



Hochschule für Angewandte  
Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

# Intelligente Systeme

– Suchen –

**Prof. Dr. Michael Neitzke**

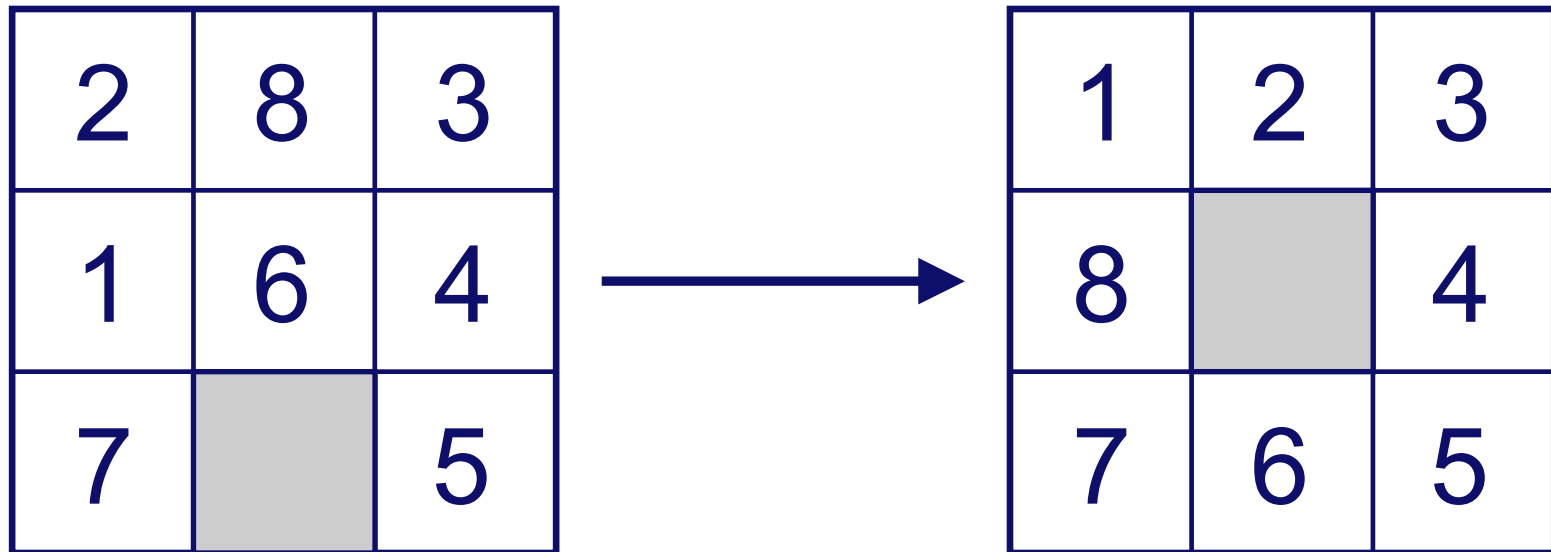
# S1: Uninformierte Suche

- Was ist Suchen?
- Definition: Zustandsraum
- Uninformierte Suchverfahren

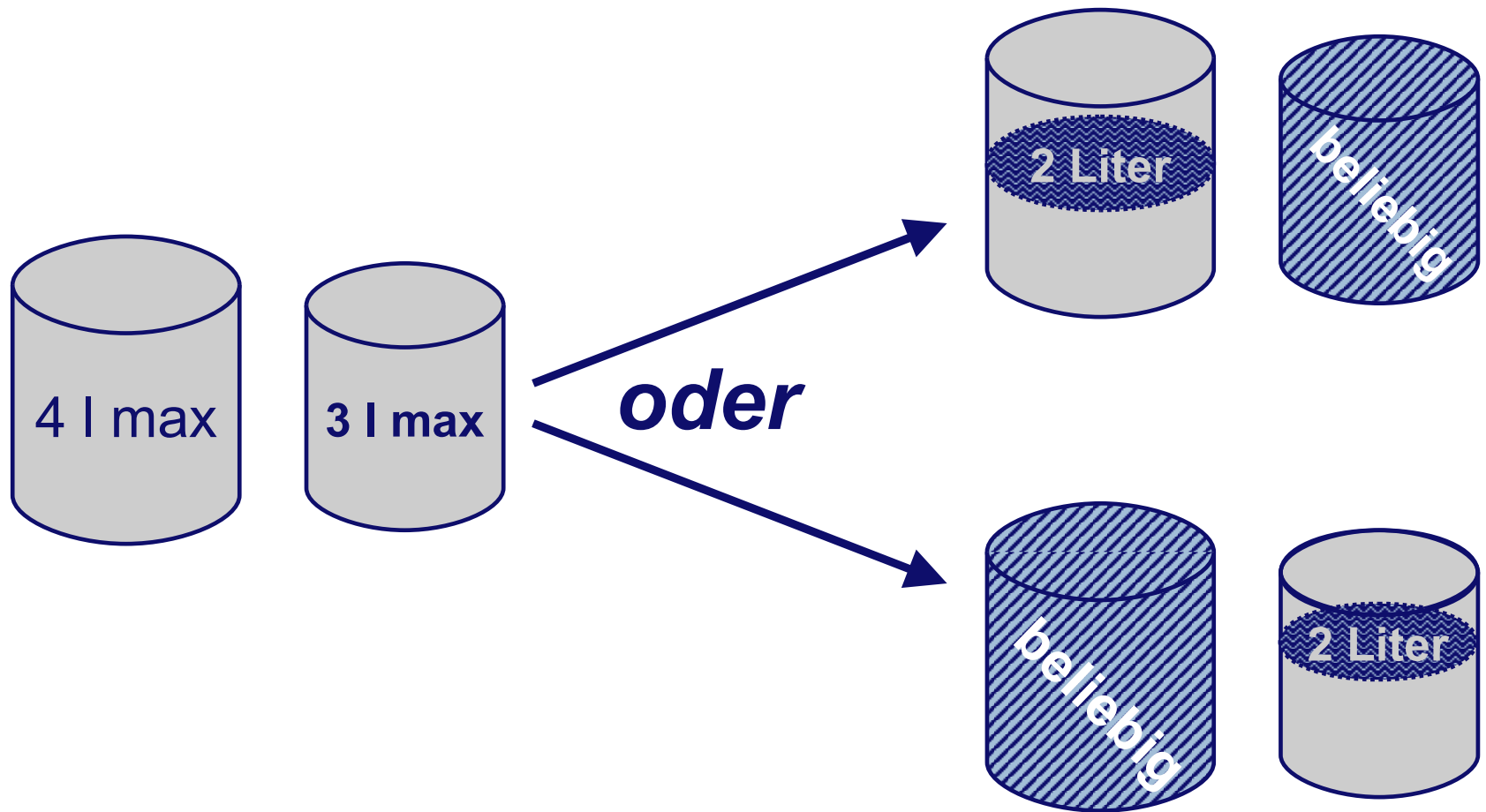
# S1: Lernziele

- V1: Erläutern können, was Suchen kennzeichnet
- V2: Eigenschaften, Vor- und Nachteile der verschiedenen uninformatierten Suchverfahren erklären können

# Problemlösen: (Standard-) Beispiel: 8-Puzzle



# Beispiel: Wassergefäße



# Beispiel: Sudoku

6			7			5		
	2	8						
			6	4		3		
7	4						2	
		1				8		
	5						3	7
		3		7	6			
						1	9	
		4			5			8

***z. B.*** →

6	3	9	7	1	8	5	4	2
4	2	8	5	3	9	7	1	6
1	7	5	6	4	2	3	8	9
7	4	6	8	5	3	9	2	1
3	9	1	4	2	7	8	6	5
8	5	2	9	6	1	4	3	7
9	8	3	1	7	6	2	5	4
5	6	7	2	8	4	1	9	3
2	1	4	3	9	5	6	7	8

# Diskutieren Sie!

- Welche Konzepte werden benötigt, um die drei Beispiel-Probleme zu lösen?
- Es sind zwei unterschiedliche Arten von Lösungen aufgetreten. Welche beiden?

