Smart Graphics: Methoden 2
Suche

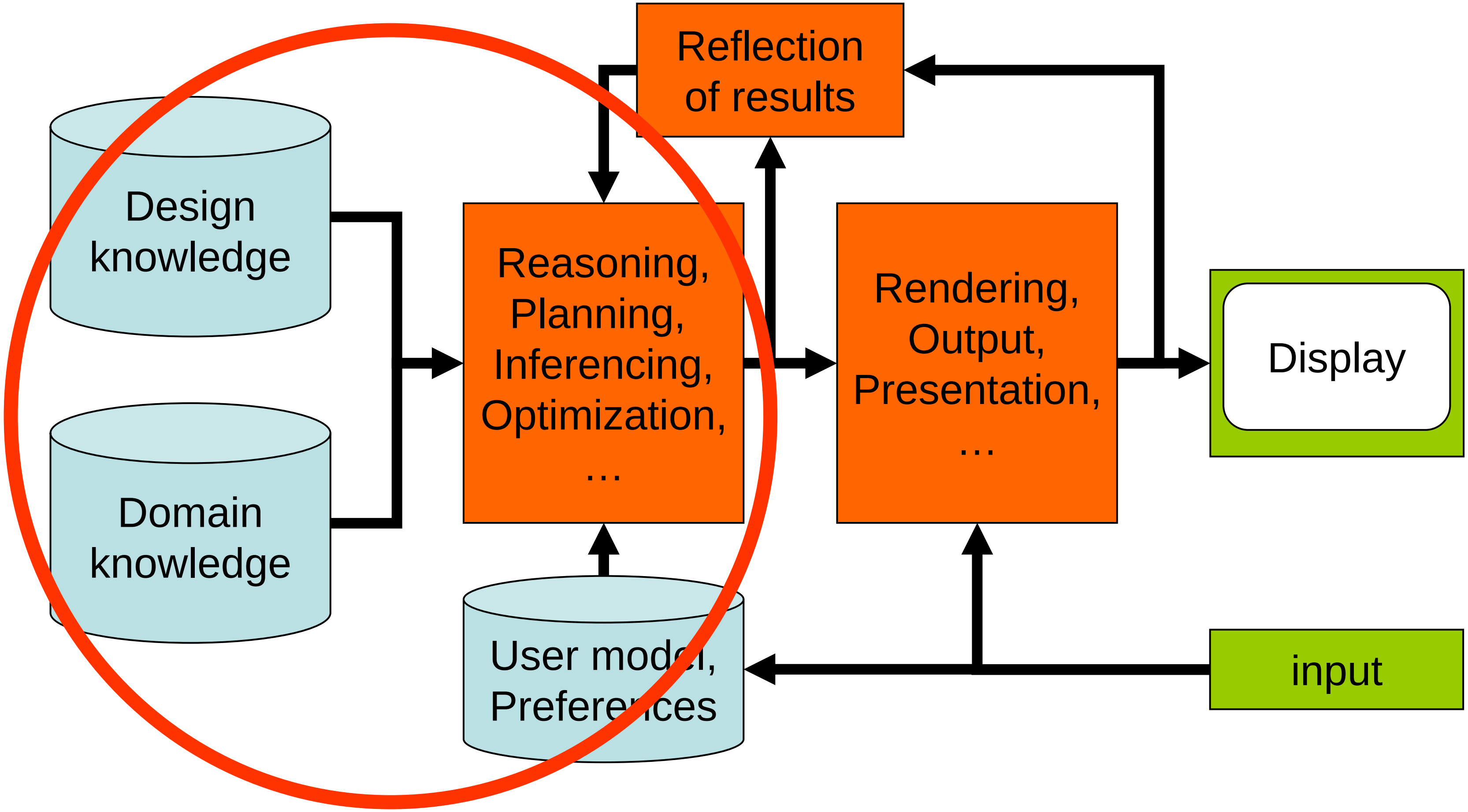
Vorlesung „Smart Graphics”

Andreas Butz, Otmar Hilliges

Mittwoch, 23. November 2005

Themen heute

- Smart Graphics Probleme als Suchprobleme
- Suchverfahren
- Graphsuche
  - Blind
  - Heuristisch
- Minimax-Verfahren

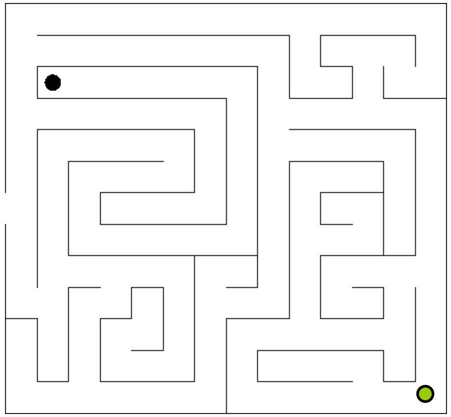

Reflection
of results
Design
knowledge
Reasoning,
Planning,
Inferencing,
Optimization,
...
Rendering,
Output,
Presentation,
...
Display
Domain
knowledge
User model,
Preferences
input

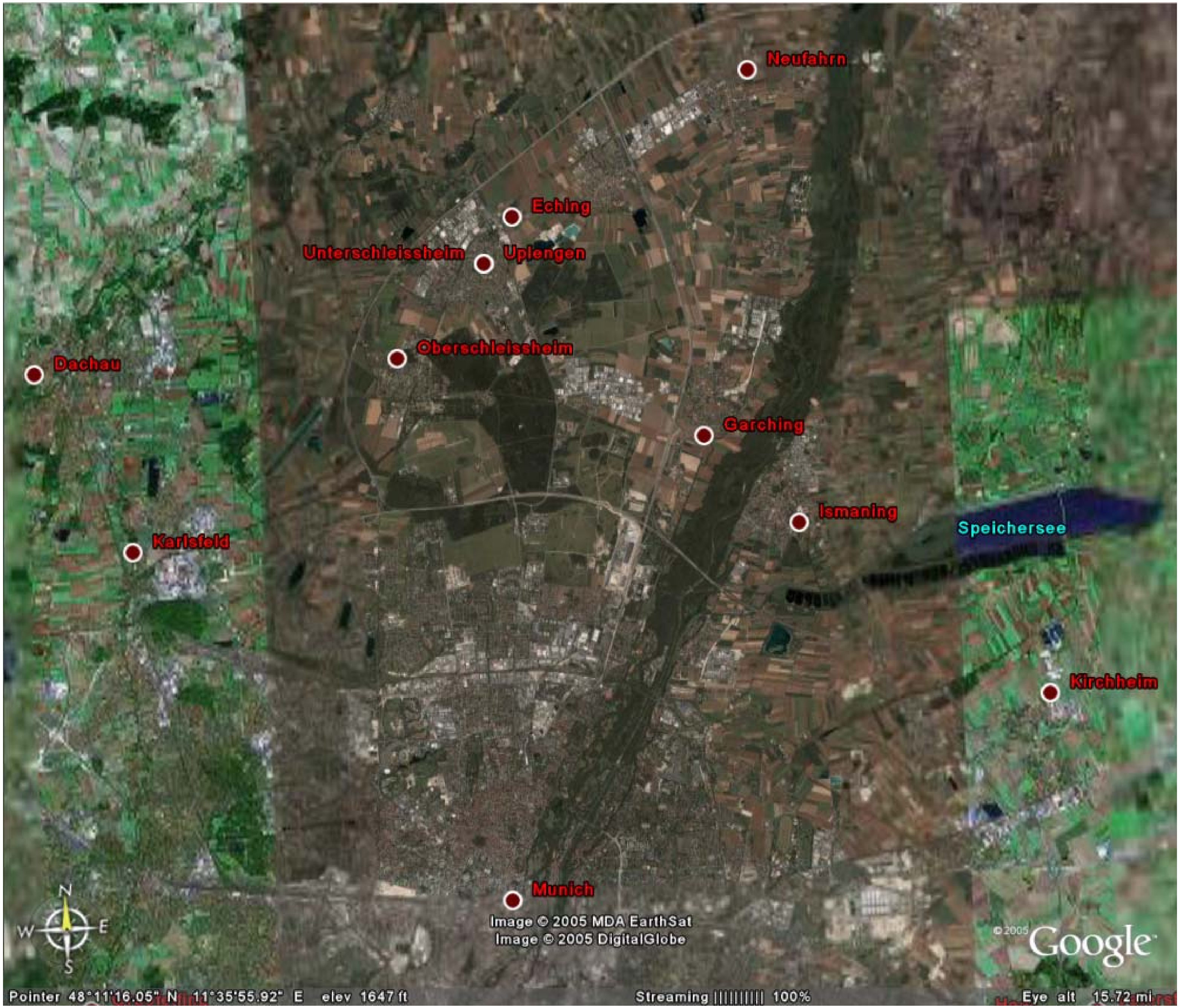
Suchverfahren

- Suchprozesse sind wichtiger Bestandteil unterschiedlicher Problemlöseverfahren
- Erster Schritt zur systematischen Suche: Formalisierung der Problemzustände in einem Zustandsgraph.
- Ein Suchschritt wird als Transformation eines Zustandes mit Hilfe eines Operators aufgefasst.
- Meist liegt ein Zustandsgraph nur implizit vor, d.h. seine Knoten und Kanten werden erst während des Suchprozesses erzeugt.
- Die Generierung von Nachfolgern eines Knotens wird als Expansion eines Knotens bezeichnet.

Wichtige Suchverfahren

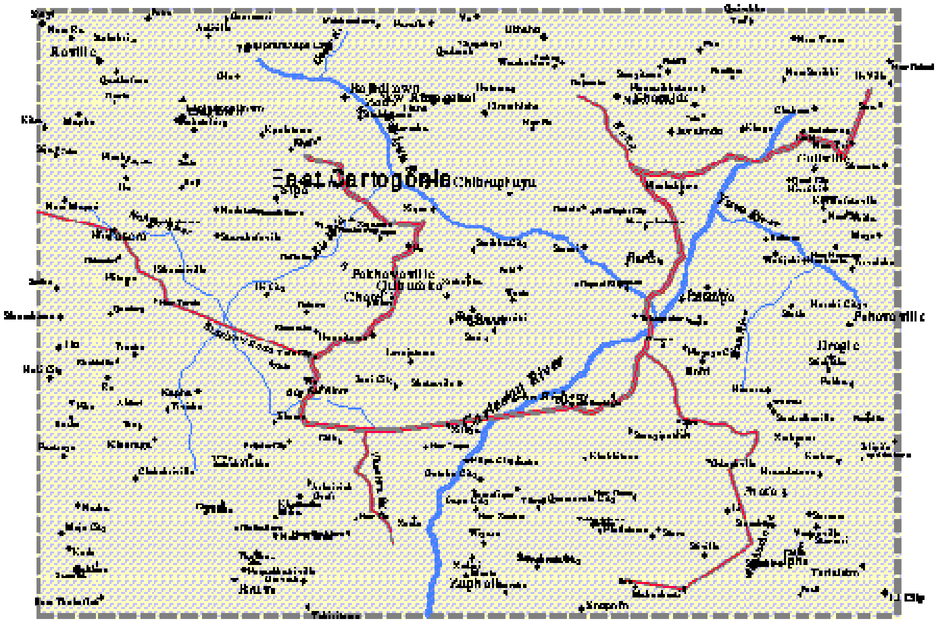
- (blinde) Tiefen- und Breitensuche
- Algorithmus A
- Algorithmus A*
- Minimax-Verfahren
- Alpha-Beta-Verfahren
- Bidirektionale Suche
- Simulated Annealing
- Genetische Algorithmen




Neufahrn
Eching
Unterschleissheim
Uplengen
Oberschleissheim
Dachau
Garching
Ismaning
Speichersee
Karlsfeld
Kirchheim
Munich
Image © 2005 MDA EarthSat
Image © 2005 DigitalGlobe
© 2005 Google
Pointer 48°11'16.05" N 11°35'55.92" E elev 1647 ft
Streaming |||||||||| 100%
Eye alt 15.72 mi
N
W
E
S

