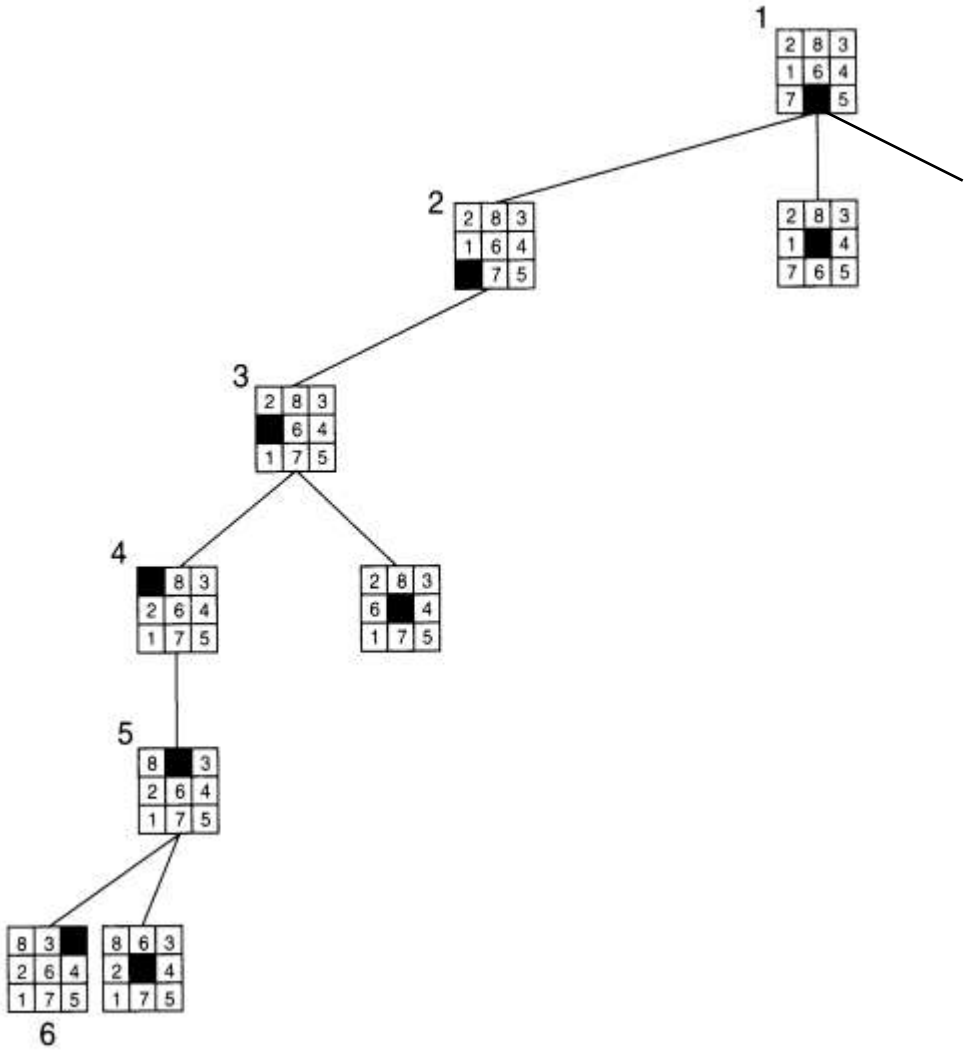


8)



(aus Luger:
„Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“)

Beispiel: 8-Puzzle



Bewertung Tiefensuche

- Geringer Ressourcenbedarf.
- Lösung kann verfehlt werden.
- Speicherbedarf: **$g * t_{max}$**
- Zeitbedarf:
  - $O(g^{t_{max}})$, da im schlimmsten Fall alle Knoten bis zur Tiefe t_{max} expandiert werden

Breitensuche versus Tiefensuche

Beide Verfahren würden das 8-Puzzle lösen.

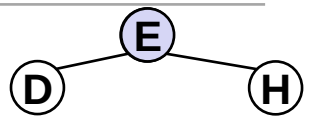
- Keine mögliche Abschätzung des Zeit- und Speicherbedarfs der Tiefensuche.
- Breitensuche liefert Lösung, die die wenigsten Bewegungen erfordert.
- Absolutes Maximum an Positionen beim 8-Puzzle: 9! = 362.880
  (für ein 15-Puzzle: 15!=1.307.674.368.000 …)
- Breitensuche: Länge l der Lösung ca. 3^l untersuchte Positionen.

Tiefensuche mit Tiefenbeschränkung

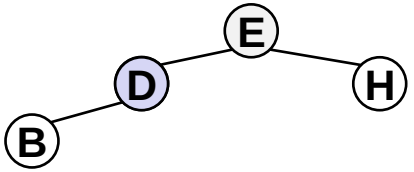
Tiefensuche mit Tiefenbeschränkung

Beispiel für Tiefenbeschränkung=2

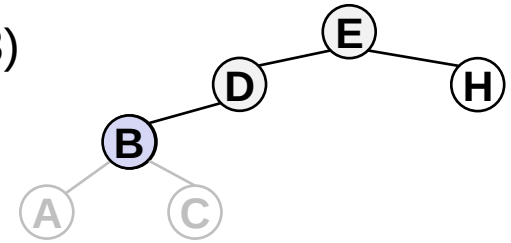
1)



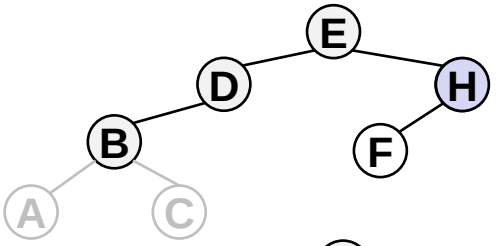
2)



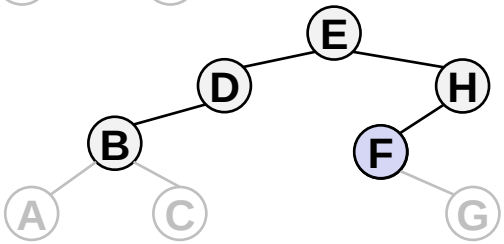
3)



4)



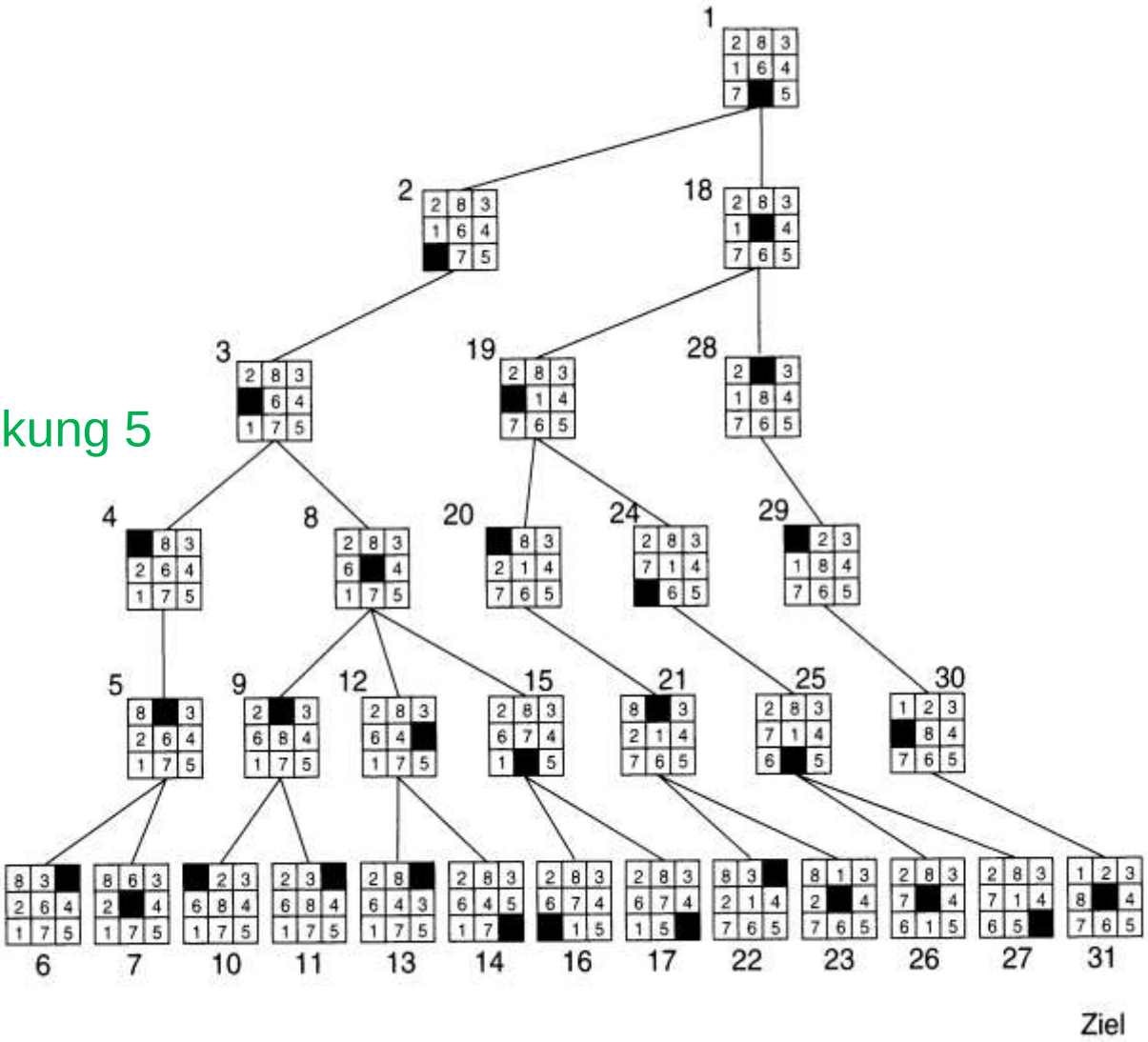
5)



Tiefensuche mit Tiefenbeschränkung

(aus Luger:
„Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“)

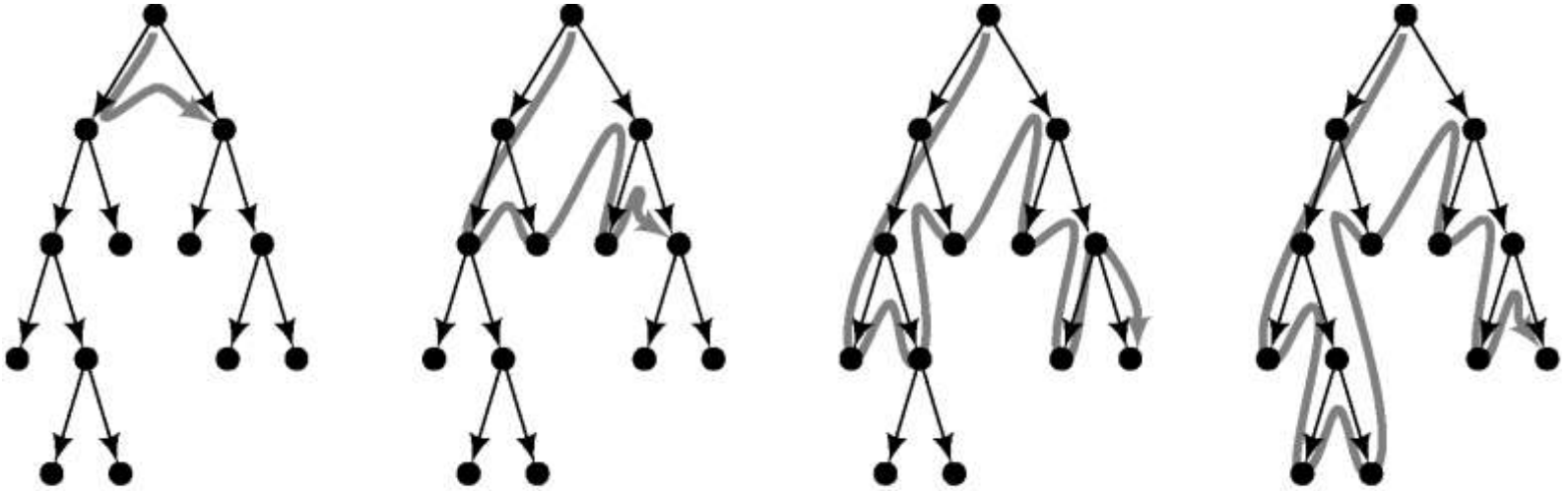
Beispiel:
8-Puzzle
mit Tiefenbeschränkung 5