- Konzepte:
  - Zustände mit geeigneter Zustandsbeschreibung
  - Zustandsübergänge
  - Start- und Zielzustände
- Zwei Arten von Lösungen:
  - Eigenschaften eines Zielzustands
  - Weg zu einem Zielzustand



# Definition Zustandsraum

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

**N:** Menge der Knoten des Zustandsraumgraphen: Zustände

**A:** Menge der Kanten des Zustandsraumgraphen:  
Zustandsübergänge, Schritte des Problemlösungsprozesses

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

**G:** nicht-leere Teilmenge von N: Zielzustände, beschrieben durch

- Eigenschaften des Zustandes, oder
- Eigenschaften des Pfades vom Startzustand zum Zielzustand

# Repräsentation von Zuständen

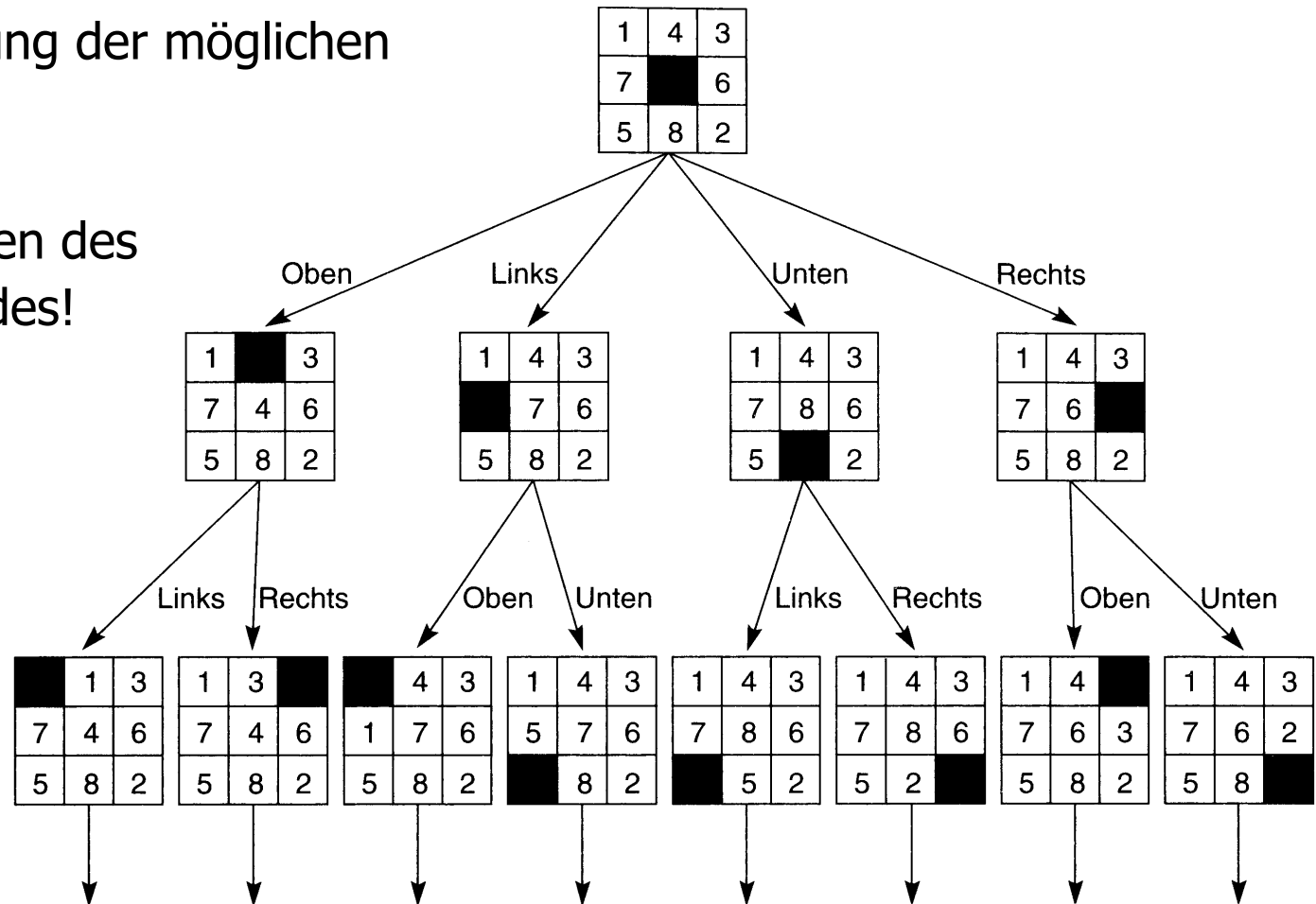
- Welche Eigenschaften sollen repräsentiert werden?
- Wie sollen diese Eigenschaften repräsentiert werden?
  - Dies hat großen Einfluss auf Effizienz und Klarheit des Lösungswegs.
- Beispiel: Mögliche Repräsentationen für das 8-Puzzle
  - $[2,8,3,1,6,4,7,0,5]$
  - $[[2,8,3], [1,6,4], [7,0,5]]$
  - $((2,8,3),(1,6,4),(7,0,5))$
  - $(\text{pos}(1,\text{ml}), \text{pos}(2,\text{ol}), \text{pos}(3,\text{or}), \dots)$
  - $(\text{ml}, \text{ol}, \text{or}, \text{mr}, \text{ur}, \text{mm}, \text{ul}, \text{om})$