Kartenbeschriftung als Suche

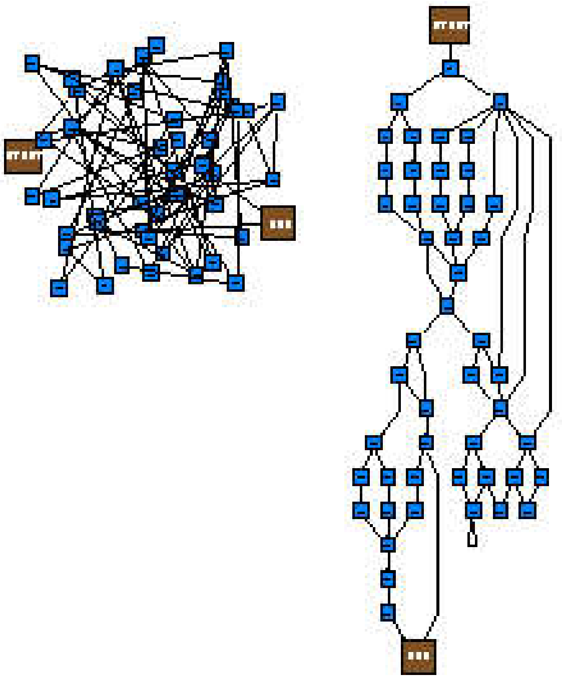
- Suche vollständige und konfliktfreie Beschriftung
- Domänenwissen = Namen der zu beschriftenden Punkte
- Verwende gestalterisches Wissen zur Ausführung der Beschriftung
  - Mindestgröße Schrift
  - Farben, Symbole
  - Haupt-Achsen
- Topologie vorgegeben
- Auch für Diagramme

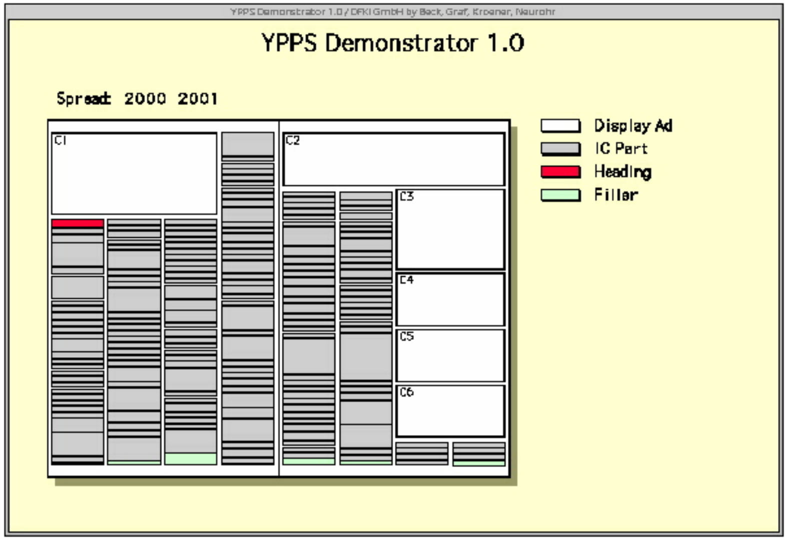


Komplexes Beispiel

Layout als Suche

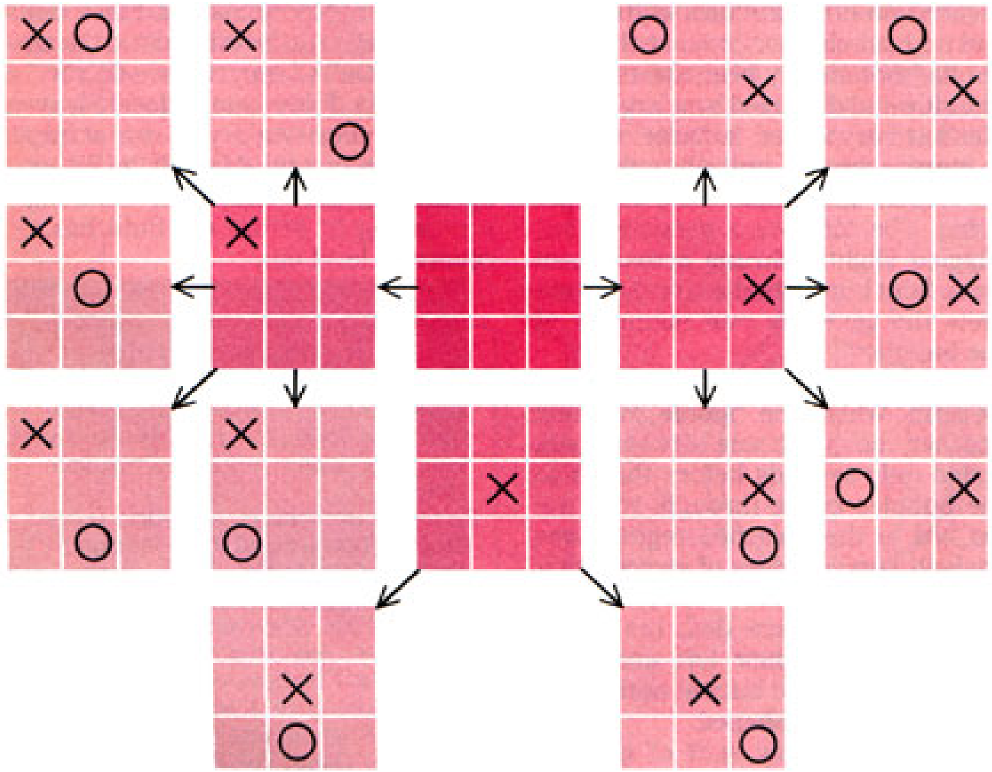
- Suche optimale Anordnung von Elementen
  - Minimaler Platzbedarf
  - Beste Lesbarkeit
- Domänenwissen = Knoten oder Textblocks
- Designwissen = Regeln für Layout
- Topologie frei veränderbar

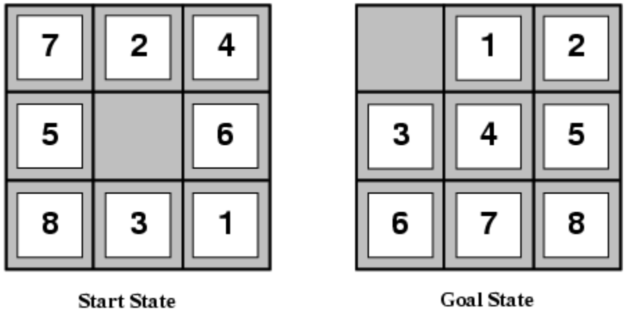




Klassische Suchprobleme

- Brettspiele
  - Schach
  - Dame
  - tic tac toe
- Knobeleien
  - 8 Damen
  - 8er Puzzle
  - Türme von Hanoi
- Je nach Größe des Problems ist eine vollständige Suche oft nicht möglich
- → heuristische Suche





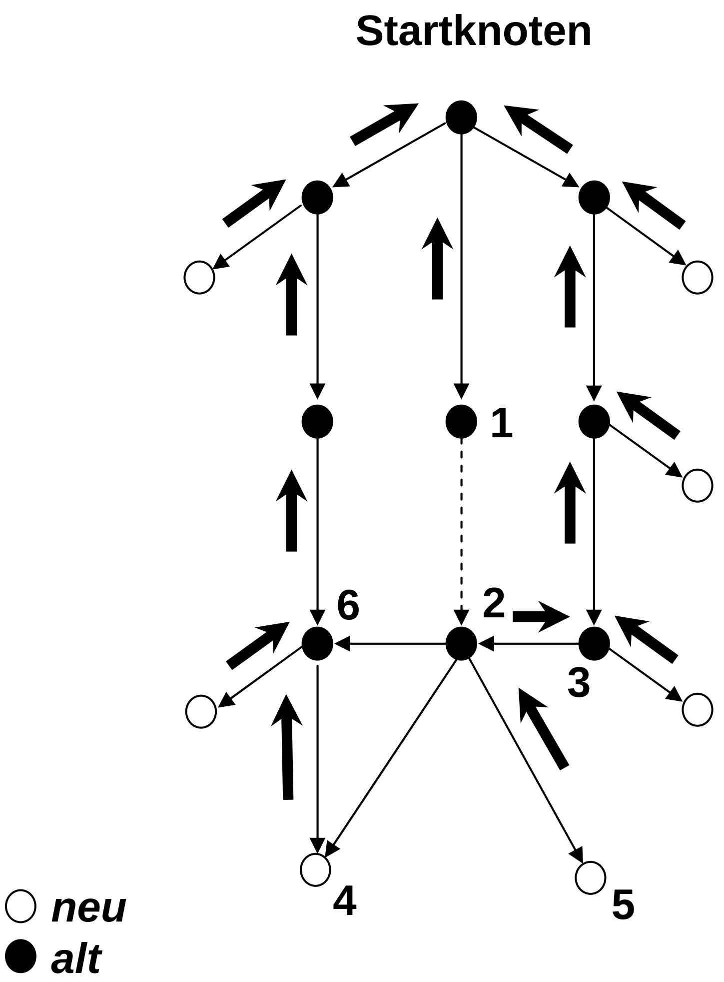
Formalisierung des Suchproblems

- Startknoten S
  - Tic tac toe: leeres Feld
  - Layout: leere Seite
  - Beschriftung: unbeschriftete Karte
- Alle Nachfolger eines Knotens durch Expansion
  - Alle Möglichkeiten zum Setzen eines Steins
  - …zum Platzieren eines Elements / einer Beschriftung
- Kriterium für Lösungsknoten
  - Tac tac toe: 3 in einer Zeile/Spalte/Diagonale
  - Layout: alles platziert
  - Beschriftung: alles beschriftet

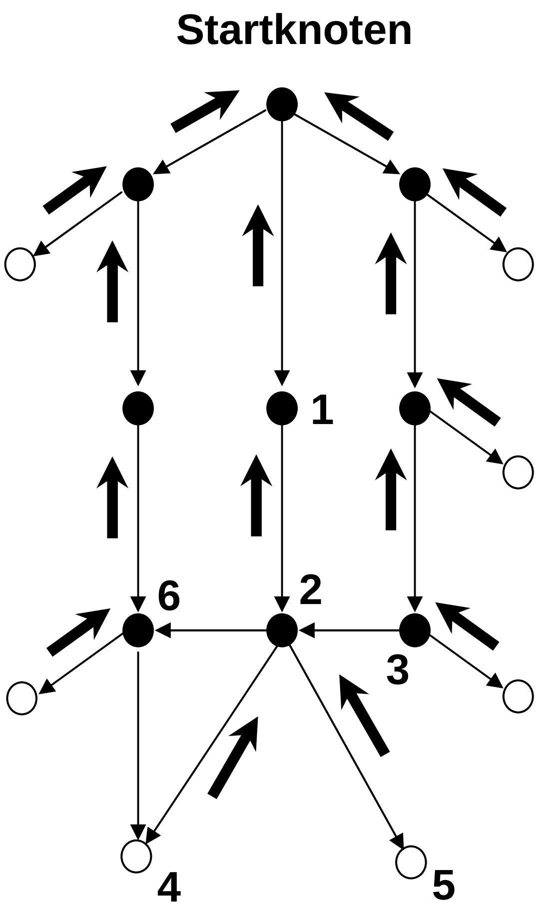
Algorithmus Graphsearch

zur systematischen Suche eines Lösungsknotens in einem Zustandsgraphen

1. Liste ***neu*** enthält Startknoten *S*, Liste ***alt*** ist leer.
2. Falls ***neu*** leer ist, breche ohne Erfolg ab.
3. Wähle Knoten aus ***neu***, nenne ihn *K*, entferne *K* aus ***neu***, füge ihn zu ***alt*** hinzu.
4. Falls *K* Lösungsknoten ist, gib Lösungsweg mit Hilfe Zeigerkette von *K* nach *S* aus, breche ab.
5. Expandiere *K*, ***m*** enthalte alle Nachfolger, die nicht Vorfahren von *K* sind.
6. Alle Knoten aus ***m***, die weder in ***neu*** noch in ***alt*** sind, erhalten einen Zeiger auf *K* und werden zu ***neu*** hinzugefügt.
   - Entscheide für alle Knoten aus ***m,*** die entweder in ***neu*** oder in ***alt*** sind, ob ihre Zeiger auf *K* gerichtet werden sollen.
   - Entscheide für alle Nachfahren der Knoten in ***m***, die bereits in ***alt*** sind, ob ihre Zeiger umgesetzt werden sollen.
7. Gehe nach 2.