Iterative Tiefensuche

(aus *Nilsson: „Artificial Intelligence – A New Synthesis“*)

Schrittweise / Iterative Vertiefung
(IDS: iterative deepening search)

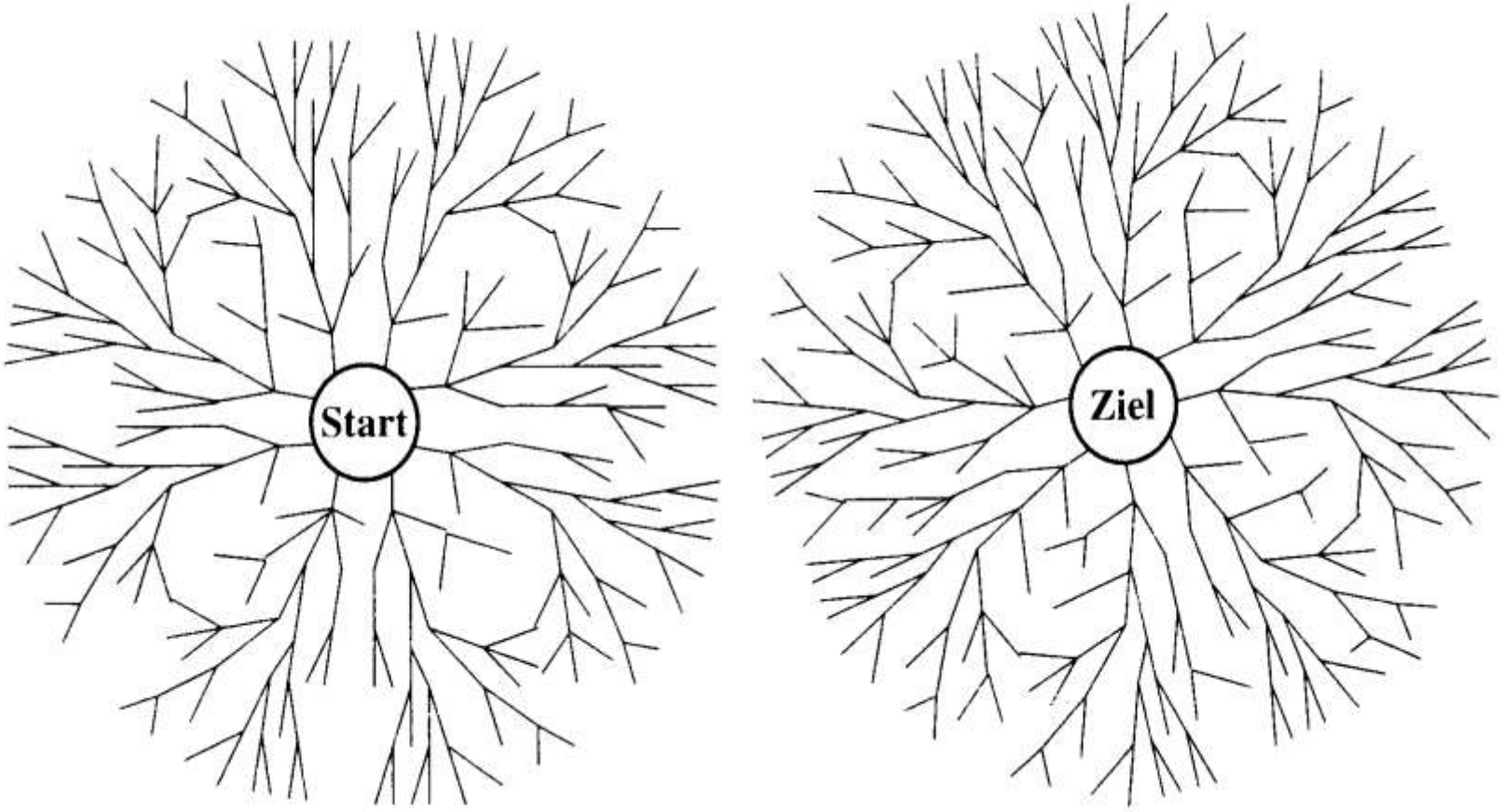


Bewertung Iterative Vertiefung

- Wiederholungen schlagen kaum zu Buche.
- Anzahl erzeugter Zustände:
  $t * g + (t-1) * g^2 + (t-2) * g^3 + \ldots + 1 * g^t = O(g^t)$
  - betrifft Zeitbedarf.
- Speicherbedarf wie Standard-Tiefensuche.
- Merkmale von Tiefen- und Breitensuche kombiniert
  - geringer Speicherbedarf,
  - Lösung wird gefunden.

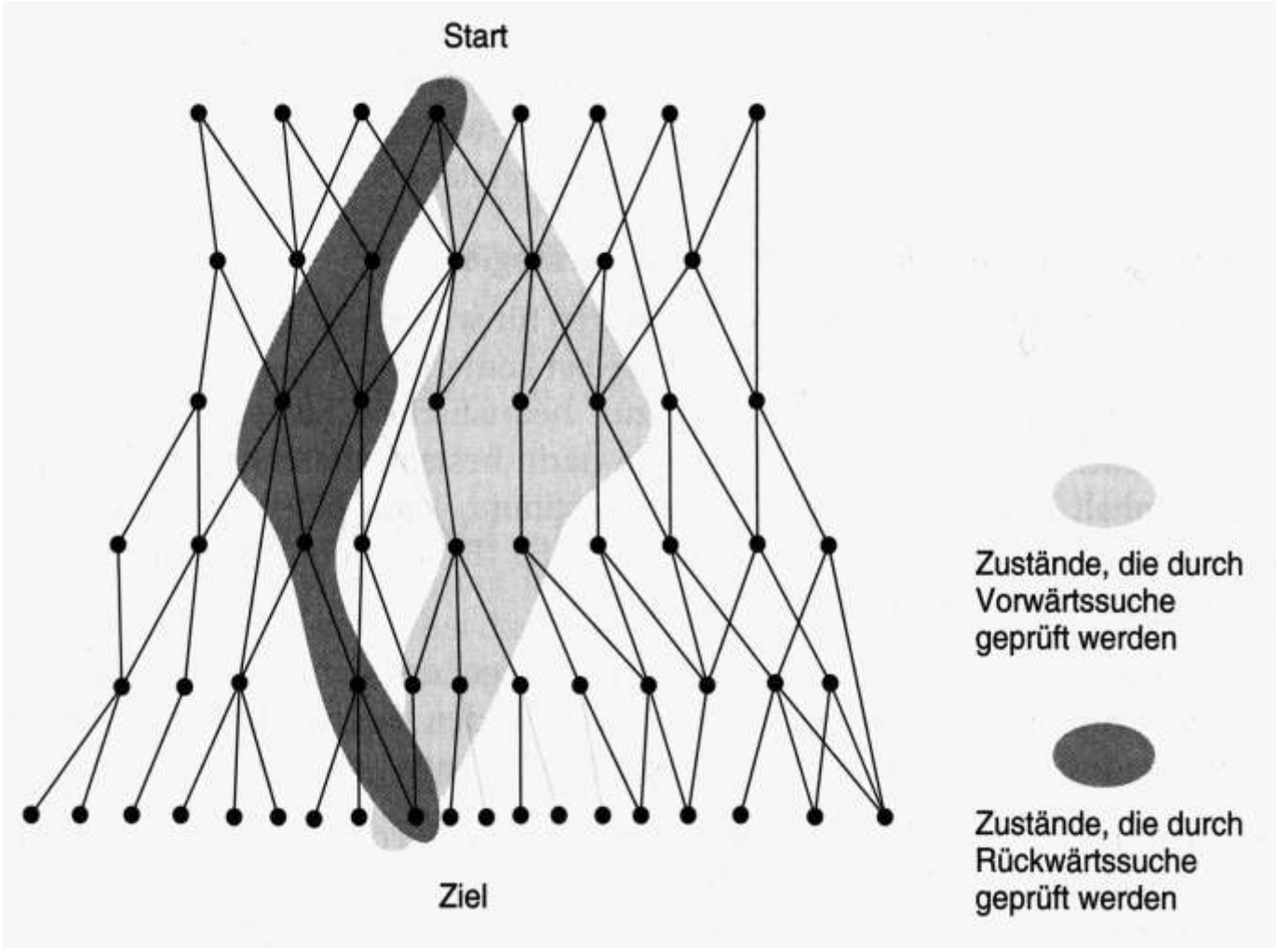
Bidirektionale Suche

(aus *Russel, Norvig: „Künstliche Intelligenz – Ein moderner Ansatz“*)



Bidirektionale Tiefensuche

(aus *Luger: „Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“*)



Bewertung bidirektionale Suche

- Vollständig und kürzester Weg nur für Breitensuche
- Zahl der erzeugten Zustände:
  $2*g + 2*g^2 + 2*g^3 + \ldots + 2*g^{t/2} = O(g^{t/2})$
    - betrifft Speicher- und Zeitbedarf

- Nicht immer anwendbar:
    - Schritte rückwärts nicht möglich.
    - Mehrere Zielzustände.
    - Gesucht ist nicht ein Lösungspfad, sondern die konkrete Ausprägung des Zielzustands.

Vergleich uninformierter Suchstrategien

	Breiten-suche	Tiefen-suche	Be-schränkte Tiefen-suche	Iterative Ver-tiefung	Bidirek-tionale Suche
Vollständig	ja	nein	nein	ja	ja
Kürzester Weg	ja	nein	nein	ja	ja
Speicher-bedarf	$O(g^{t+1})$	$O(g*t_{max})$	$O(g*t_{grenz})$	$O(g*t)$	$O(g^{t/2})$
Zeitbedarf	$O(g^{t+1})$	$O(g^{t_{max}})$	$O(g^{t_{grenz}})$	$O(g^{t})$	$O(g^{t/2})$

… Eigenschaften, Vor- und Nachteile und Arbeitsweise der vorgestellten Suchalgorithmen erläutern und sie untereinander vergleichen können.

Informierte Suche

- Bisher:
  Wissen über Qualität der Zustände nicht genutzt.
- Sicher bekannt:
  bisherige Kosten
- Wünschenswert:
  Information über Restkosten
  - nur Schätzung möglich,
  - Heuristiken.
- Schwierigkeit:
  beste Heuristik finden

ευρισκειν (gr.): (er)finden, entdecken

erprobte, aber nicht notwendigerweise optimale Methoden zur Vereinfachung von Problemen.

Einfache Bewertungsfunktionen (f(n)):

g: Bisherige Kosten

h: geschätzte Kosten der noch nötigen Schritte

3 Möglichkeiten:

f(n) = g(n)

f(n) = h(n)

f(n) = g(n)+h(n)

Nearest Neighbour

f(n)=g(n)

Nächster Nachbar – Nearest Neighbour (Heuristic)

- Voraussetzung für Nearest Neighbour (NNH) ist das Aufwandsmaß g für die Kosten der Zustandsübergänge.
- Beispiel: Travelling Salesman Problem:
  Auf einer Rundreise sollen n Städte besucht werden (, wobei ein bestimmter Faktor zu minimieren ist (z.Bsp. die Reisezeit)).

n! Kombinationsmöglichkeiten!!

- NNH:
  (1) Beliebige Start-Stadt wählen.
  (2) Wähle aus den noch nicht besuchten Städten die zur aktuellen Stadt am nächsten gelegene Stadt.
  (3) Falls noch nicht alle Städte besucht wurden, gehe zu (2).

Nearest Neighbour

- Kein Zurückgehen (Backtracking), um Alternativlösungen zu suchen.
- Demzufolge unvollständig und i.a. nicht optimal.
- Zeitaufwand nur $O(n^2)$.
- Nur bei Vorliegen von genauem Abstandmaß g sinnvoll.