2	8	3
1	6	4
7		5

# Zustandsübergänge für 8-Puzzle

- Modellierung der möglichen Aktionen:

Verschieben des freien Feldes!



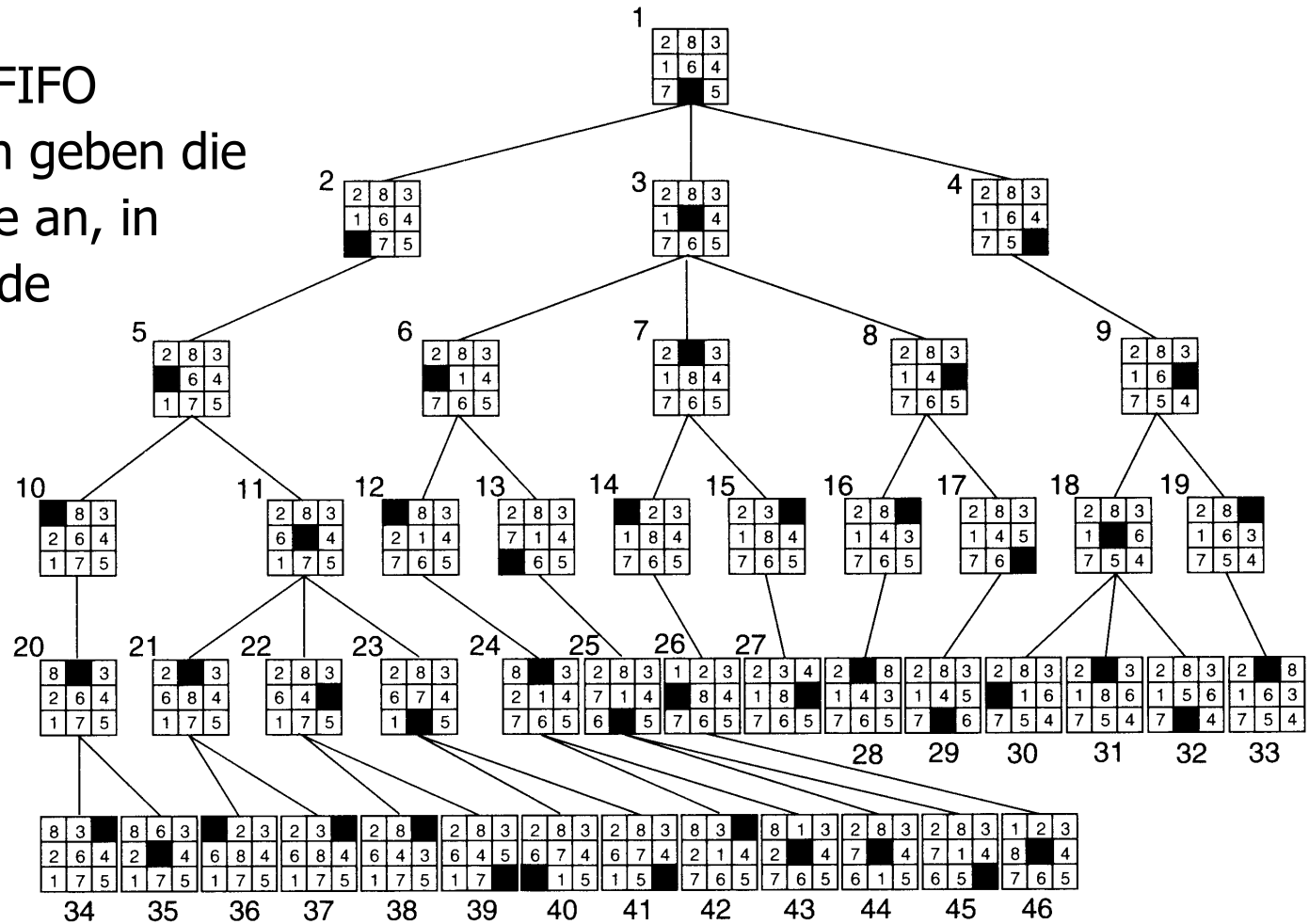
# Diskutieren Sie!