Suchgraph und Suchbaum
vor Expansion von Knoten 1



Suchgraph und Suchbaum
nach Expansion von Knoten 1

Eigenschaften von Graphsearch

- Graphsearch entwickelt einen Zustandsbaum durch Vermeiden von Duplikationen bis ein Lösungsknoten gefunden ist.

- Die Kanten im Zustandsbaum sind durch Zeiger von jedem Knoten zu seinem Vorgänger repräsentiert.

- Die Kanten des Zustandsgraphen können mit unterschiedlichen Kosten markiert sein. Schritt (6) erlaubt es, Knoten so in den Baum aufzunehmen, dass sie von *S* auf dem kostengünstigsten (nicht notwendigerweise kürzesten) Pfad erreicht werden.

- In Schritt (3) sind verschiedene Auswahlkriterien möglich.

Variationen von Graphsearch

(Michie/Ross 1970)

- Erzeuge in Schritt (5) jeweils nur einen Nachfolger pro Durchlauf, setze *K* erst dann auf ***alt***, wenn alle Nachfolger erzeugt sind.

- Auf diese Weise können ggf. Expansionskosten eingespart werden (z.B. wenn der Lösungsknoten als Nachfolger eines Knotens *Y* erzeugt wird, bevor alle Nachfolger von *Y* generiert sind.)

Spezielle Baumsuchmethoden

Graphsearch

Nicht-informierte Methoden
‘blinde Suche‘:
z.B. Breitensuche
Tiefensuche

Informierte Methoden – Heuristiken
Heuristische Suche:
z.B. Bewertungsfunktionen
A*-Algorithmus

Breitensuche

Auswahlkriterium für Schritt (3) in Graphsearch: ‚Wähle Knoten *geringster* Tiefe.‘

- Breitensuche findet kürzesten Pfad zu einem Zielknoten (falls es einen gibt).

Tiefensuche

Auswahlkriterium für Schritt (3) in Graphsearch: ‚Wähle Knoten *größter* Tiefe.‘

- Tiefenbeschränkung notwendig (z.B. 8-er Puzzle: maximale Tiefe = 5)

Backtracking

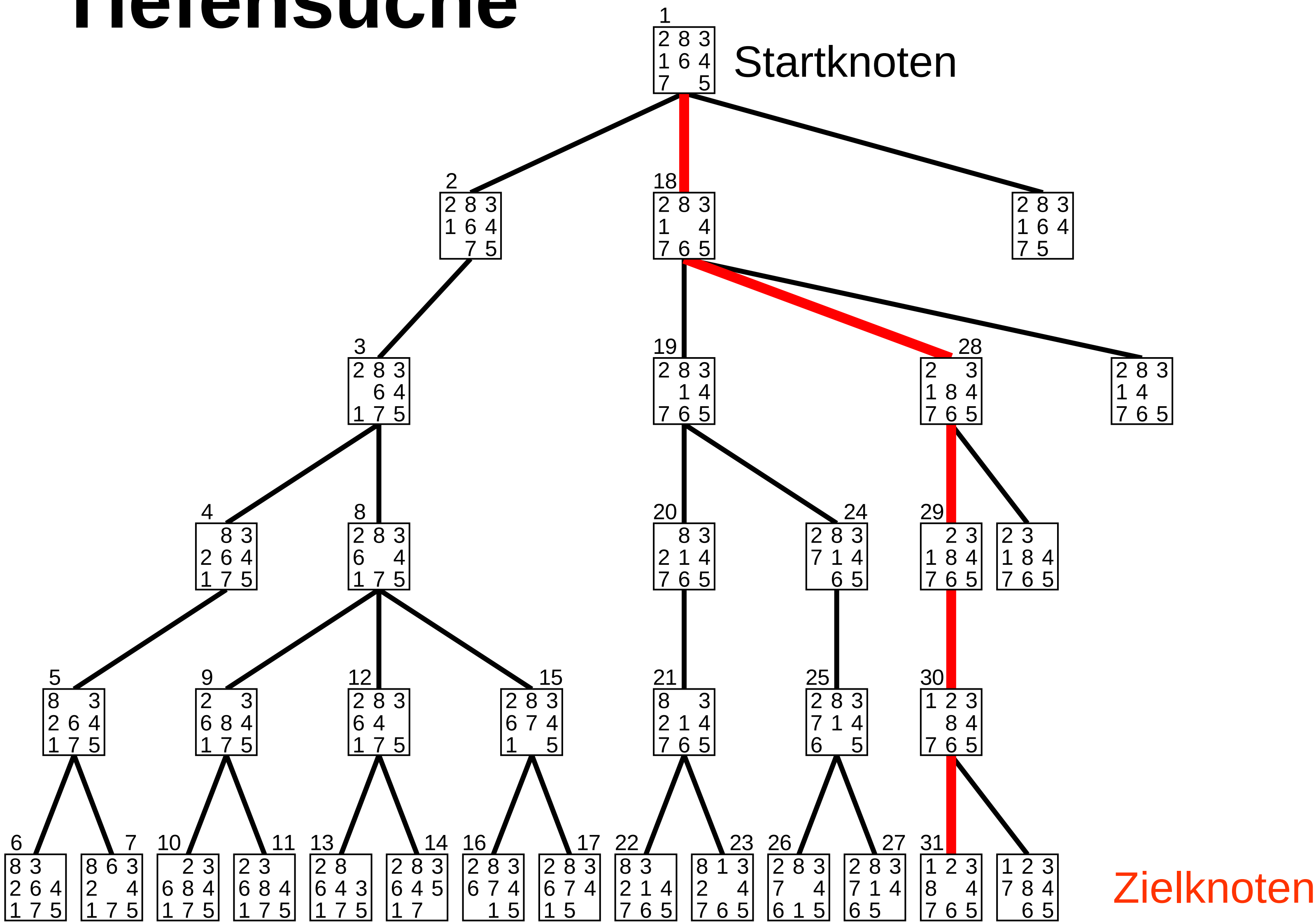
Entspricht Tiefensuche, bei der jeweils ein Nachfolger generiert und nur ein Pfad (vom aktuellen Knoten zur Wurzel) gespeichert wird.

Eigenschaften Nicht-informierter Methoden:

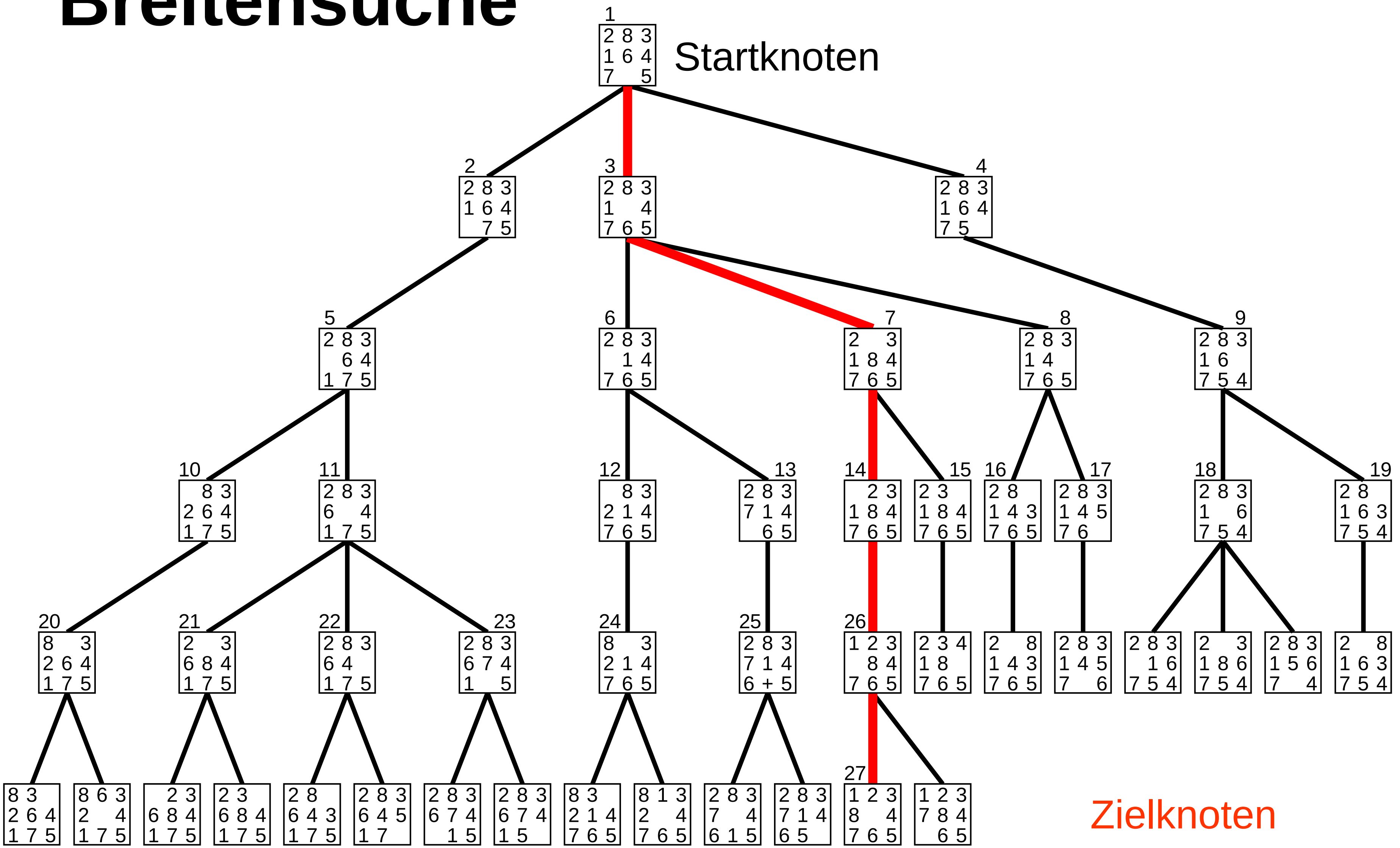
- einfache Kontrollstruktur
- Expansion vieler Knoten

Verringerung des Aufwandes durch Berücksichtigung zusätzlicher Probleminformation – Informierte Methoden, Heuristische Suche

Suchbaum für 8-er Puzzle bei Tiefensuche



Suchbaum für 8-er Puzzle bei Breitensuche



Bewertungsfunktionen zur Steuerung der Knotenauswahl

- In Schritt (3) von Graphsearch wird für einen Knoten K geprüft:
  - „Wahrscheinlichkeit, daß K auf dem Lösungspfad liegt“,
  - „Abstand von der Lösung“
  - „Qualität der Teillösung“
- Ansatz: f(K) reellwertige Funktion
- f(K1) < f(K2) ➔ K1 vor K2 untersuchen

Algorithmus A

- Seien S Startknoten, K beliebiger Knoten, T_i Zielknoten, i = 1, 2, 3, ...
- f(K) sei Kostenschätzung für günstigsten Pfad von S über K zu einem Zielknoten T_i.
- K_i mit geringstem $f(K_i)$-Wert wird als nächster expandiert.
- $k(K_i, K_j)$ seien die tatsächlichen Kosten für günstigsten Pfad zwischen K_i und K_j (undefiniert, falls kein Pfad existiert).
- $h^*(K) = \min_i k(K, Ti)$ Minimalkosten für Weg von K zu einem Zielknoten T_i
  $g^*(K) = k(S,K)$ Minimalkosten für Weg von S nach K
  $f^*(K) = g^*(K) + h^*(K)$ Minimalkosten für Lösungsweg über K

- Zerlege f(K) in 2 Bestandteile:
  - f(K) = g(K) + h(K)
  - schätzt g*(K) schätzt h*(K)
- Wahl für g(K): Summe der bish. Kantenkosten von S bis K ➔ g(K) >= g*(K)
- Wahl für h(K): beliebige heuristische Funktion
- Graphsearch mit dieser Bewertungsfunktion heißt Algorithmus A.
- Sonderfall: g(K) = Tiefe von K, h = 0, →Breitensuche ist ein Algorithmus A.