Beispiel:

- Berlin, Dresden, Hamburg, Hannover

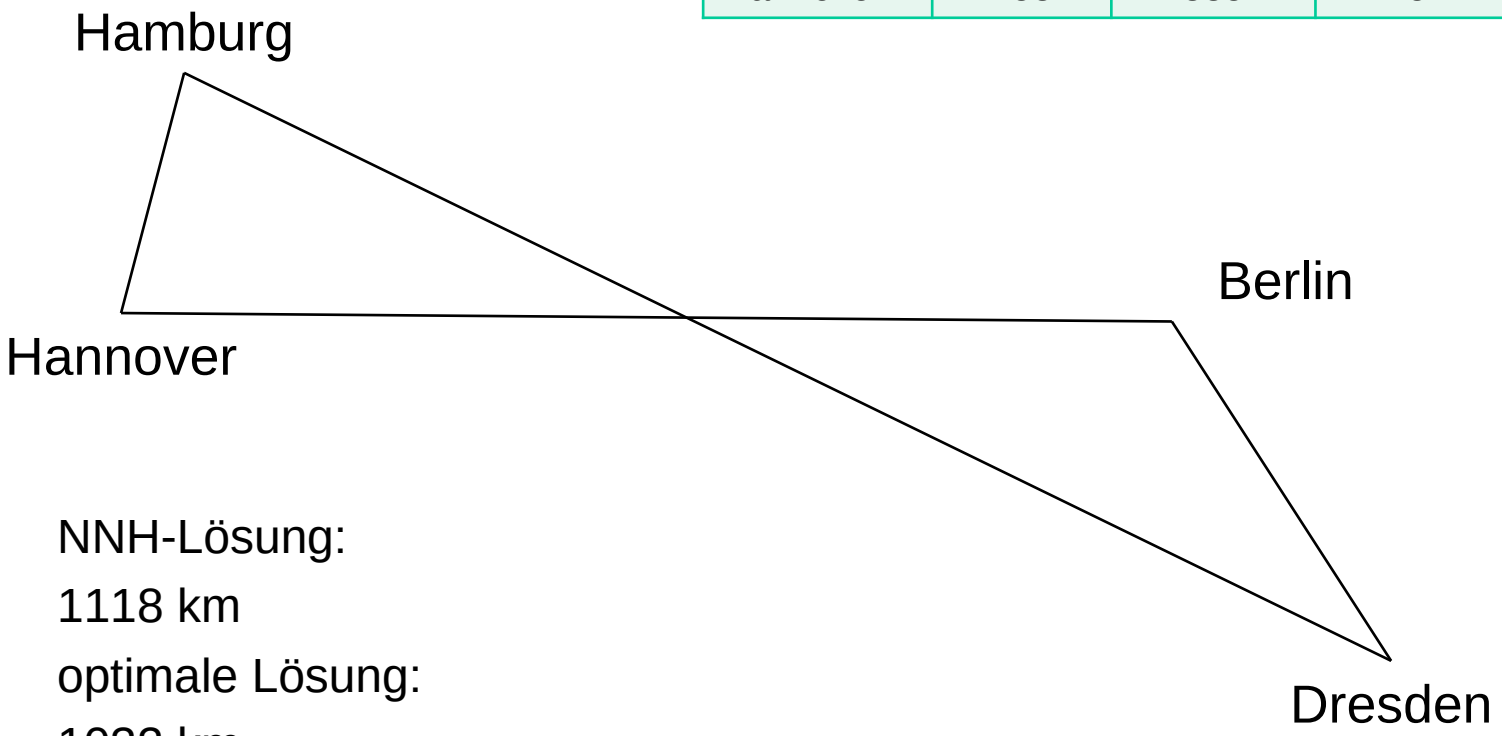
Kilometer	Berlin	Dresden	Hamburg	Hannover
Berlin	0	214	279	258
Dresden	214	0	492	385
Hamburg	279	492	0	154
Hannover	258	385	154	0

Nearest Neighbour

Kilometer	Berlin	Dresden	Hamburg	Hannover
Berlin	0	214	279	258
Dresden	214	0	492	385
Hamburg	279	492	0	154
Hannover	258	385	154	0

Beispiel (Fortsetzung):

Wir starten in Hamburg.



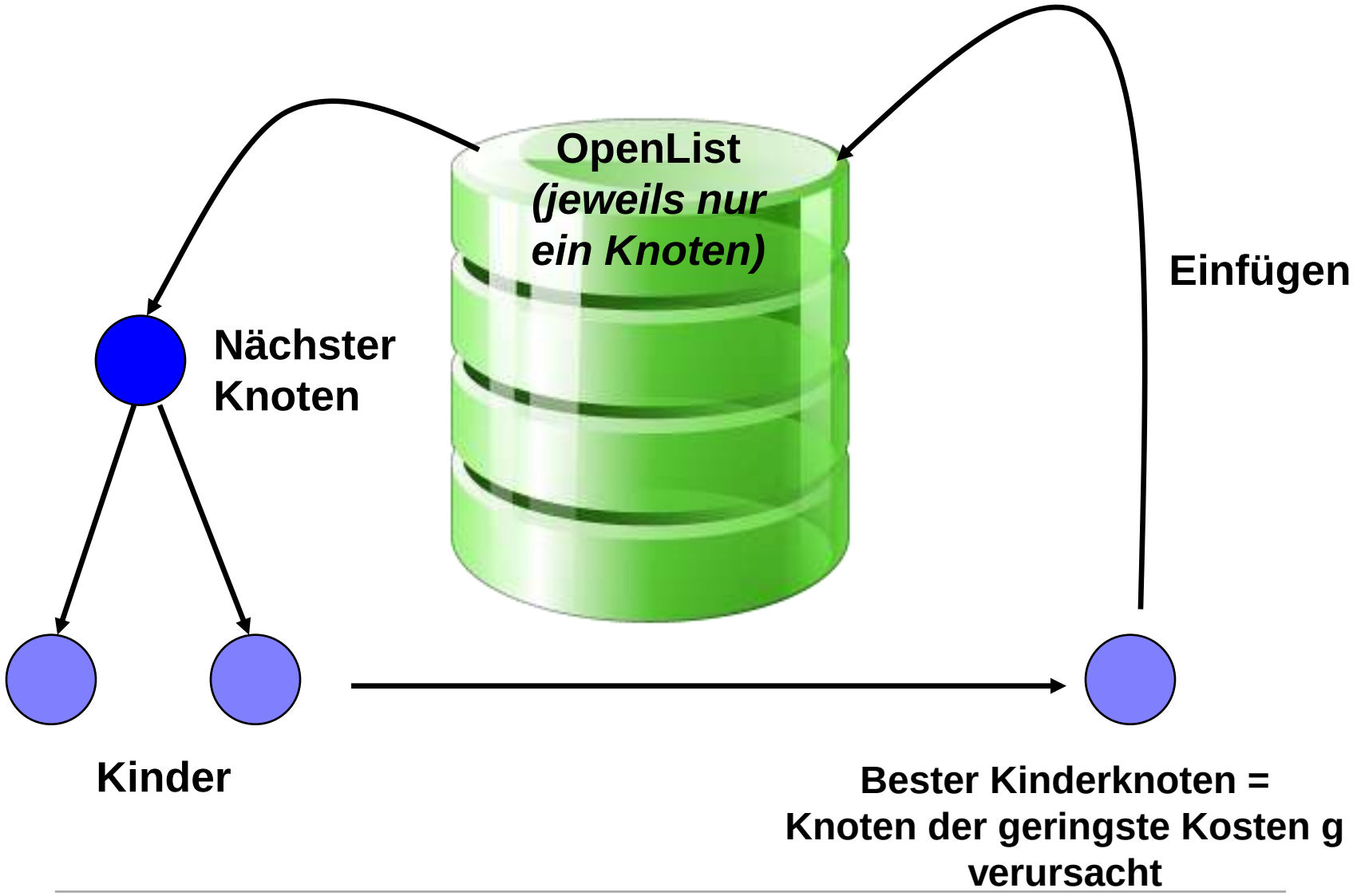
NNH-Lösung:

1118 km

optimale Lösung:

1032 km

Nearest Neighbour



Optimierungsprobleme

Bei vielen Suchproblemen gibt es mehrere Pfade zu Zielknoten, die sich in ihrer Qualität unterscheiden.

Ziel:

- Qualitativ beste Lösung finden, d.h. Pfad mit den geringsten Kosten.

oder

- Qualität einer Lösung liegt in der Lösung selbst und nicht der Pfad zur Lösung ist interessant.

Eine heuristische Funktion

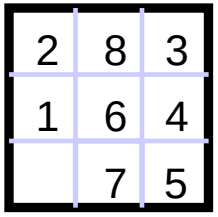
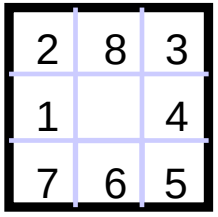
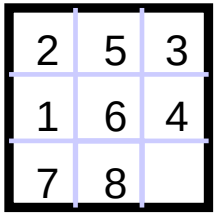
- ist eine Funktion *h*, die jedem Knoten *k* im Suchbaum eine nicht negative Zahl *h(k)* zuordnet. Diese Zahl gibt eine Schätzung für die Entfernung des Knotens *k* zum nächsten Zielknoten an. Ist *k* ein Zielknoten, so ist h(k)=0.
- ist umso besser, je stärker sie differenziert.

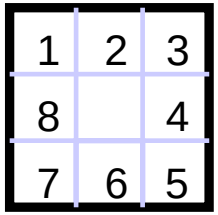
Wie findet man eine *nützliche* heuristische Funktion?

Beispiel 8-Puzzle:

1. $h_1(z)$ Anzahl der Spielsteine im Zustand z, die an falscher Position liegen. Je kleiner die Zahl, desto mehr Spielsteine liegen korrekt.
2. $h_2(z)$ Summe der Entfernungen aller Spielsteine von ihrer jeweiligen Zielposition.

verschiedene Heuristiken für das 8-Puzzle

	Anzahl Spielsteine auf falscher Position	Summe der Entfernungen, in denen sich Spielsteine zu ihren Zielpositionen befinden	2x Anzahl direkter Spielsteintausch
	5	6 1+1+0+0+0+1+1+2	0
	3	4 1+1+0+0+0+0+0+2	0
	5	8 1+1+0+0+3+1+0+2	0



Suchalgorithmen mit *f(n) = h(n)*

- Bestensuche (best-first search)
- Hill Climbing
- Hill Climbing mit Backtracking