- Welche Suchverfahren kennen Sie bereits?
- Was sind die Unterschiede?
- Welche Entscheidungsfreiheiten hat man grundsätzlich, um verschiedene Suchverfahren zu definieren?

- Wenn man sich entlang der Kanten zwischen den Zuständen durch den Zustandsraum bewegen will, hat man nur die Entscheidungsfreiheit, welchen Zustand des bisher erforschten Zustandsraum man als nächstes expandieren will. D. h. man beginnt mit einem Startzustand oder auch einem Zielzustand und erzeugt die Nachbarn. Dann entscheidet man, von welchem Zustand als nächstes die Nachbarn erzeugt werden. Bei dieser Vorgehensweise kann man zwischen drei Mengen von Zuständen unterscheiden:
  - Den bereits expandierten Zuständen, der so genannten Closed List
  - Den noch nicht expandierten, aber bereits erzeugten Zuständen, der so genannten Open List. Im Falle eines Suchbaums sind dies die Blattknoten.
  - Den noch nicht „entdeckten“ Zuständen des Zustandsraums.
- Alternativ könnte man für bestimmte Aufgabenstellungen auch z. B. durch Zufallsprozesse Zustände des Zustandsraums ohne Berücksichtigung der Übergänge erzeugen. Aber diese stochastische Form der Suche wird in dieser Veranstaltung nur am Rande betrachtet.

# Breitensuche

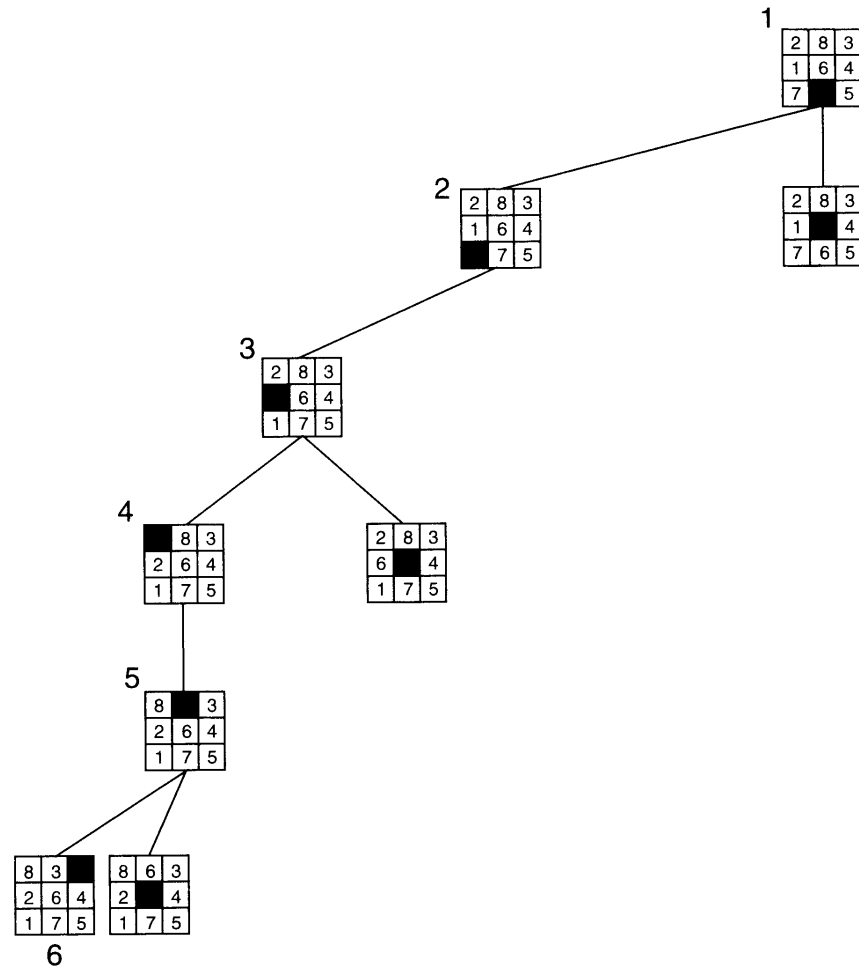
- Strategie: FIFO  
(Die Zahlen geben die Reihenfolge an, in der Zustände expandiert werden.)