Türme von Hanoi

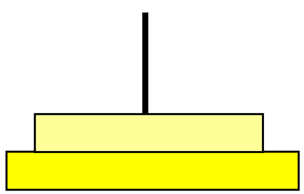
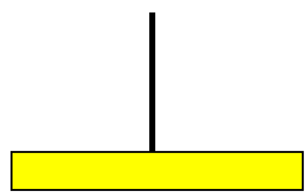
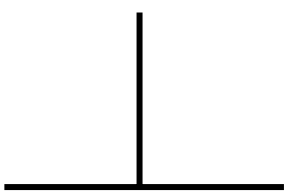
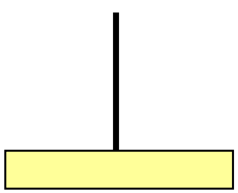
Es gibt 3 Gerüste und 64 gelochte Scheiben unterschiedlicher Größe.

Beginn:
Alle Scheiben sind nach Größe geordnet auf dem ersten Turmgerüst.

Ziel:
Turm auf 3. Turmgerüst transferieren.

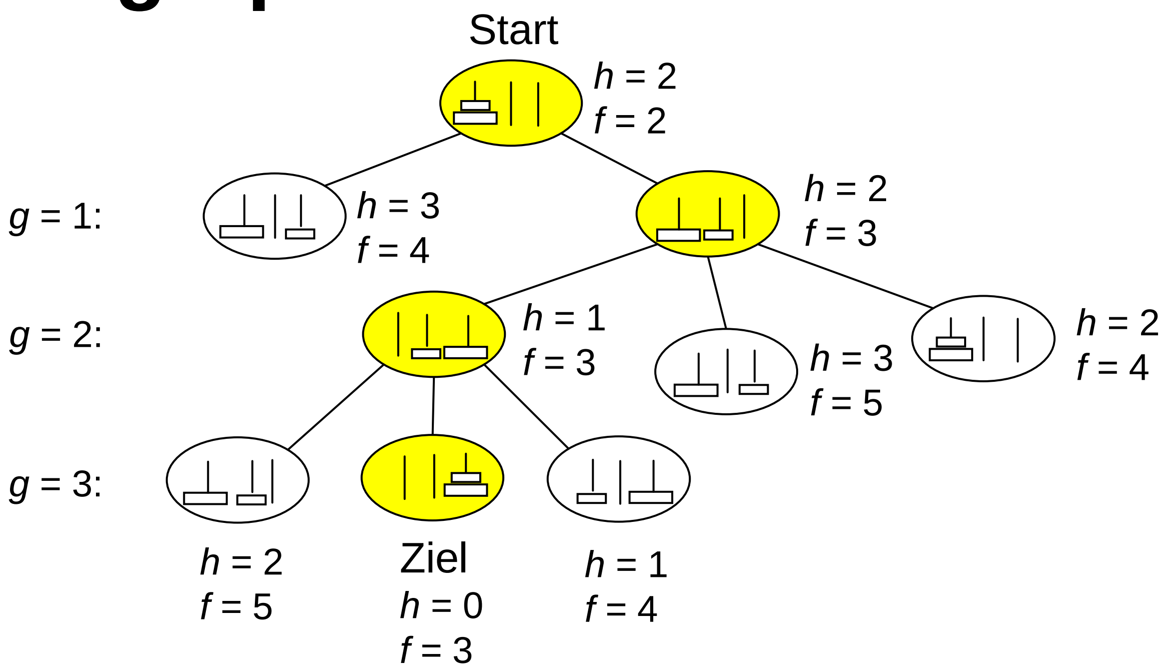
Randbedingung:
Nur jeweils eine Scheibe darf in einem Schritt bewegt werden.
Keine Scheibe darf zwischenzeitlich auf einer kleineren Scheibe liegen.

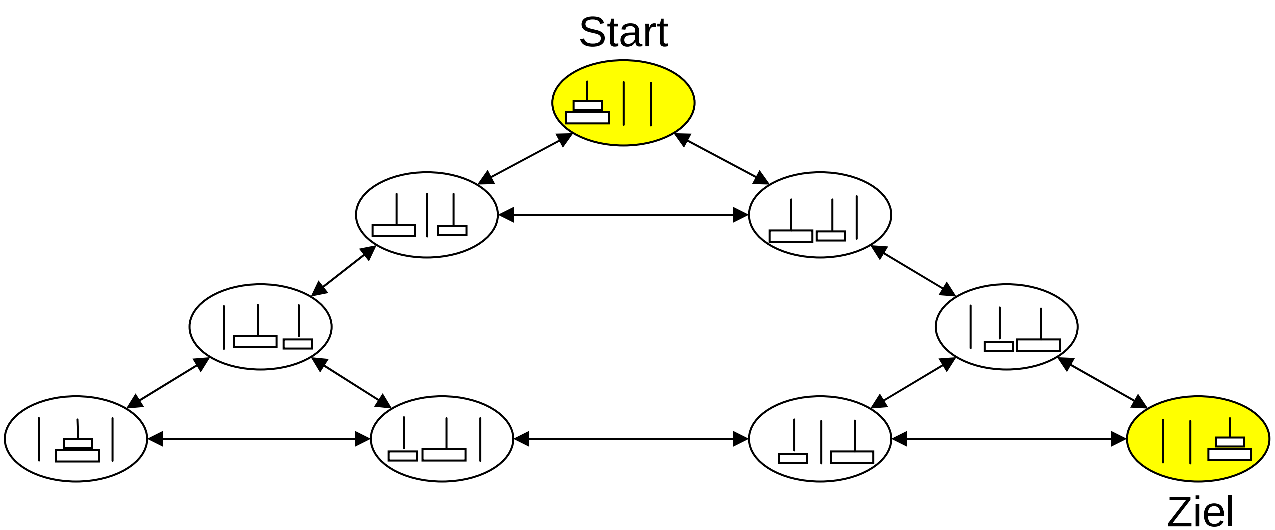
Legenden:
Einige Mönche in der Nähe von Hanoi arbeiten an dem Puzzle und wenn es fertig ist, ist das Weltende gekommen.

Endturm-gerüst				
h	0	1	2	3

$$f(N) = \text{Tiefe}(N) + \text{Endturmgerüstbewertung}(N)$$

Zustandsgraph für die Türme von Hanoi





Algorithmus A*

Ein Algorithmus A mit der Eigenschaft **h(*K*) <= h*(*K*)** findet den optimalen Lösungsweg. Ein solcher Algorithmus heißt A*. Da h(*K*) = 0 <= h*(*K*) folgt: Breitensuche ist ein Algorithmus A*.

Def.: Ein Suchalgorithmus heißt *zulässig*, wenn er für alle Graphen den optimalen Lösungsweg findet und damit terminiert, falls ein solcher existiert.

Es gilt: A* ist zulässig.

h = 0 ergibt Zulässigkeit, aber führt zu ineff. Breitensuche.

h als größte untere Schranke von h* expandiert am wenigsten Knoten und garantiert Zulässigkeit.

Oftmals: Zulässigkeit aufgeben, um härtere Probleme durch heuristische Verfahren lösen zu können.

Bewertungsfunktionen

Die *relative Gewichtung* von g und h für f kann durch eine positive Zahl w gesteuert werden: $f(K) = g(K) + w\, h(K)$.

großes w: betont die heuristische Komponente
kleines w: führt zu einer Annäherung an eine Breitensuche

oft günstig: w umgekehrt proportional zur Tiefe der untersuchten Knoten variieren:

bei *geringer* Suchtiefe: hauptsächlich gesteuert durch Heuristik
bei *großer* Suchtiefe: stärker breitenorientiert, um Ziel nicht zu verfehlen.

Zusammenfassung:
3 wichtige Einflussgrößen für die heuristische Stärke des Algorithmus A:

- Die Pfadkosten
- Die Zahl der expandierten Knoten zum Finden des Lösungsweges
- Der Aufwand zur Berechnung von h.

Fokussierung der Suche durch h(*N*) = *P*(*N*) + 3*S*(*N*)

P(*N*): Summe der Abstände jeder Kachel von ihrem Zielfeld

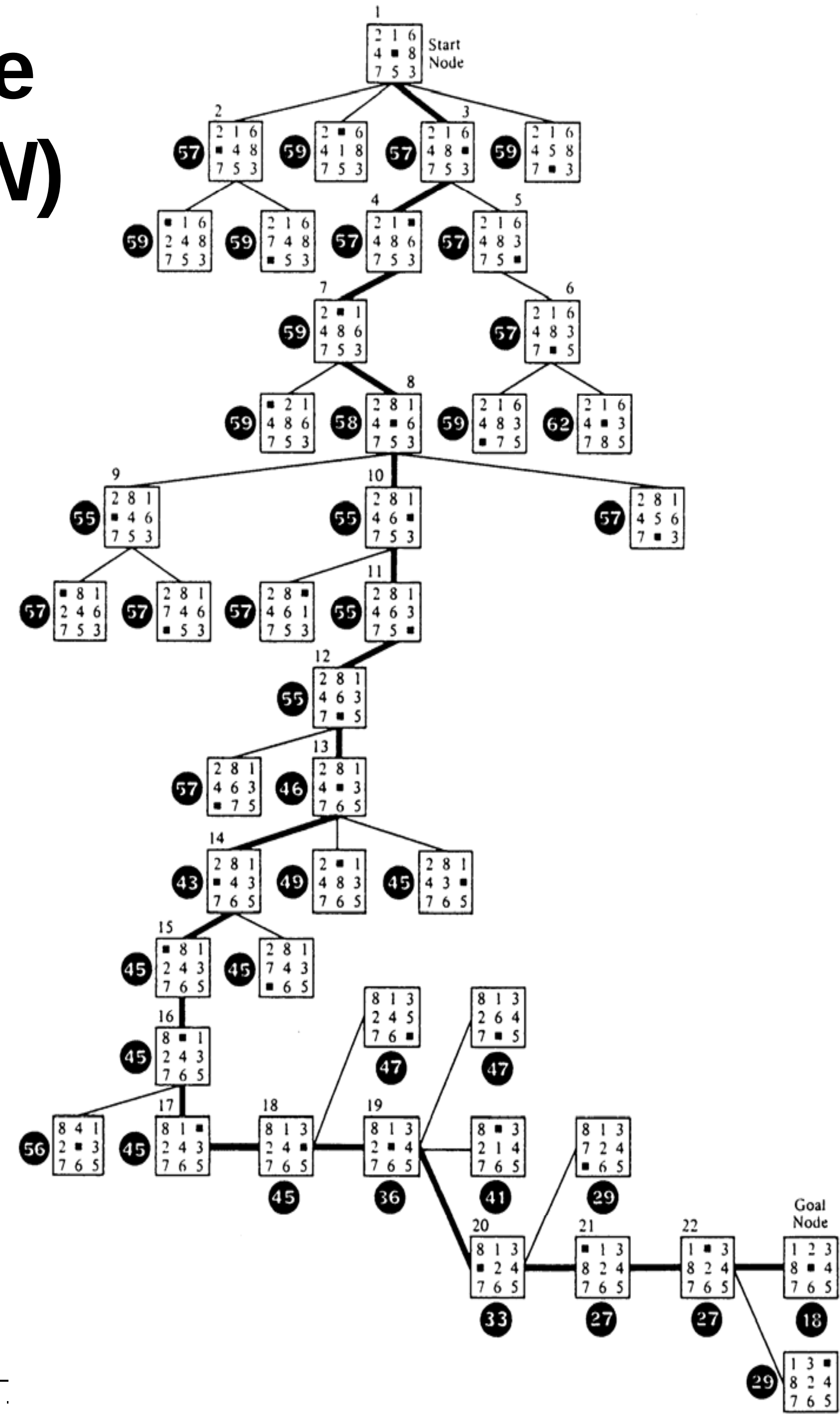
S(*N*): Für alle Kacheln, die nicht in der Mitte liegen:
Wert := 2, falls Kachel nicht neben ihrem richtigen Nachfolger liegt
Wert := 0, sonst