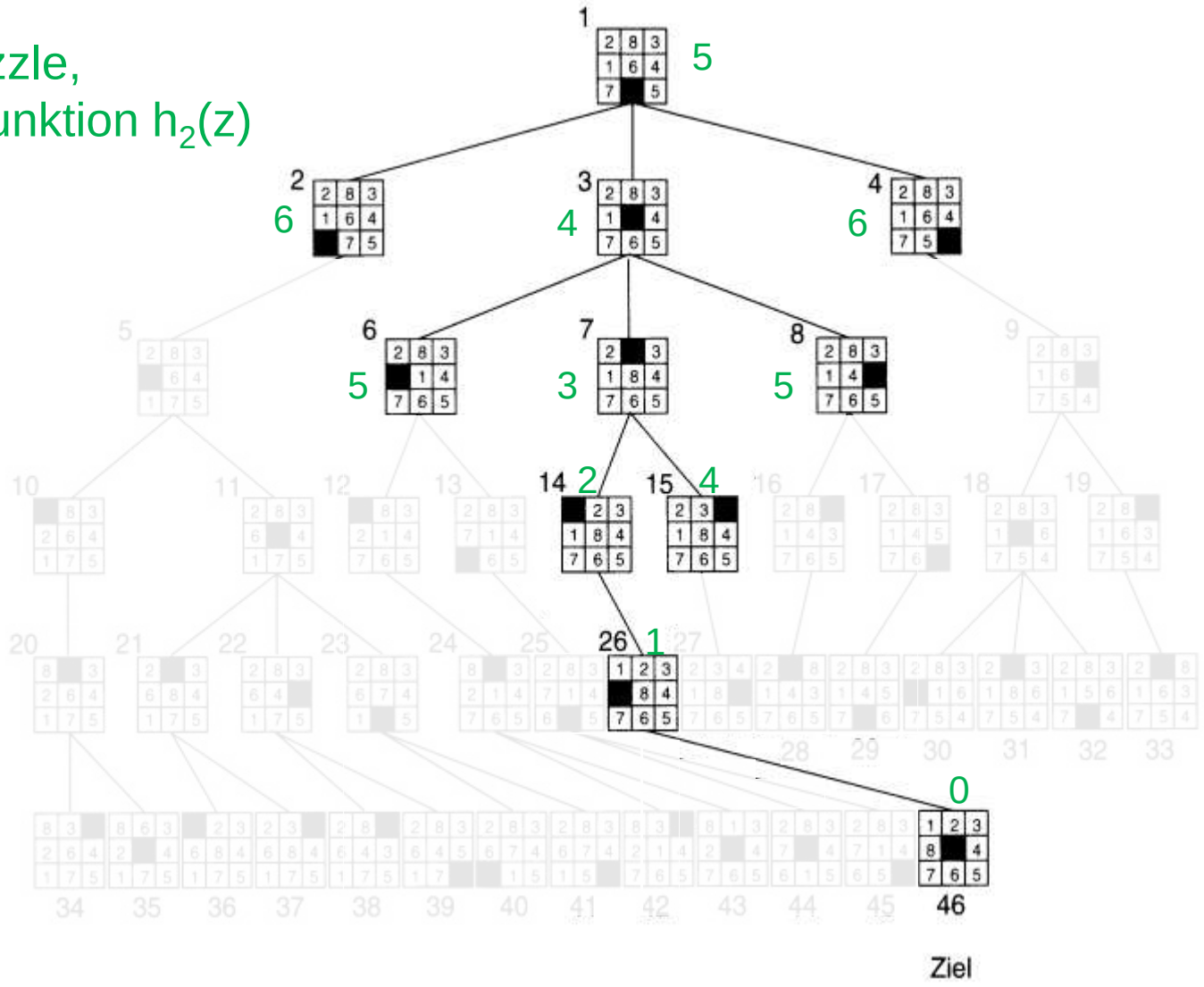
Nützlich

- für bestimmte Suchprobleme zur Bestimmung von Lösungspfaden.
- wenn nicht der Pfad, sondern ein Zielzustand gesucht wird.
- für Optimierungsprobleme
  (h(n) liefert Wert des Zustandes).

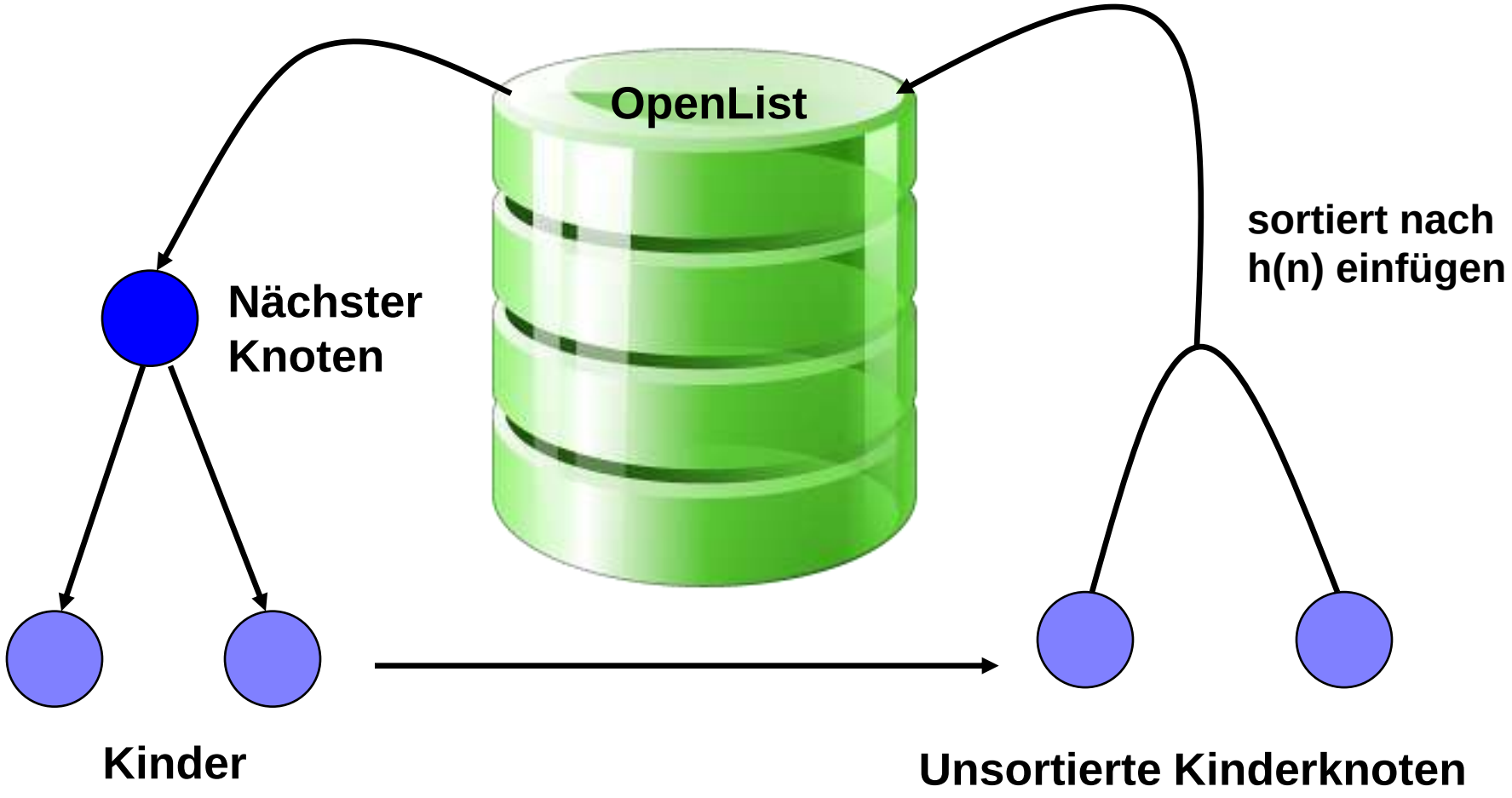
- Sinn der heuristischen Funktion: vielversprechende Knoten als nächstes zu untersuchen und alle anderen hinten an stellen.
- Sinnvoll Knoten nach ihrer Bewertung zu ordnen, die besten in der OpenList nach vorn.
- **Bestensuche** (best-first search) verwendet demzufolge ebenfalls eine OpenList, Einfügen der Knoten erfolgt entsprechend der heuristischen Werte.

Beispiel: 8-Puzzle,
heuristische Funktion $h_2(z)$

Schr.	OpenList
1	15
2	34,26,46
3	73,65,85,26 ,46
4	142,154, 65, 85,26,46
5	261,154, 65, 85,26,46
6	460,154, 65, 85,26,46



Bestensuche



Sie erinnern sich:
Travelling Salesman Problem:

Auf einer Rundreise sollen n Städte besucht werden, wobei ein bestimmter Faktor zu minimieren ist (z.Bsp. die Reisezeit).

Suche nach dem optimalen Weg

Ein optimaler Weg wird nur dann gefunden, wenn jeder mögliche Weg generiert und der mit dem geringsten Wert ausgewählt wird.

n! Kombinationsmöglichkeiten!!

Travelling Salesman Problem:

Selbst wenn ein Rechner pro Weg nur 0.0001 Sekunden plant, dauert es bei 15! über 4 Jahre, den optimalen Weg zu finden. (Kombinatorische Explosion)

n	n^2	2^n	n!
1	1	2	1
2	4	4	2
3	9	8	6
4	16	16	24
5	25	32	120
7	49	128	5040
10	100	1024	3628800
12	144	4096	479001600
15	225	32768	1307674368000

Rucksack befüllen:

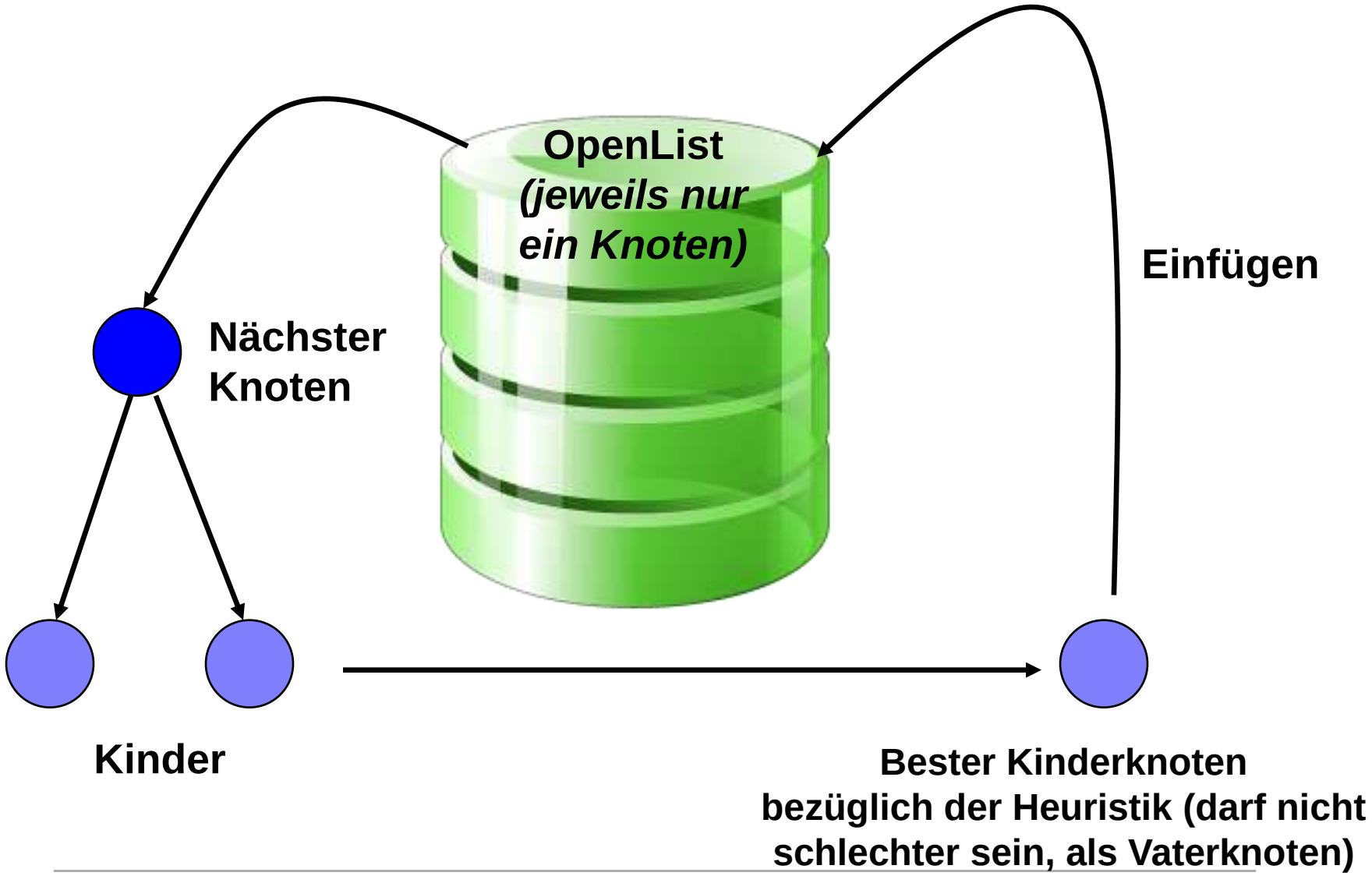
Der Rucksack kann maximal mit 20kg befüllt werden. Die Objekte, die eingepackt werden können, haben einen Wert und ein Gewicht. Ziel ist es, den Rucksack so zu füllen, dass die enthaltenen Objekte einen maximalen Gesamtwert haben.