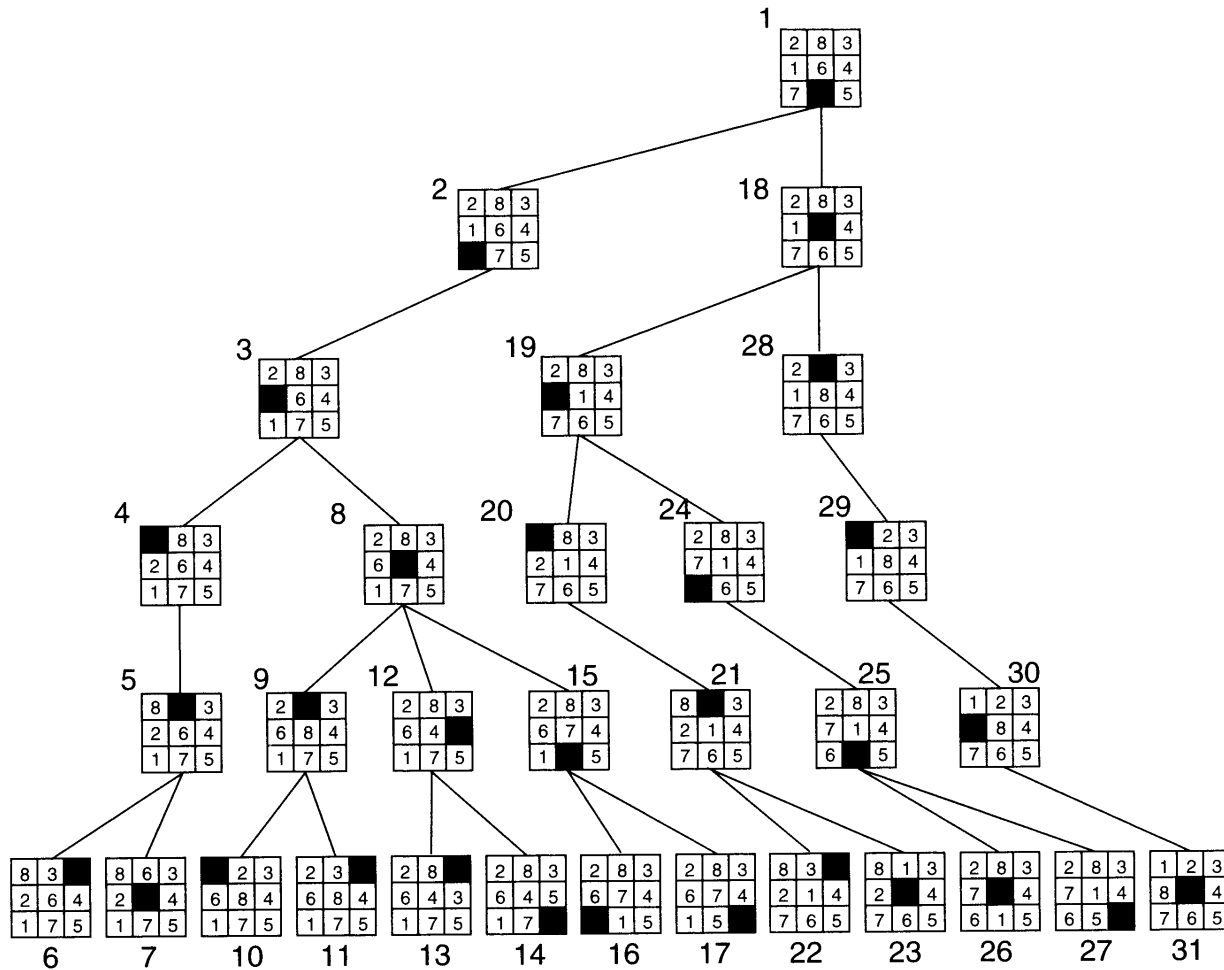
Ziel

# Tiefensuche

## ■ Strategie: LIFO



# Tiefensuche mit Tiefenbeschränkung von 5



Ziel



# Diskutieren Sie!

- Welche Vor- und Nachteile sehen Sie für Breiten- und Tiefensuche?

# Bewertung Breitensuche

- Wenn Lösung existiert, wird sie gefunden
- Kürzester Weg wird gefunden
- 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 Verzweigungsrate  $v$  und Tiefe des Zielknotens  $t$ :

$$v + v^2 + v^3 + \dots + (v^{t+1} - v) = O(v^{t+1})$$

- Worst Case Annahme: Zielknoten befindet sich unten rechts
  - Startknoten wird nicht erzeugt
  - Startknoten auf Ebene 0
  - Erst Zieltest, dann Kinder erzeugen. Kinder zunächst nicht untersuchen
- 
- $O(v^{t+1})$  betrifft Speicher- und Zeitbedarf

# Beispiel für Zeit-/Speicherbedarf der Breitensuche

- Annahmen:
  - Verzweigungsrate 10
  - Zeitbedarf 10.000 Knoten / Sekunde
  - Speicherbedarf 1000 Byte / Knoten

Tiefe	Knoten	Zeit	Speicher
2	1100	11 Sekunden	1 Megabyte
4	111100	11 Sekunden	100 Megabyte
6	$10^7$	19 Minuten	10 Gigabyte
8	$10^9$	31 Stunden	1 Terabyte
10	$10^{11}$	129 Tage	100 Terabyte
12	$10^{13}$	35 Jahre	10 Petabyte
14	$10^{15}$	3523 Jahre	1 Exabyte

# Überlegen Sie!

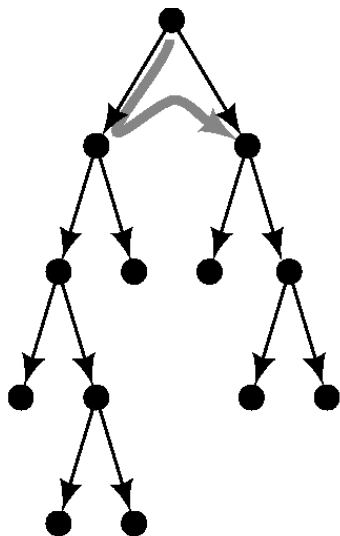
- Welchen Zeit- und Speicherbedarf hat eine Tiefensuche?

# Bewertung Tiefensuche

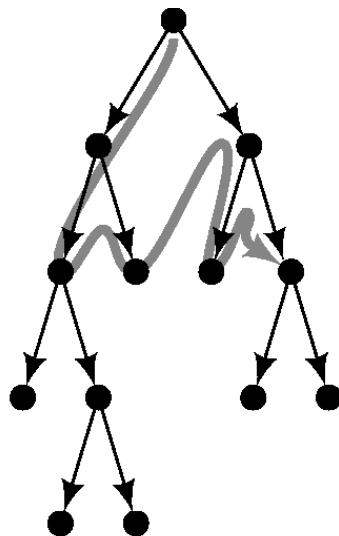
- Geringer Ressourcenbedarf
- Lösung kann verfehlt werden
- Speicherbedarf:  $v * t_{\max}$
- Zeitbedarf:
  - $O(v^{t_{\max}})$
  - da im schlimmsten Fall alle Knoten bis zur Tiefe  $t_{\max}$  expandiert werden