Für Kacheln in der Mitte gilt:
Wert := 1

Bsp:

1	3	
8	2	4
7	6	5

P(*N*) = 2
S(*N*) = 2 + 1

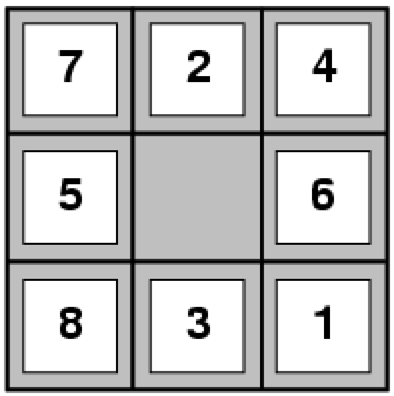


Andere zulässige Heuristiken

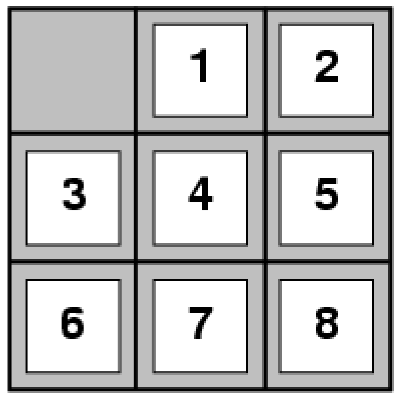
Für 8er-Puzzle:

- $h_1(n)$ = Anzahl der Kacheln, die nicht am Platz sind
- $h_2(n)$ = Summe der Manhattan-Distanzen

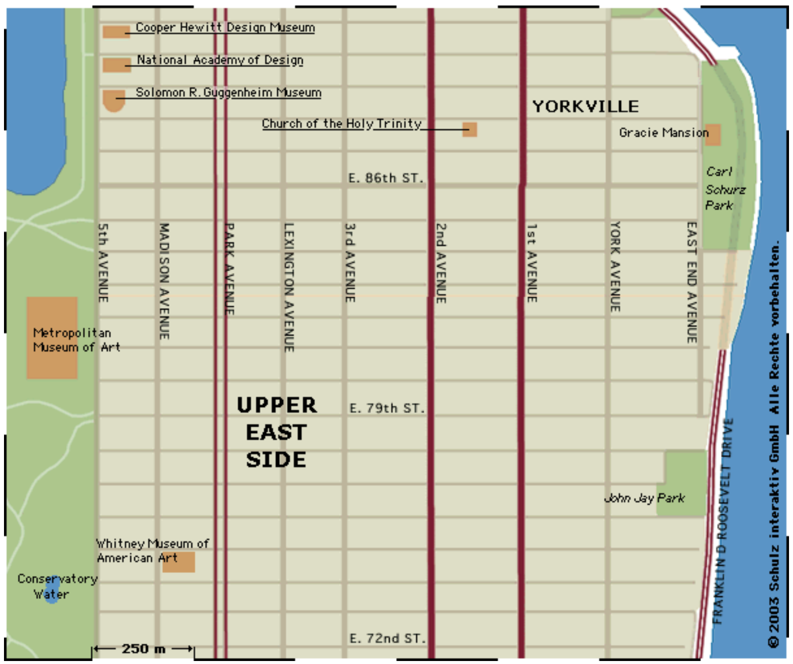
(= Abstand jeder Kachel zu ihrem Zielplatz)



Start State



Goal State



- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+2+3+3+2 = 18

Leistungsmaße für Graph Search

1. **Penetranz**

   $P = L/T$

   L = Länge des Lösungspfades

   T = Anzahl der insgesamt expandierten Knoten

   ‘Keine Lösung‘ $0 <= P <= 1$ ‘Zielstrebig‘

   ‘Buschiger Baum‘ ‘Schlanker Baum‘

2. **Effektiver Verzweigungsfaktor**

   $B + B^2 + \cdots + B^L = T$

   $B(B^L – 1)/(B – 1) = T$

   Keine explizite Lösung für B, aber B immer >= 1

   B = 1: Es werden nur Knoten auf dem Lösungspfad expandiert.

   Kleines B: Schlanker Baum Großes B: Buschiger Baum

Merksatz: Mehr Wissen bedeutet weniger Suchaufwand.