Hill Climbing

Sonderform der Tiefensuche

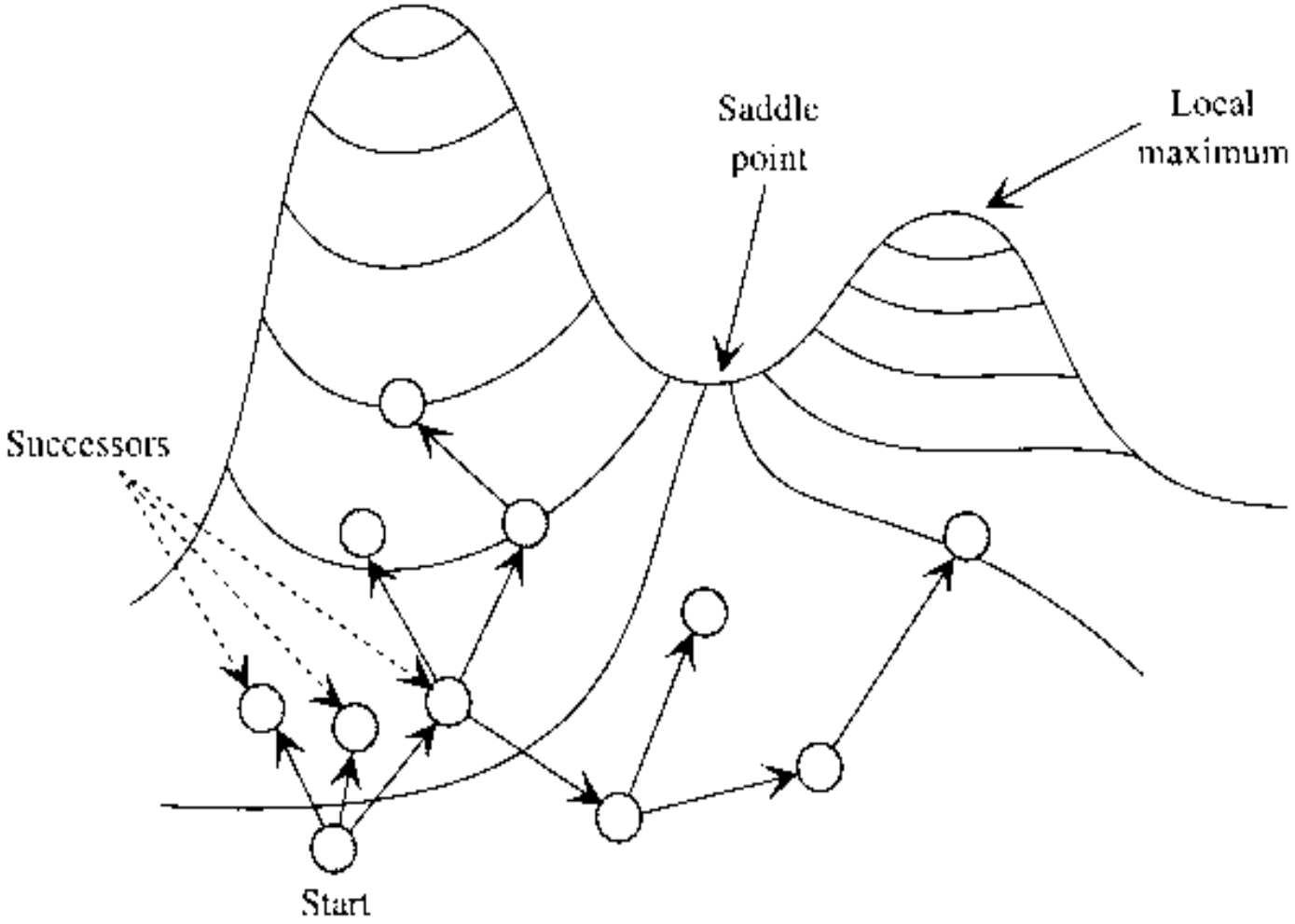
- Nur direkte Nachfolger des aktuellen Knotens werden betrachtet.
- Nachfolger mit geringstem Wert von ***h*** wird gewählt (steilster Anstieg).
- Nachfolger mit Verschlechterung gegenüber aktuellem Zustand scheiden aus.
- Häufig sehr effizient.
- Extrem geringer Speicherbedarf (maximale Verzweigungsrate).
- Nicht vollständig.
- Problem: lokale Maxima.

Hill Climbing



Problem der lokalen Maxima (1)

Der nächste Knoten, der expandiert wird, hat die kürzeste Entfernung zum Ziel.



Problem der lokalen Maxima (2)

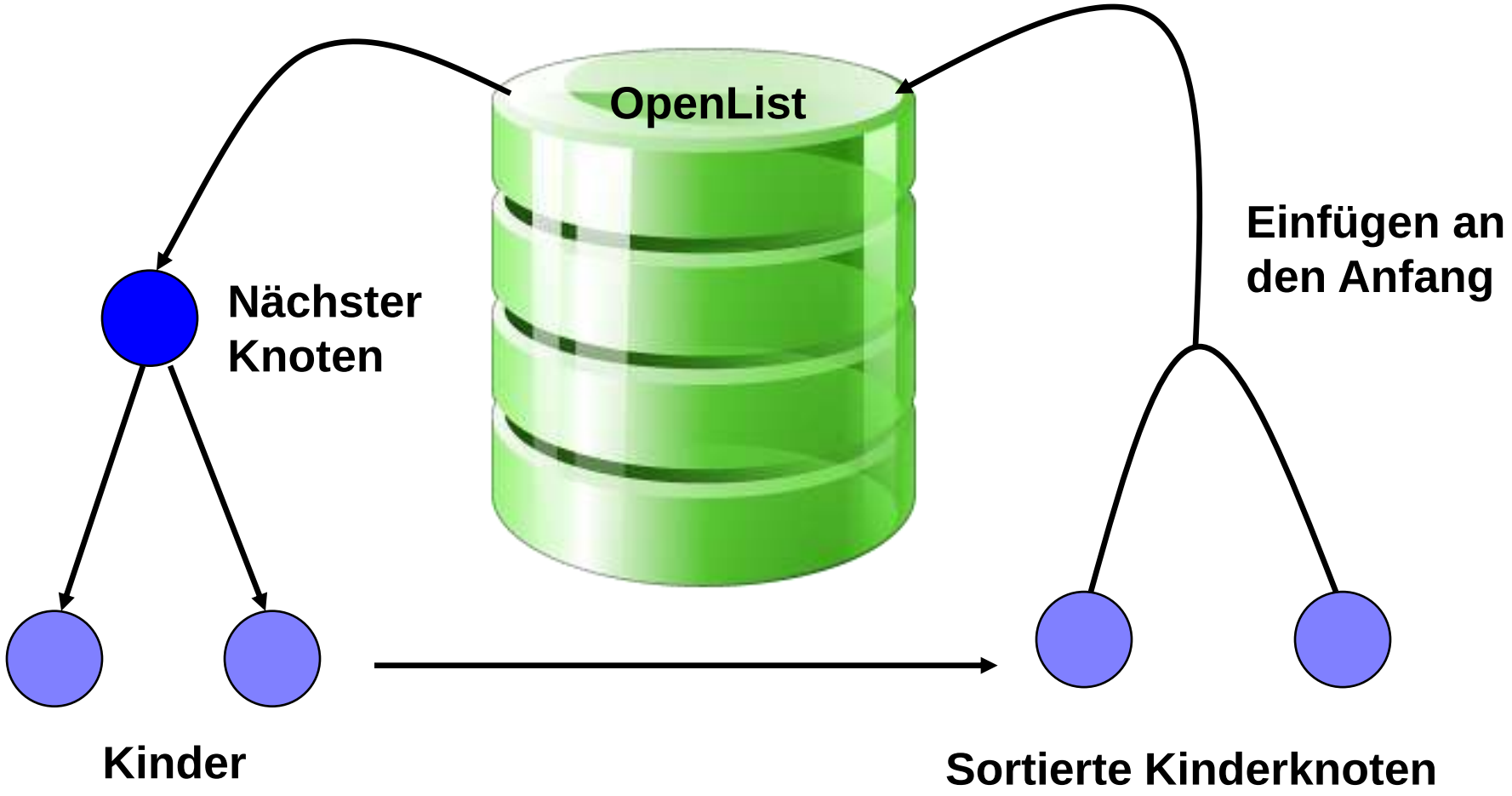
Heuristik der Nachfolgeknoten schlechter der Bewertung des aktuellen Knotens.

Es besteht aber die Möglichkeit, dass erst nach Durchlaufen eines schlechter bewerteten Knotens die Heuristiken wieder bedeutend besser werden.

Um dieses Problem zu umgehen gibt es einige Abwandlungen, bspw.

- Hill Climbing mit Backtracking
- Sintflut Algorithmus
- …

Hill Climbing mit Backtracking



Hill Climbing mit Backtracking vs. Bestensuche (1)

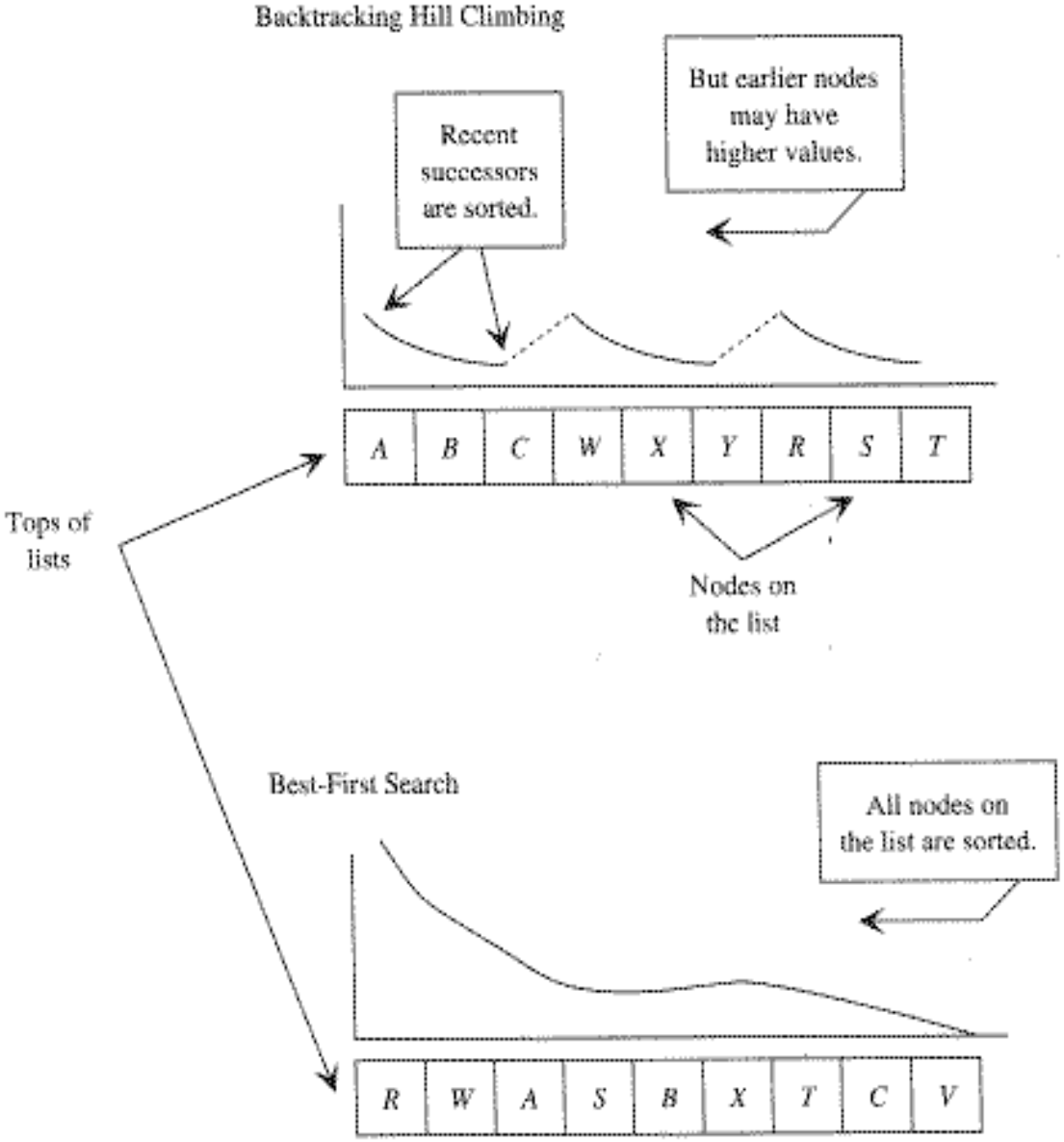
Hill Climbing mit Backtracking

- Informierte Tiefensuche (aber ohne Berücksichtigung von ***g***).
- Aber: Geschwister-Knoten jeweils untereinander sortiert.
- Backtracking.
- Nicht vollständig.