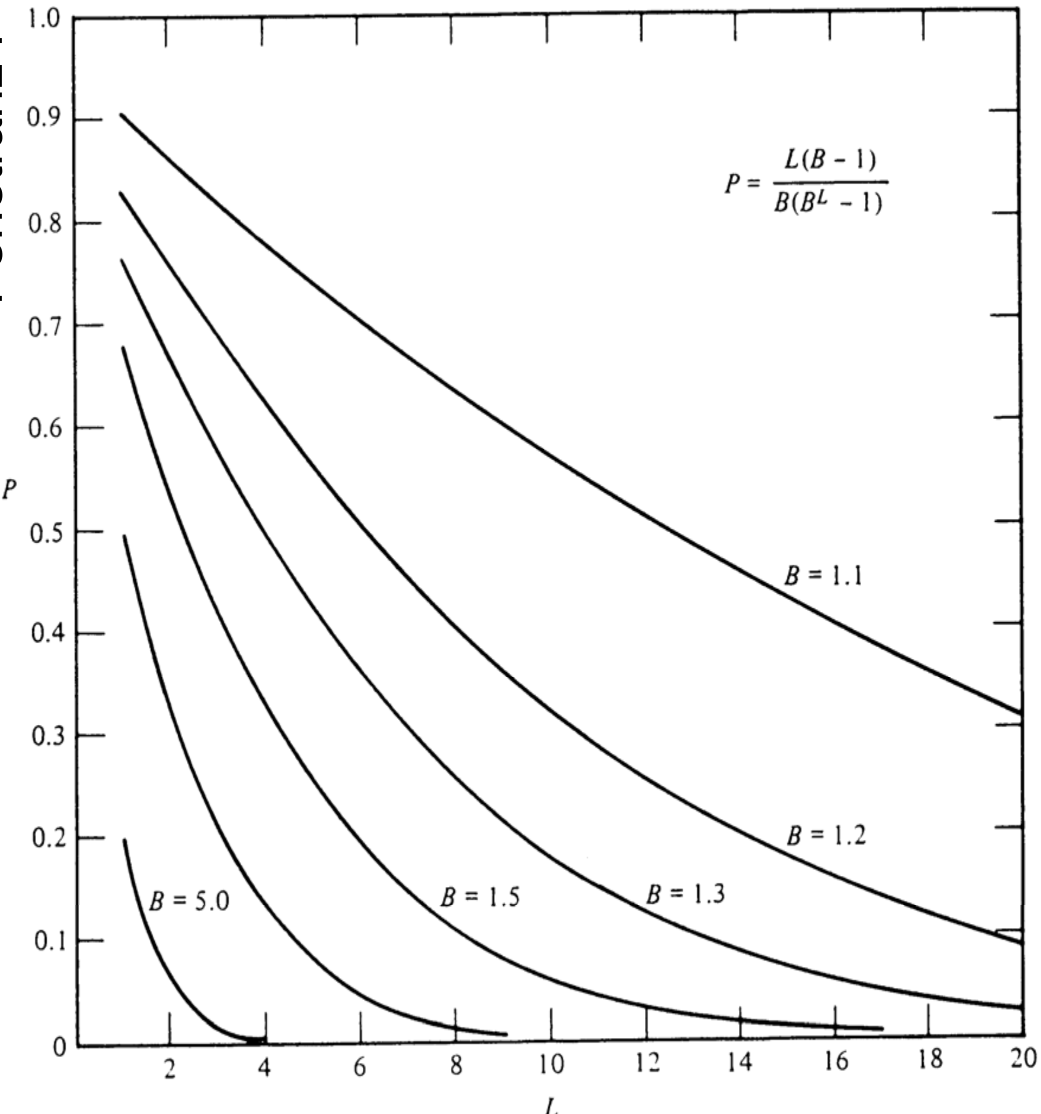
P zu L für verschiedene Werte von B

P = L/T

T := Gesamtzahl expandierter Knoten

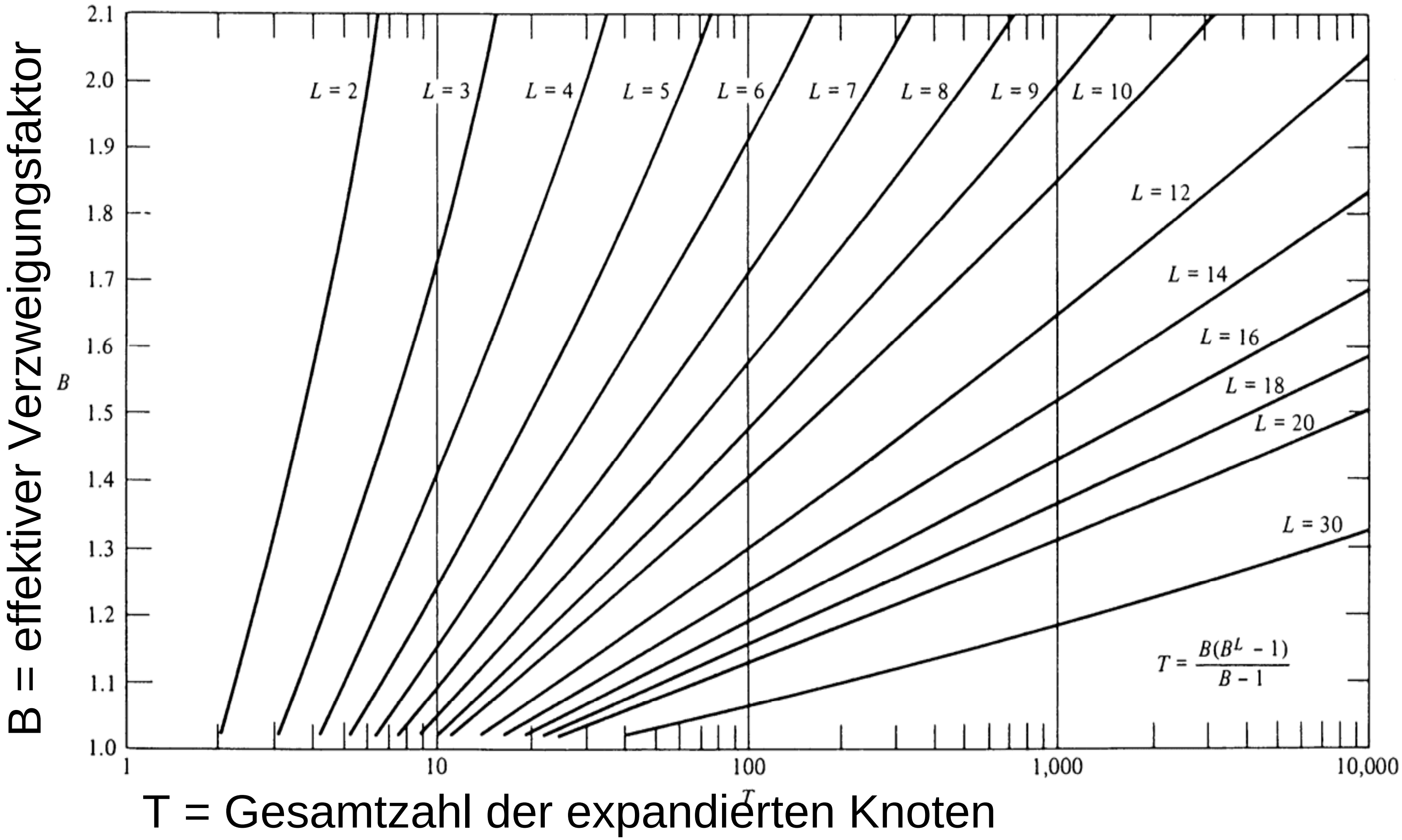
B := effektiver Verzweigungsfaktor

Penetranz P



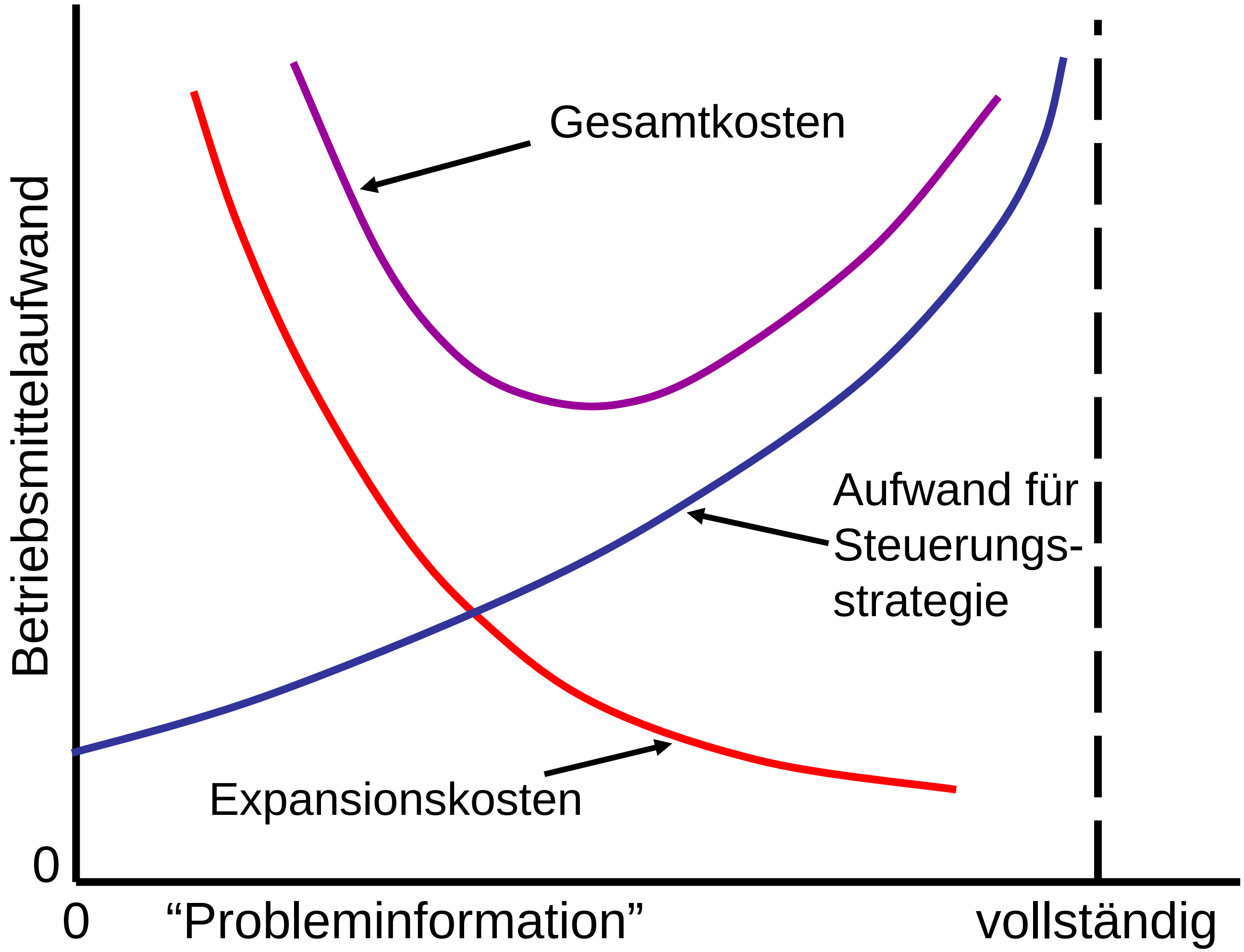
Länge des Lösungspfades L

B zu T für verschiedene Werte von L

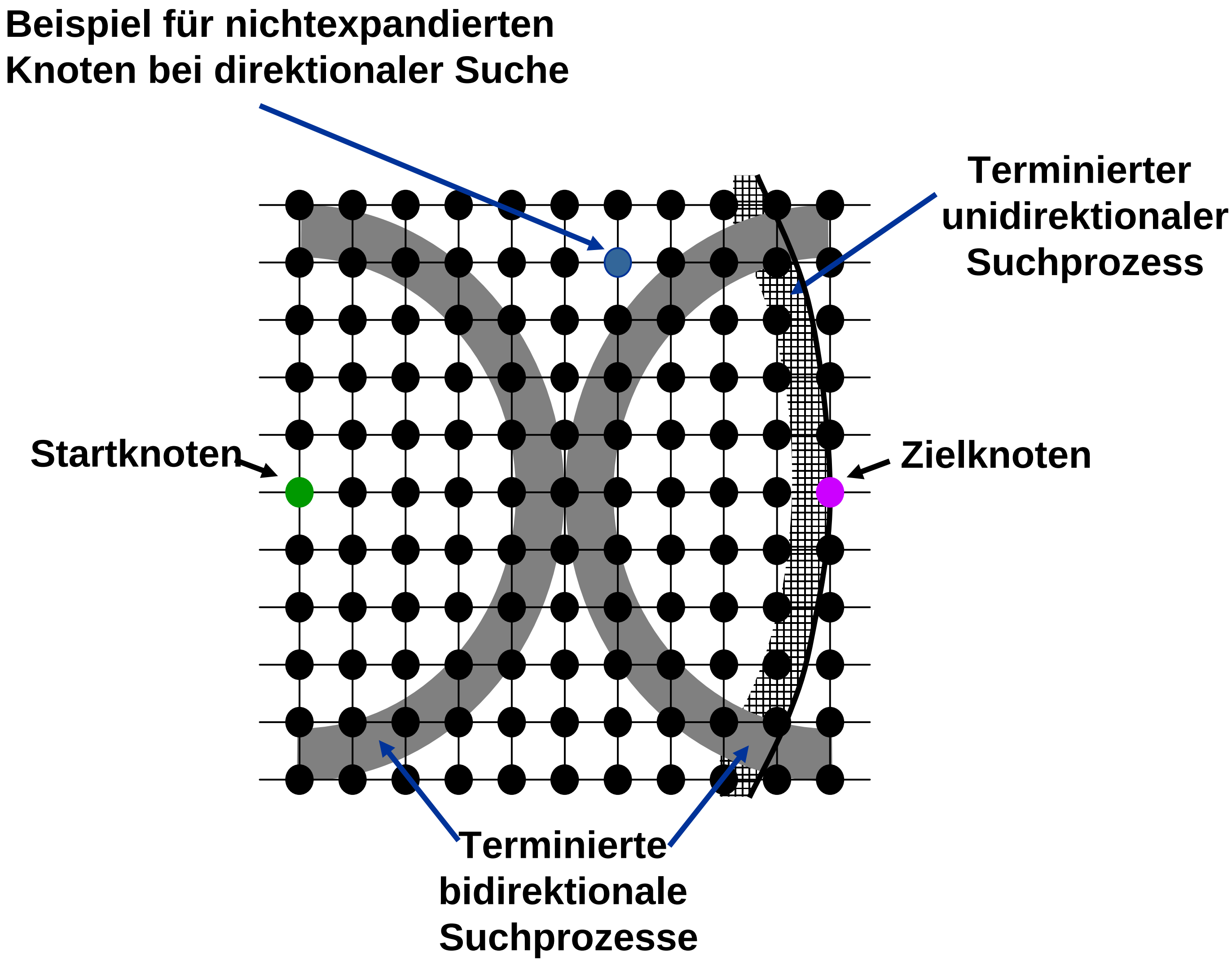


Für 8-er Puzzle mit f(*N*) = g(*N*) + P(*N*) + 3 S(*N*) ergibt sich B = 1.08.
Bei 30-er Schrittlösung werden 120 Knoten expandiert.

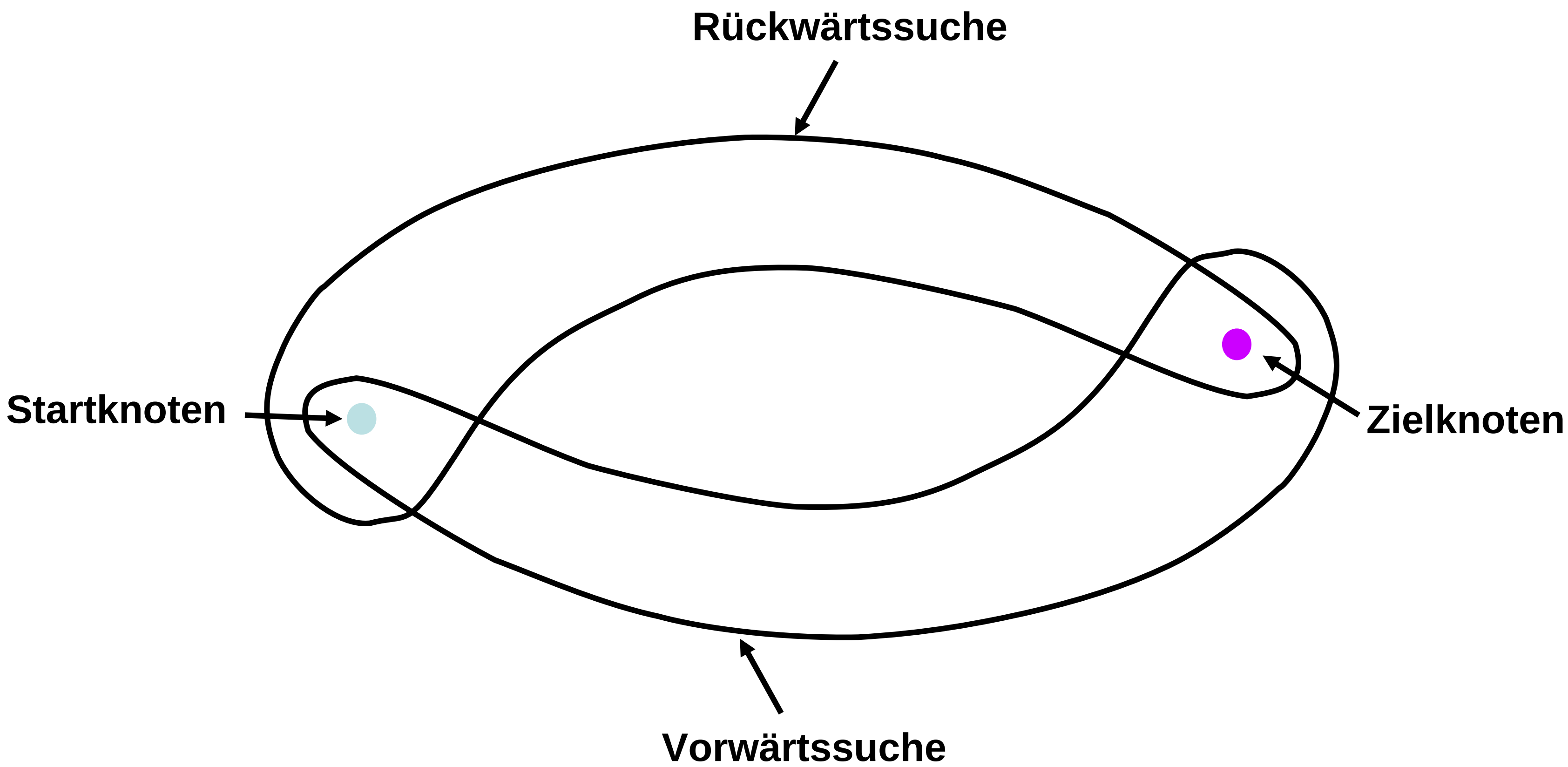
Aufwandsreduktion durch Berücksichtigung zusätzlicher Probleminformation



Effizienzgewinn durch bidirektionale Breitensuche



Effizienzverlust bei heuristischer bidirektionaler Suche mit ungünstiger Bewertungsfunktion



Das Minimax-Verfahren

Bei Problemen, in denen *nichtkooperative Aktoren* beteiligt sind (klassisches Beispiel: Gesellschaftsspiele wie Schach, Go, Tic-Tac-Toe), ist der Suchgraph meist extrem groß.

z.B.: Schach, Verzweigungsfaktor: ca. 35,
Züge pro Spieler: ca. 50;
Suchgraph hätte also ca. 35^{100} Knoten.

daher: Zu Beginn eines Spiels ist sogar eine heuristische Suche nach einem vollständigen Lösungsweg aussichtslos.

Ziel der Suche daher:

Bewertung des nächsten Handlungsschrittes

Allgemeines Vorgehen:

Vorausschau und *Bewertung* einiger Züge, für die eine Expansion des Suchgraphen vorgenommen wird.

Das Minimax-Verfahren

- Zwei Spieler Max und Mini, *statische Bewertungsfunktion* für Knoten K des Suchgraphen sei B(K), Knoten entspricht z.B. Brettposition bei Spielen.
  - Falls K Vorteil für Max: B(K) > 0 (Max sucht maximalen Wert)
  - Falls K Vorteil für Min: B(K) < 0 (Mini sucht minimalen Wert)
  - Bei ‘ausgeglichener‘ Position gilt: B(K) etwa = 0
- Max soll als nächster agieren (wählt höchstbewerteten Knoten).
- Expandiere Suchgraph bis zur Tiefe $D = 2N - 1$, $N >= 2$.
- Bewerte alle Blätter mit der statischen Bewertungsfunktion B.
- Verteile die Bewertungen nach folgendem Schema weiter zu den Vorgängern:
  - Knoten der Tiefe $D - 1$ erhalten jeweils das Minimum der Bewertungen ihrer Nachfolger (‘Min-Zug’).
  - Knoten der Tiefe $D - 2$ erhalten jeweils das Maximum der Bewertungen ihrer Nachfolger (‘Max-Zug’).
  - Knoten der Tiefe 0 erhalten das Maximum.

Das Minimax-Verfahren

- Der Wechsel zwischen der Wahl von Knoten mit minimaler und maximaler Bewertung gab dem Minimax-Verfahren seinen Namen.

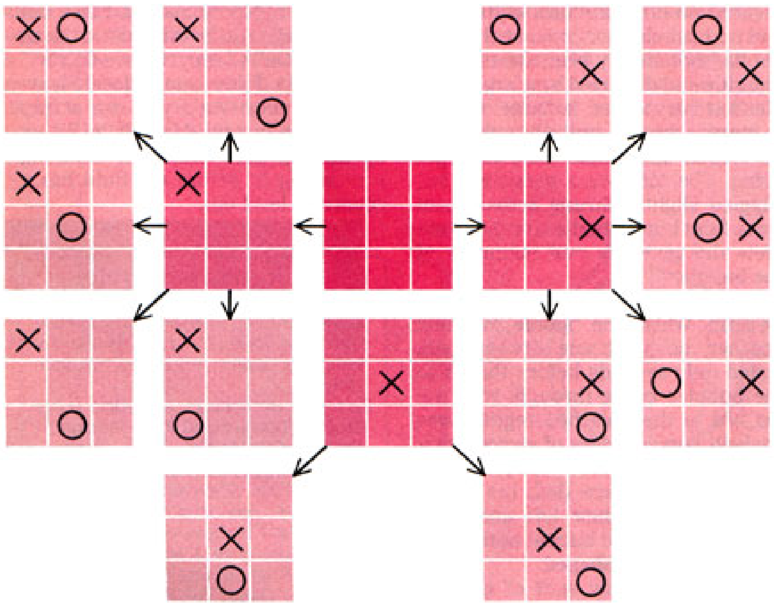
- Annahmen bei Minimax:
  – Die durch die Vorausschau erhaltene Bewertung der Nachfolger des Startknotens ist zuverlässiger als die durch direkte Bewertung der Nachfolger mit B erhaltenen Hinweise.
  – Der jeweilige Gegenspieler entscheidet sich immer rational für den bestbewerteten Zug nach dem Minimax-Verfahren.

Das Minimax-Verfahren für Tic-Tac-Toe

- Statische Bewertungsfunktion:

B(*K*) = <Anzahl der noch offenen Reihen/Spalten/Diagonalen für Max bei K>
- <Anzahl der noch offenen Reihen/Spalten/Diagonalen für Mini bei K>

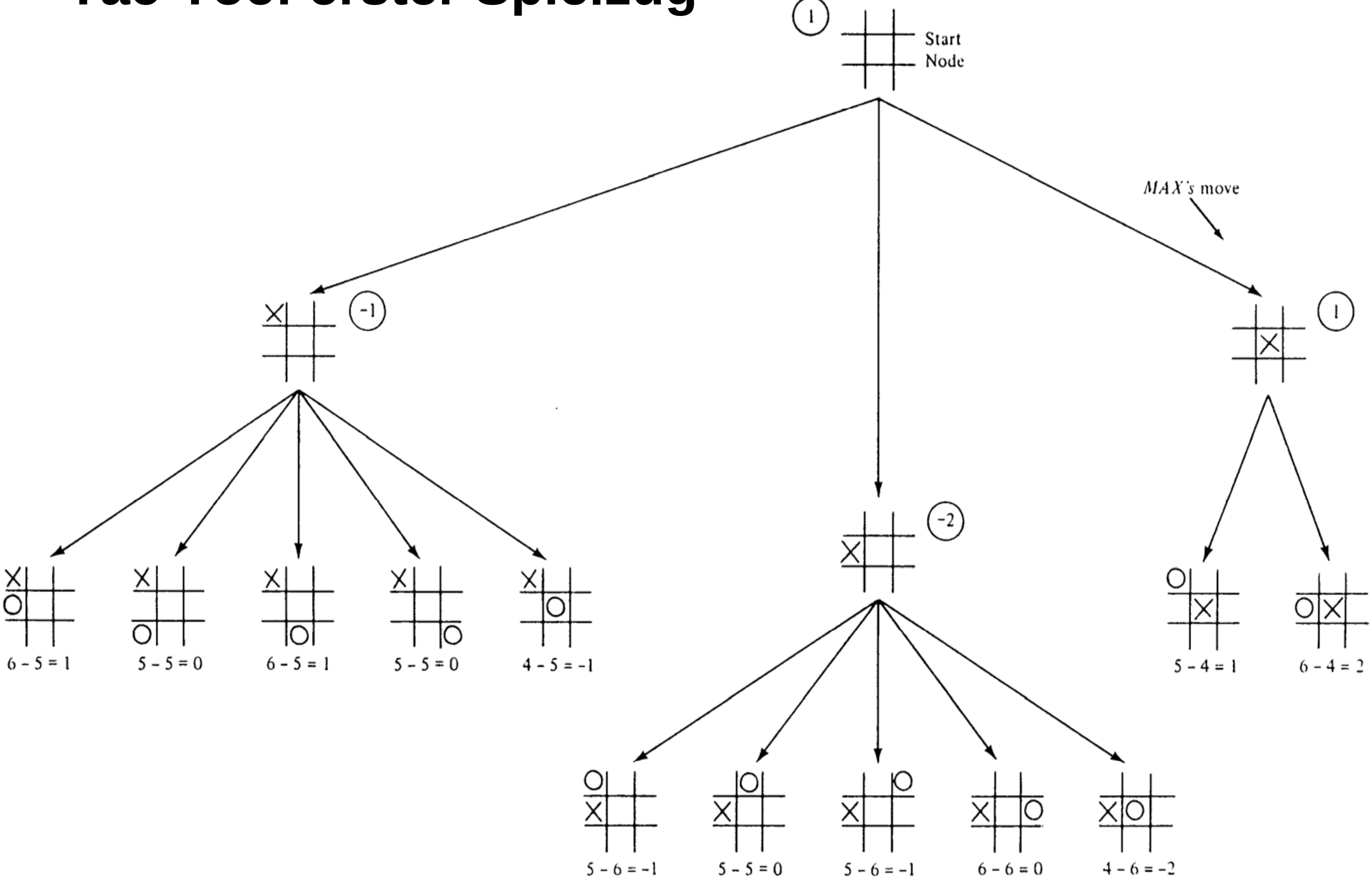
- Durch Symmetrien (Drehung und Spiegelung am Gitter), bleibt der Verzweigungsfaktor zu Beginn klein.
- Später bleibt er durch die verringerte Zahl der noch offenen Gitterplätze klein.
- Im Beispiel: Breitensuche bis alle Knoten der Tiefe 2 expandiert sind.



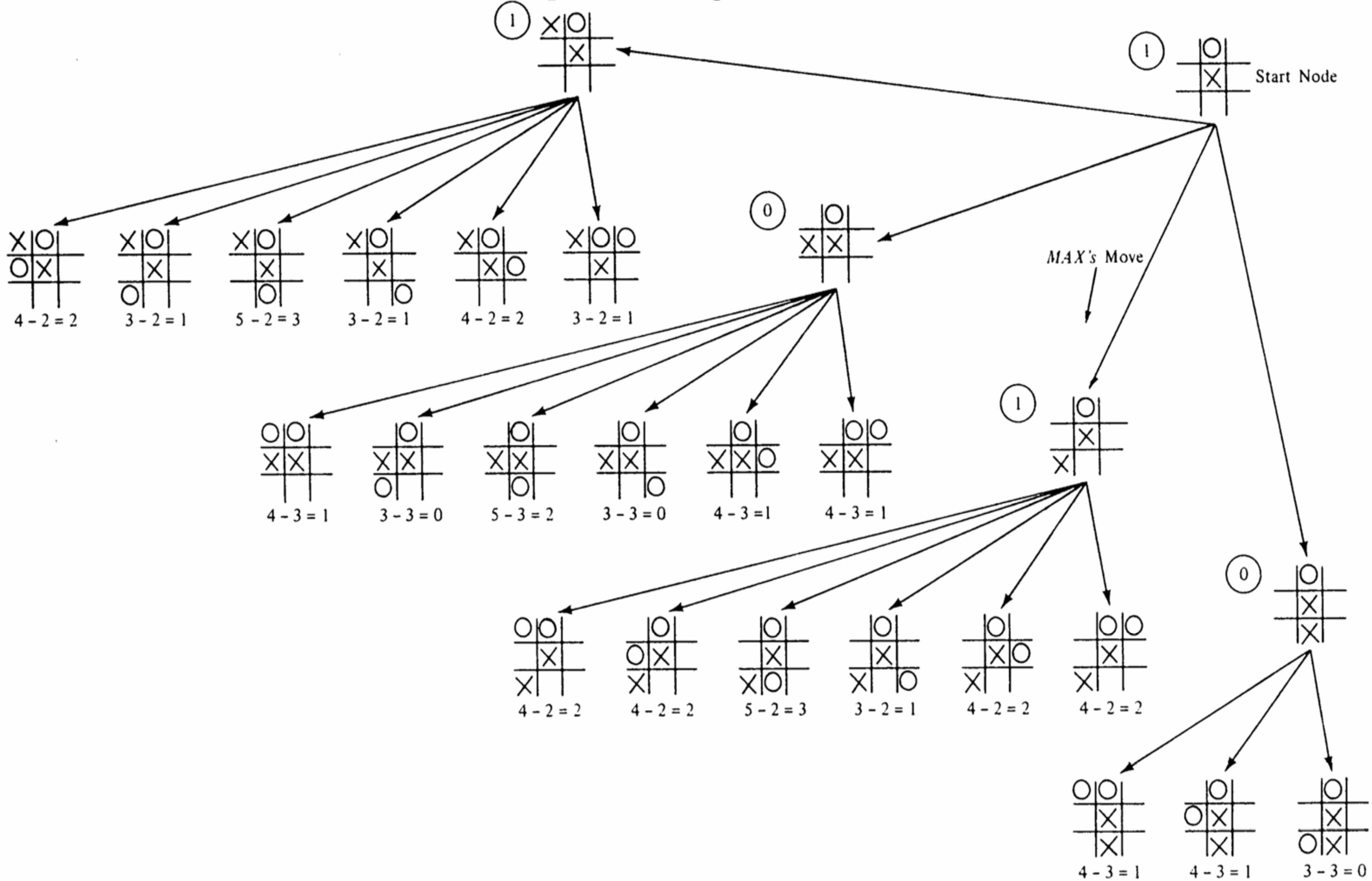
Das Minimax-Verfahren für Tic-Tac-Toe

- Nachteile:
- Eine *Zahl* als Positionsbewertung ist *wenig aussagekräftig*; sie sagt nichts darüber aus, wie sie entstanden ist.
- Hoher Aufwand von Minimax:
  - *Alle möglichen Nachfolger* werden erzeugt.
  - Für *alle möglichen Pfade* wird erst statische *Bewertungsfunktion* berechnet.