Bestensuche

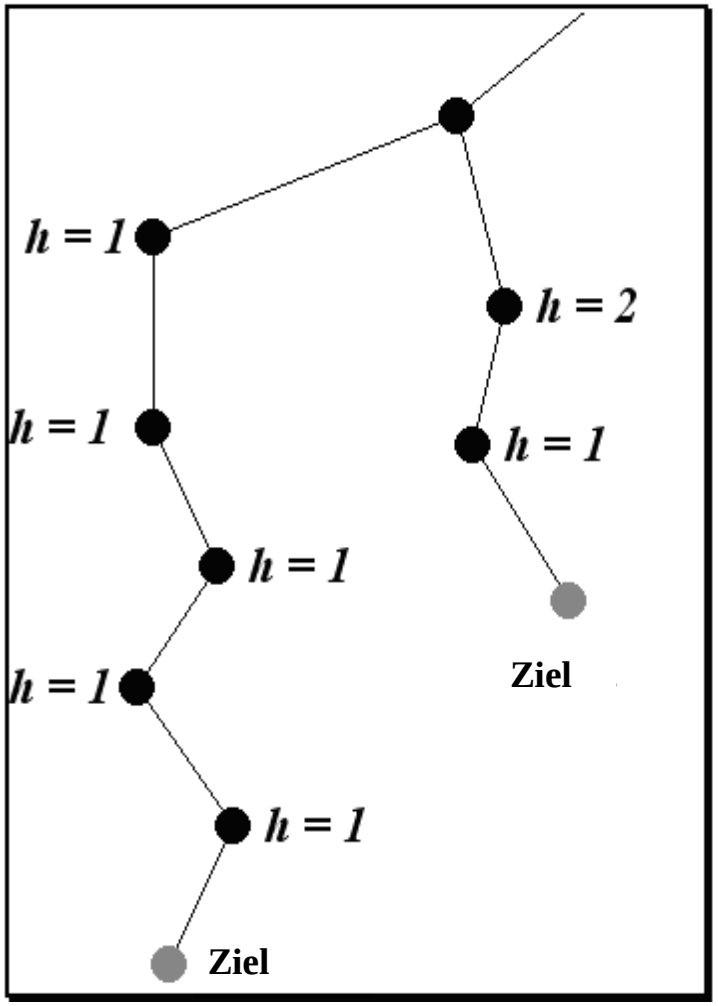
- Informierte Breitensuche (aber ohne Berücksichtigung von ***g***).
- Alle Knoten entsprechend ***h*** sortiert.
- Ebenfalls nicht vollständig!

Hill Climbing mit Backtracking vs. Bestensuche (2)



Probleme durch Vernachlässigung der bisherigen Kosten (Funktion *g*)

- Optimale Lösung kann verfehlt werden
- Lösung insgesamt kann verfehlt werden (Verlust der Vollständigkeit)
- unendlich lange Pfade mit guter Bewertung ***h***



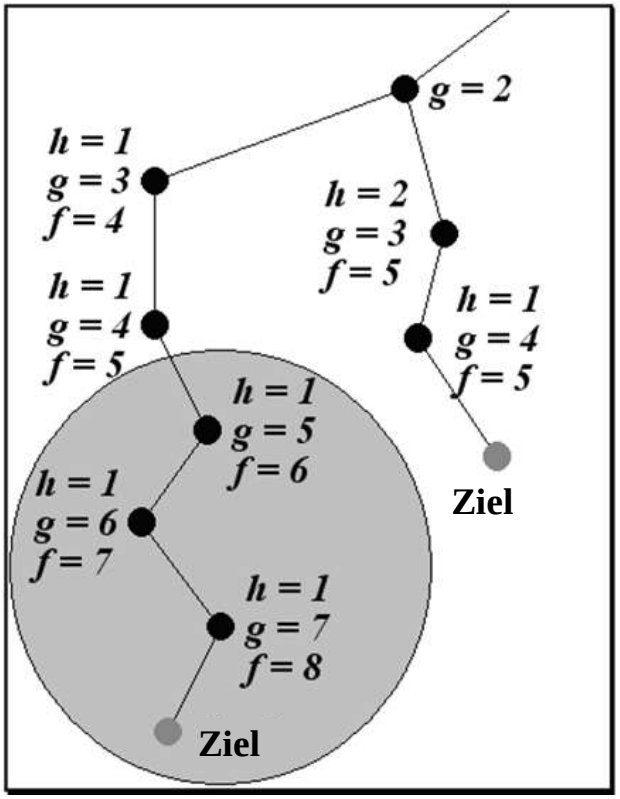
A-Algorithmus (1)

Gesamtkosten eines Knotens *f(n)*=

bisherige Kosten ab dem Start *g(n)*

\+ geschätzte Restkosten bis zum Ziel *h(n)*

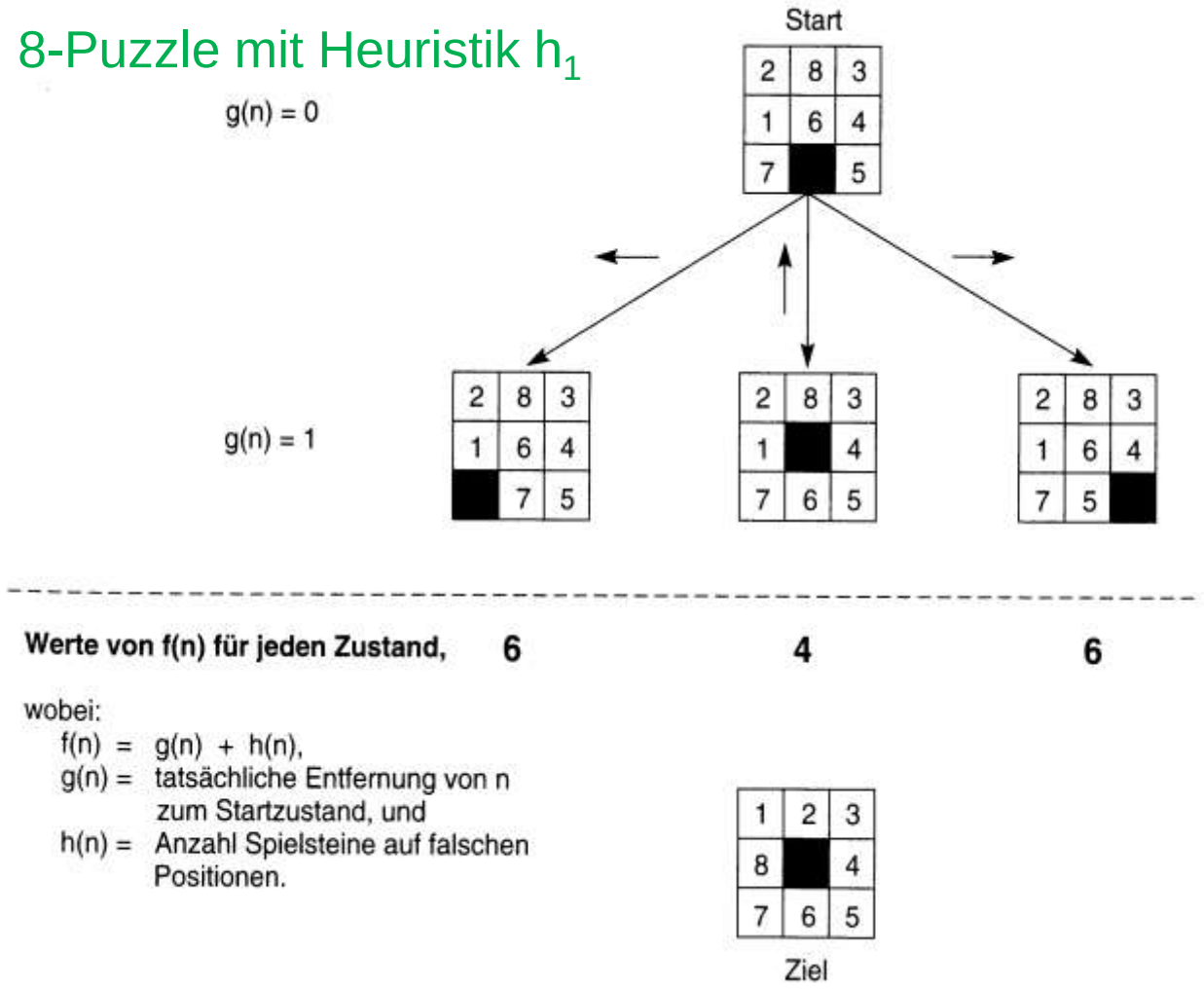
f(n) = g(n) + h(n)



A-Algorithmus (2)

(aus *Luger: „Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“*)

Beispiel: 8-Puzzle mit Heuristik h_1



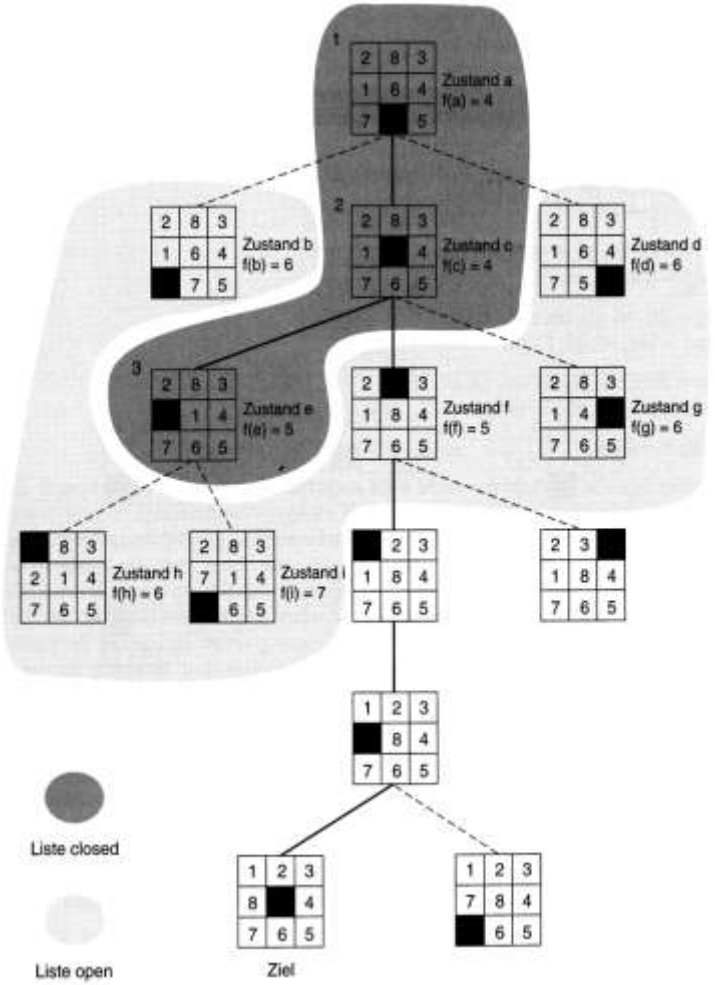
A-Algorithmus (3)

(aus *Luger: „Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“*)

Beispiel:
8-Puzzle mit Heuristik h_1,
offene Zustände nach 3 Schritten

Liste closed:
bereits abgearbeitete Zustände

Liste open:
OpenList (expandierte und bewertete Zustände zur weiteren Bearbeitung)



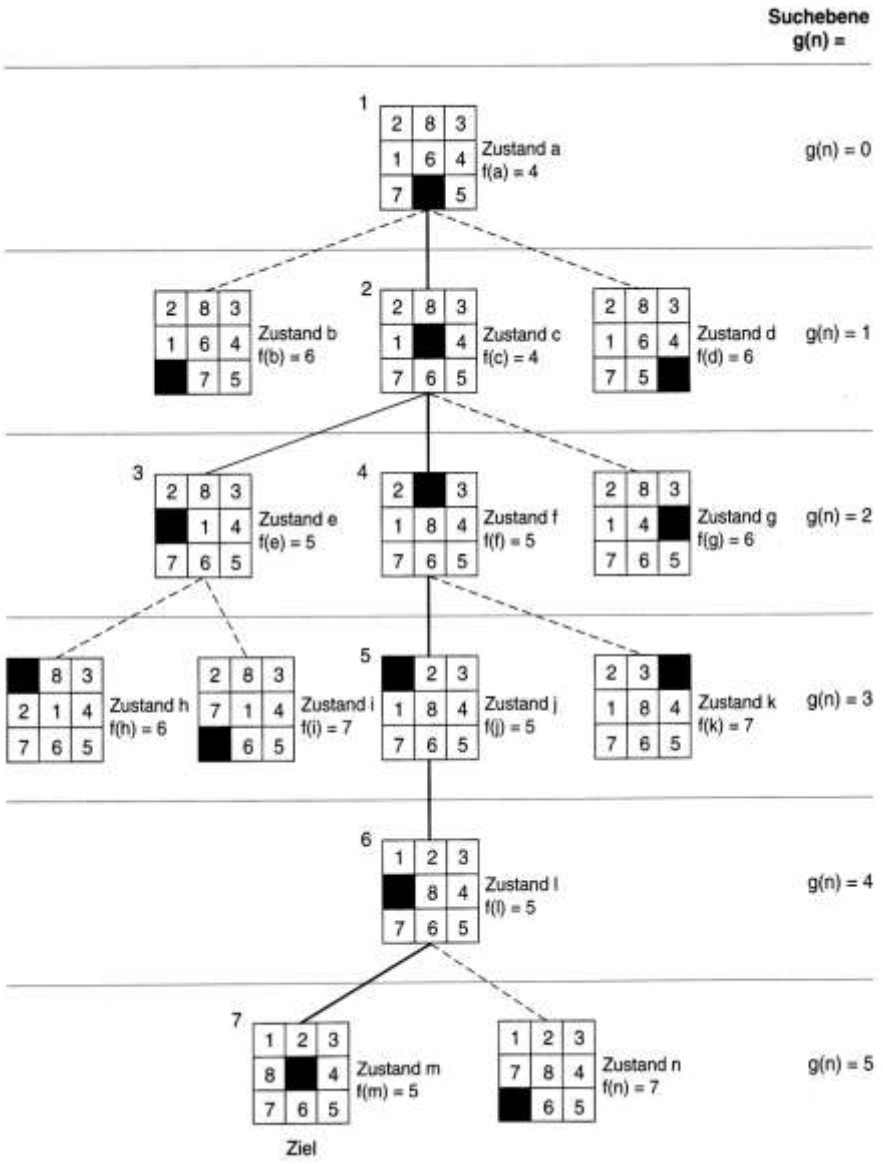
A-Algorithmus (4)

Beispiel

(aus *Luger: „Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme“*)

Beispiel:
8-Puzzle mit Heuristik h_1,
vollständig bis zum Ziel
expandierter Suchbaum

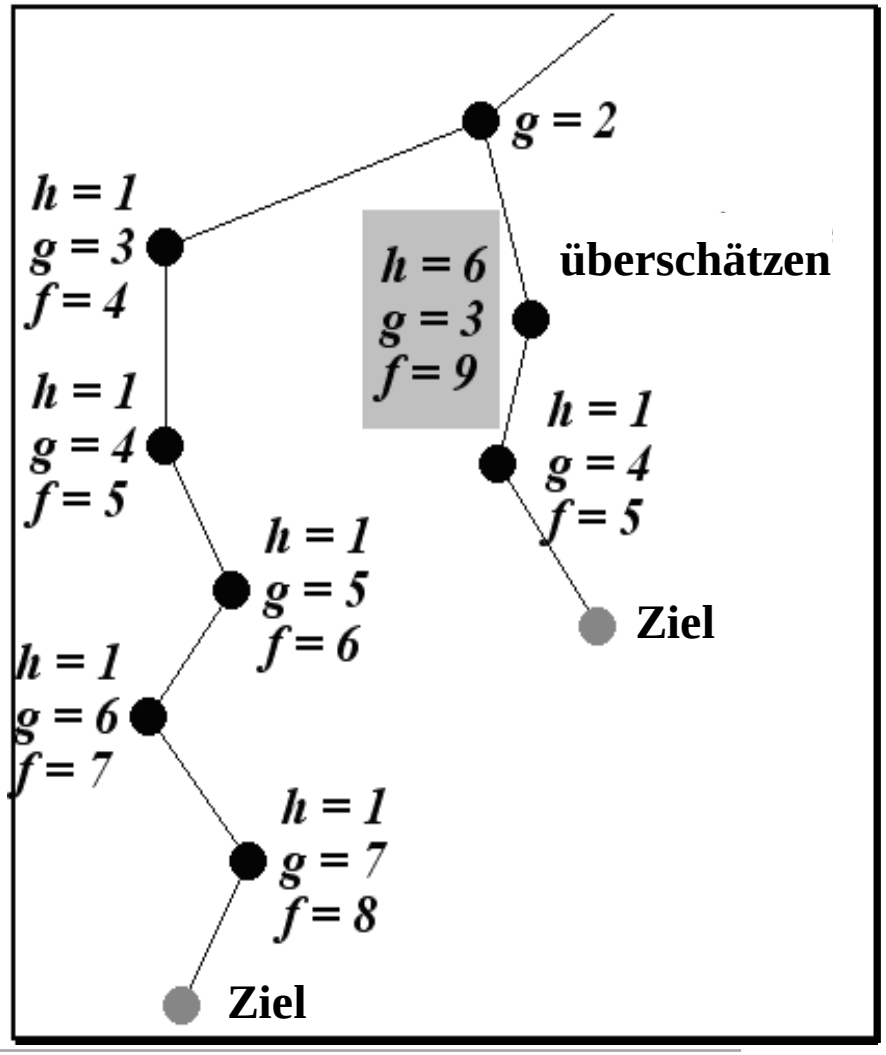
Das Ziel hat einen Kostenfaktor von 5.



A-Algorithmus (5)

Findet der A-Algorithmus die optimale Lösung?

- nicht unbedingt,
- wenn heuristische Funktion h die Kosten bis zum Ziel über-schätzt, kann ein anderes, nicht opti-males Ziel gefunden werden.



Eigenschaften von Heuristiken

Schätzfunktion / Heuristik ***h*** heißt

- zulässig, wenn sie die Restkosten nicht überschätzt:

  g(Zielknoten) – g(n) ≥ h(n)*, für alle n, bzw.*
  h*(n) ≥ h(n)*, für alle n,*
  (*h**: Funktion der tatsächlichen Kosten)

- monoton (konsistent), wenn die Kosten zwischen
  zwei beliebigen Knoten nie überschätzt werden
  D.h.: Die Werte von ***f*** entlang eines Pfades dürfen nicht fallen.
  Bsp.: Für unser 8-Puzzle: h_1 und h_2 sind monoton,
  Routenplaner: Luftlinie ist monoton

Zulässige Heuristiken sind meistens auch monoton
(Gegenbeispiele schwer zu finden).

- Für zwei zulässige Funktionen $\boldsymbol{h_1}$, $\boldsymbol{h_2}$ gilt:
  Wenn $\boldsymbol{h_2(n) \geq h_1(n)}$ für alle ***n***, dann ist $\boldsymbol{h_2}$ informierter.
  Bsp.: 8-Puzzle: h_2 ist informierter als h_1

A*-Algorithmus (1)

Wenn die in einem A-Algorithmus verwendete Heuristik zulässig ist, dann bezeichnet man dies als A*-Algorithmus**.**

Der A*-Algorithmus ist optimal, d.h. er findet den günstigsten Pfad von einem Startzustand zu einem Zielzustand.

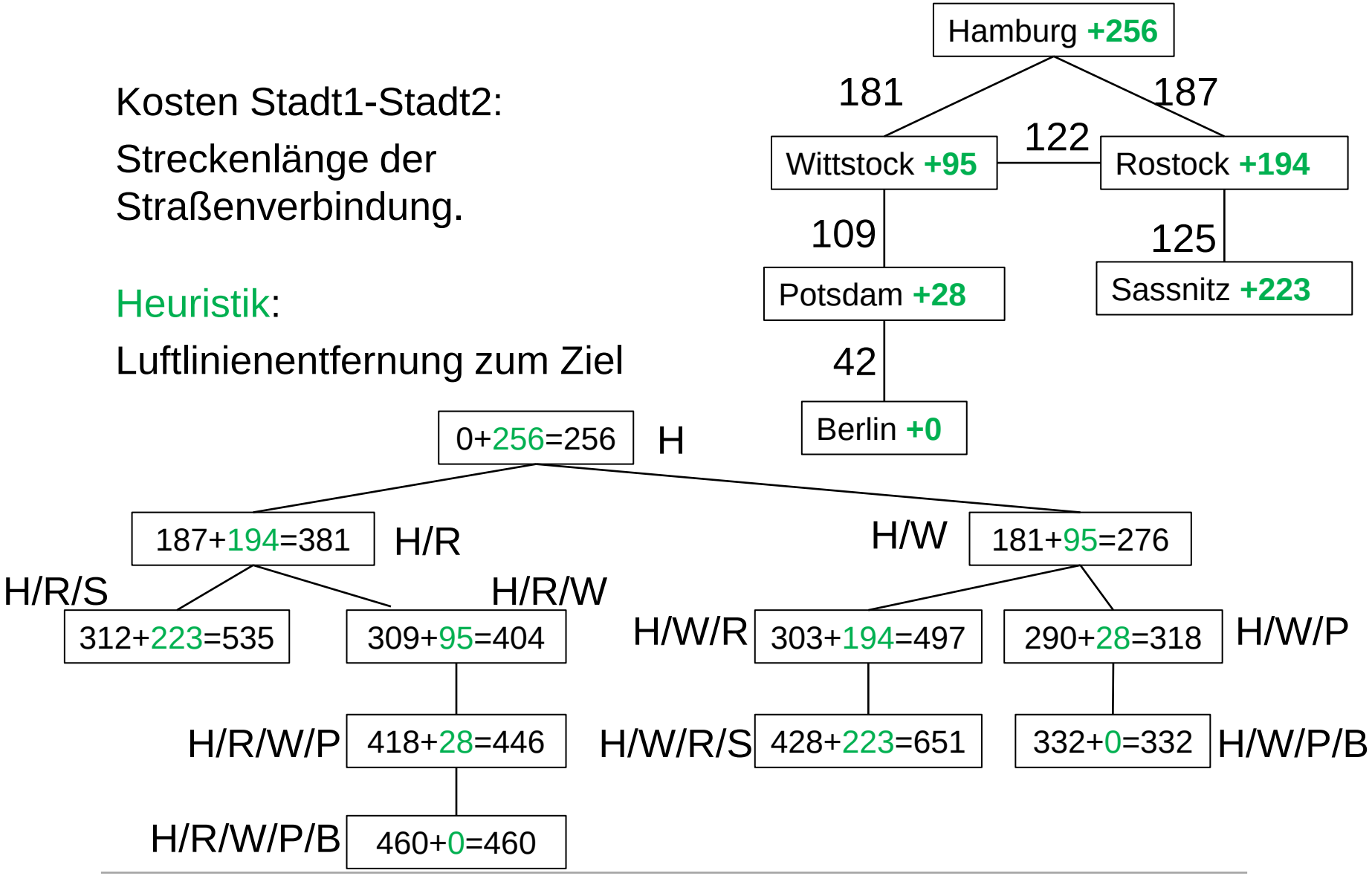
A*-Algorithmus (2)