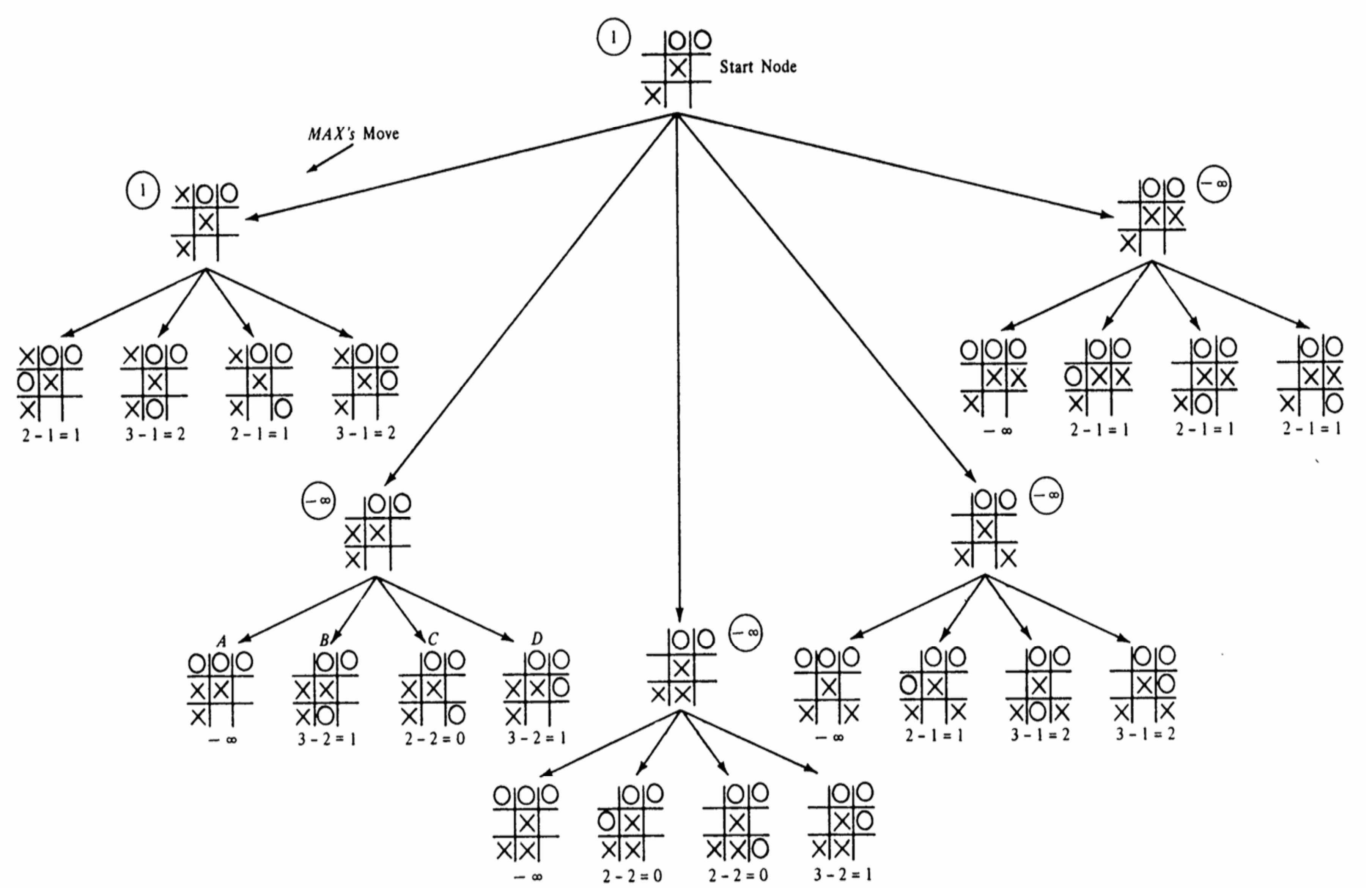
Das Minimax-Verfahren am Beispiel von Tic-Tac-Toe: erster Spielzug



Das Minimax-Verfahren am Beispiel von Tic-Tac-Toe: zweiter Spielzug

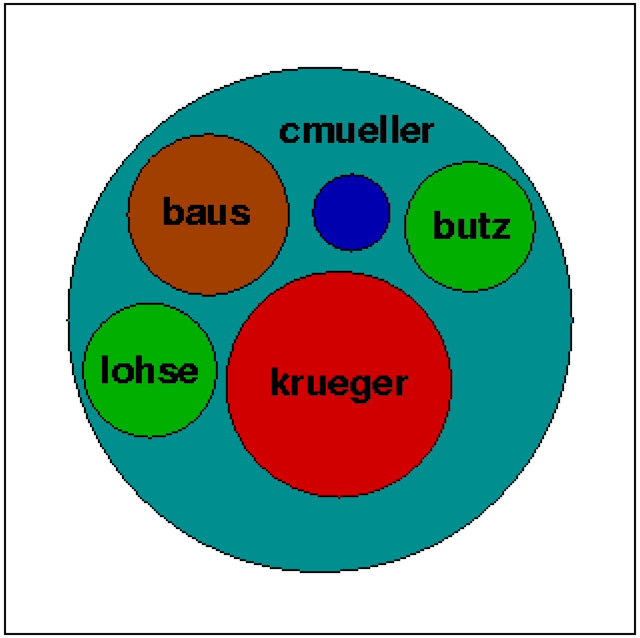


Das Minimax-Verfahren am Beispiel von Tic-Tac-Toe: dritter Spielzug



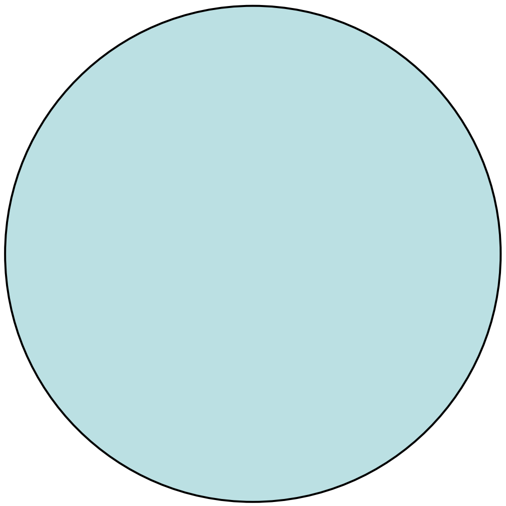
Beispiel: Kreisdiagramm

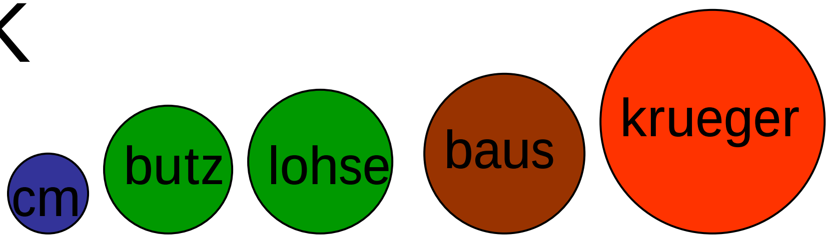
1448004	krueger
1136311	baus
960804	butz
909182	lohse
645718	cmueller
282454	detlev
262432	cray
234793	jameson
141997	kern
134568	roquas
128748	bdecker
120860	columbus
115924	bohne
107541	florian
99645	brueck



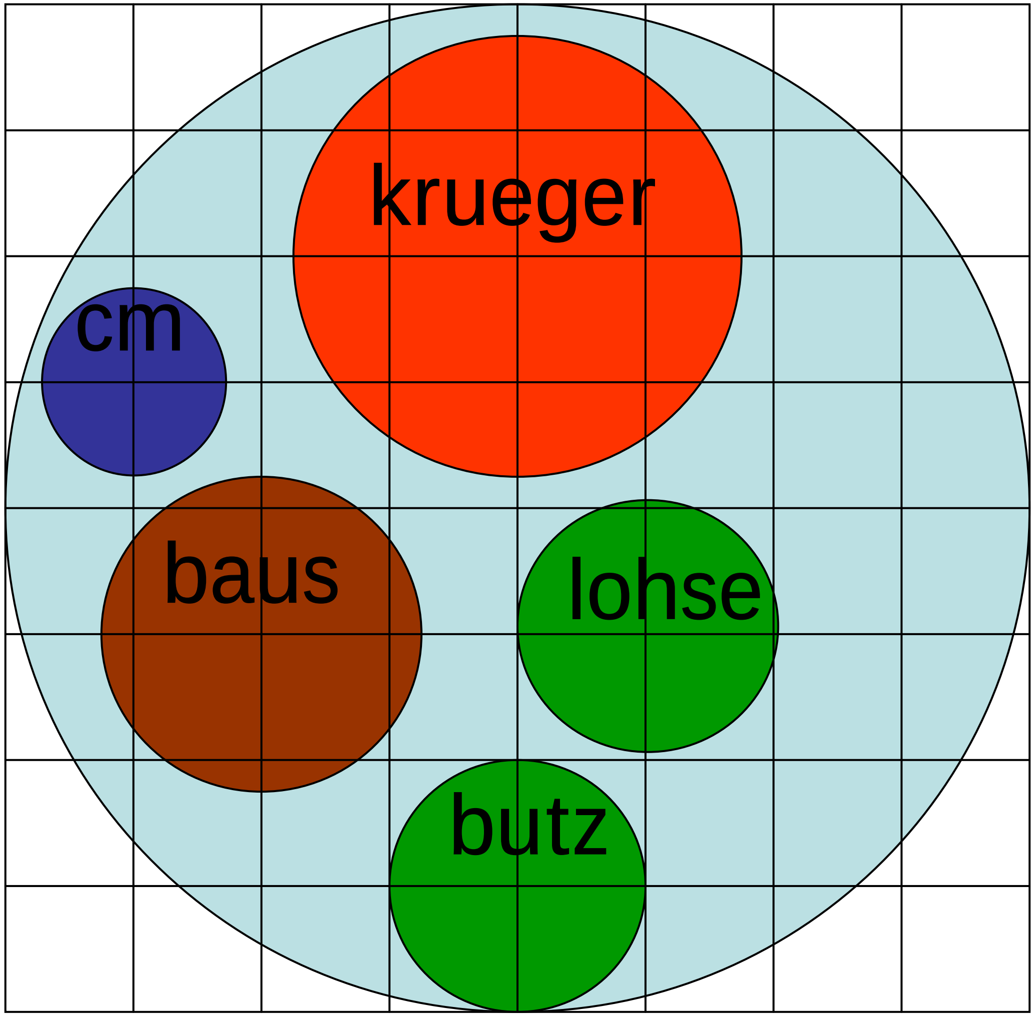
Grid Layout (1)

- gleichmäßiges Gitter
- Diskretisierung des Suchraumes
- zeilenweise absuchen
- „dummes“ Verfahren
- Ergebnisse trotzdem einigermaßen OK





Grid Layout (2)



Verbesserung: Heuristiken

- größtes Element zuerst
- kleinstes Element zuerst
- in der Mitte anfangen
- ...???
- anderes Gitter? (z.B. Dartscheibe)
- alternative Positionen merken und Backtracking anwenden
- ...???

Literatur, Links

- Günther Görz (Hrsg.): Einführung in die künstliche Intelligenz, Addison-Wesley (1993), Bonn, ISBN 3-89319-507-6
- Stuart Russell und Peter Norvig: Künstliche Intelligenz, ein moderner Ansatz, Prentice Hall (2004), München, ISBN 3-8273-7089-2
- http://w5.cs.uni-sb.de/teaching/ws0506/KI/
  (daraus auch wesentliche Teile der heutigen Vorlesung)