Kosten Stadt1-Stadt2:

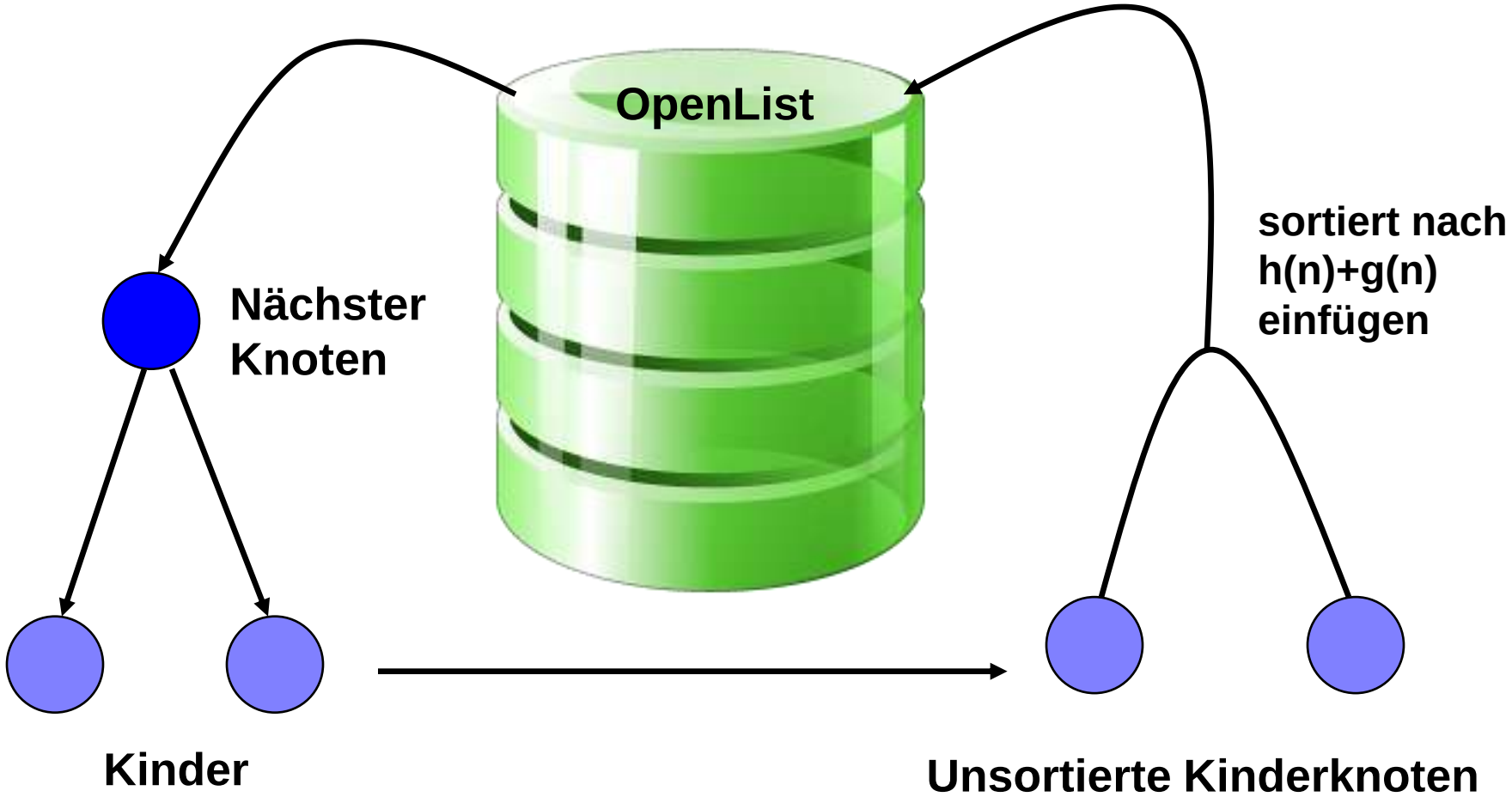
Streckenlänge der Straßenverbindung.

Heuristik:

Luftlinienentfernung zum Ziel



A*-Algorithmus (3)



A*-Algorithmus (4)

Für die meisten Probleme exponentiell

- Anzahl der expandierten Knoten wächst exponentiell in Abhängigkeit von der Tiefe bis zum Zielknoten

Primäres Problem: Speicherbedarf

Verbesserung durch Modifikation des Algorithmus

- Weniger Speicherbedarf zu Lasten der Rechenzeit.
- Nach wie vor optimal und vollständig.

A*-Algorithmus (5)

verwandte Algorithmen:

- Algorithmus von Dijkstra (verwendet keine Heuristik),
- Greedy Algorithmus (nur Restkosten werden betrachtet).

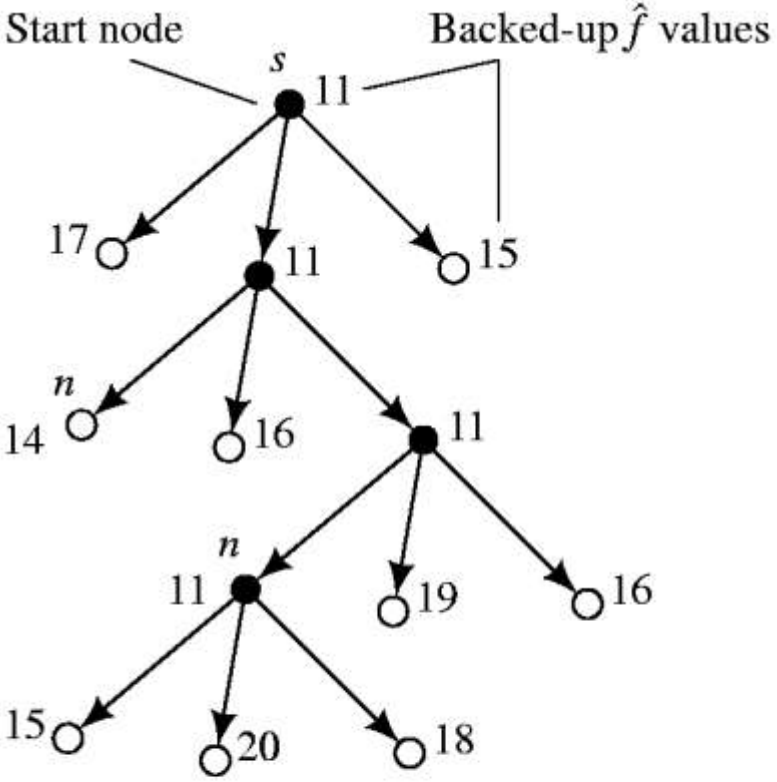
verbesserte Algorithmen:

- IDA* (Iterative Deepening A*):
  Variante der iterativen Tiefensuche.
- RBFS (Recursive Best-First Search): beschränkt den Speicherplatzverbrauch linear zur Länge der Lösung.
- MA* (Memory-Bounded A*) und SMA* (Simplified MA*): benutzen jeweils eine fest vorgegebene Speicherplatzmenge.

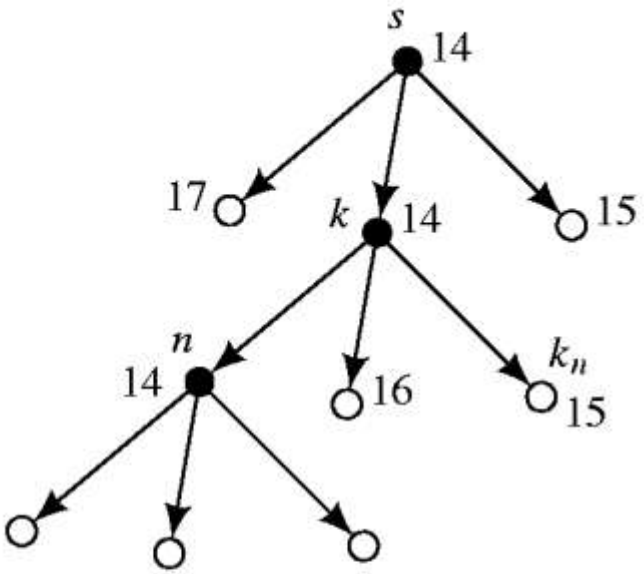
Erweiterung:

- D* (dynamischer A*): wiederholte Neuplanung nach Erhalt neuer Informationen (Navi: Brücke ist defekt, Roboter in Katastrophengebieten, …)

Rekursive Bestensuche (RBFS)



(a) RBFS has just expanded node n but has not yet backed up the $\hat{f}$ values of its successors



(b) $\hat{f}$ values have been backed up, the subtree below k_n has been discarded, and search continues below n

Vergleich von Heuristiken

(aus *Russell, Norvig: „Künstliche Intelligenz – Ein moderner Ansatz“*)

h_1: Anzahl Spielsteine auf falschen Positionen, h_2: Summe der Entfernungen

	Suchkosten			Effektiver Verzweigungsfaktor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	–	539	113	–	1,44	1,23
16	–	1301	211	–	1,45	1,25
18	–	3056	363	–	1,46	1,26
20	–	7276	676	–	1,47	1,27
22	–	18094	1219	–	1,48	1,28
24	–	39135	1641	–	1,48	1,26

Abbildung 4.8: Vergleich der Suchkosten und der effektiven Verzweigungsfaktoren für die ITERATIVE-DEEPENING-SEARCH- und A*-Algorithmen mit h_1 und h_2. Die Daten wurden über 100 Instanzen des 8-Puzzles für verschiedene Lösungslängen gemittelt.

Übersicht Suchverfahren

Verfahren		Charakteristika
Breitensuche	-	Ältere Kandidaten werden vor neueren behandelt.
Tiefensuche	-	Neuere Kandidaten werden vor älteren bearbeitet.
Tiefensuche mit Tiefenbeschränkung	-	Neuere Kandidaten werden vor älteren bearbeitet, wobei ab einer bestimmten Tiefe Knoten nicht mehr expandiert werden.
iterative Tiefensuche	-	Wie Tiefensuche mit Tiefenbeschränkung, wobei die Tiefenbeschränkung schrittweise vergrößert wird.
Bidirektionale Suche	-	Suche gleichzeitig vom Start- und vom Zielknoten ausgehend, die sich auf halbem Weg in einem gemeinsamen Knoten treffen müssen. (Eigentlich 2 Suchprozesse, vernünftigerweise Breitensuche.)
Nearest Neighbour	g	Es wird nur genau der Nachfolgeknoten, der die geringsten Kosten g verursacht, weiter betrachtet.
Bestensuche	h	Alle Kandidaten werden nach ihrer Heuristik sortiert abgearbeitet.
Hill Climbing	h	Es wird nur genau der Nachfolgeknoten mit der besten h-Bewertung weiter betrachtet, wobei diese nicht schlechter als die h-Bewertung des Vaterknotens sein darf.
Hill Climbing mit Backtracking	h	Es werden die Nachfolgeknoten nach ihrer h-Bewertung sortiert vor allen anderen Kandidaten weiter betrachtet, wobei die h-Bewertung nicht schlechter als die h-Bewertung des Vaterknotens sein darf.
A / A*Algorithmus	g+h	Bestensuche mit Bewertungs-Funktion g+h, Mehrfachwege werden vermieden. Bei A*: h darf nicht überschätzen!

… Eigenschaften, Vor- und Nachteile und Arbeitsweise der vorgestellten Suchalgorithmen erläutern und sie untereinander vergleichen können,

… Heuristiken erstellen, verstehen und bewerten können,

… Knoten anhand eines Suchalgorithmus expandieren können,

… Aufbau eines allgemeinen Suchalgorithmus erläutern können,

… erklären können, wie Steuerung der Suchstrategie über Sortierung der OpenList für die verschiedenen Suchverfahren erfolgt.