# Schrittweise / Iterative Vertiefung

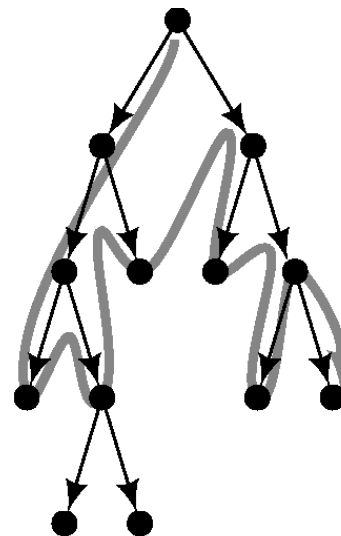
- Tiefensuche mit Beschränkung
- In jeder Iteration wird Tiefe um 1 erhöht
- Jede Iteration beginnt komplett von vorn
- Englischer Begriff: Iterative Deepening



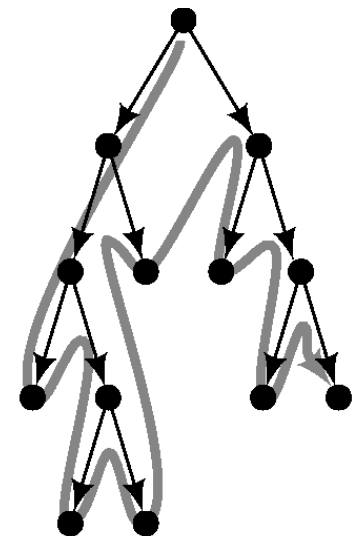
Depth bound = 1



Depth bound = 2



Depth bound = 3



Depth bound = 4

# Diskutieren Sie!

- Wie bewerten Sie die iterative Vertiefung?



# Bewertung Iterative Vertiefung

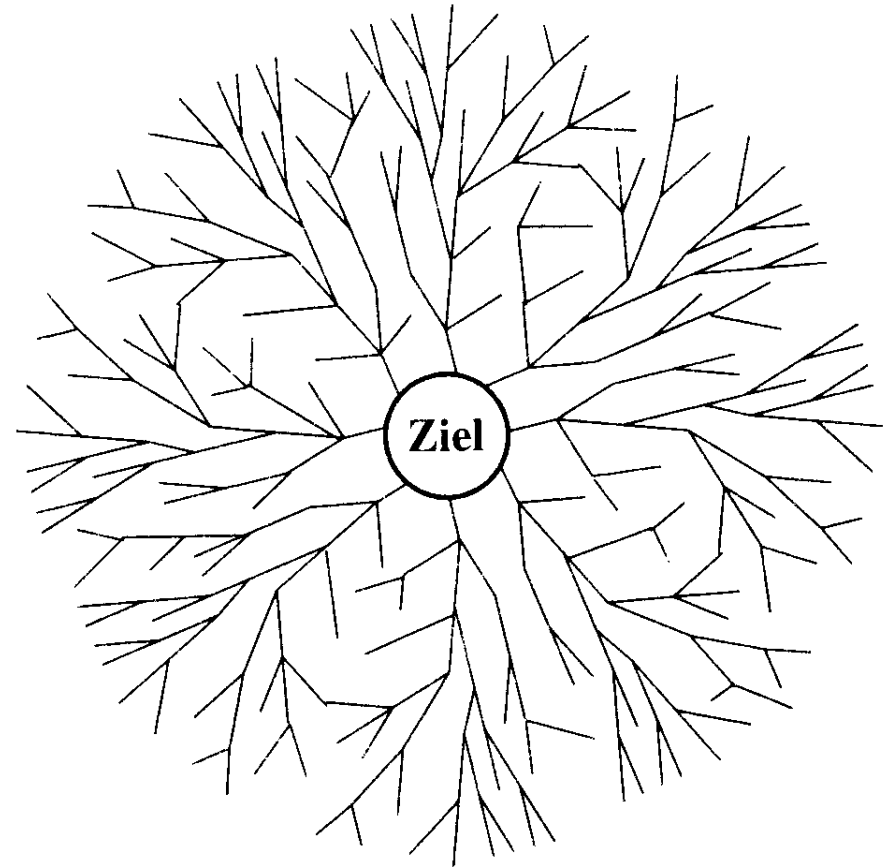
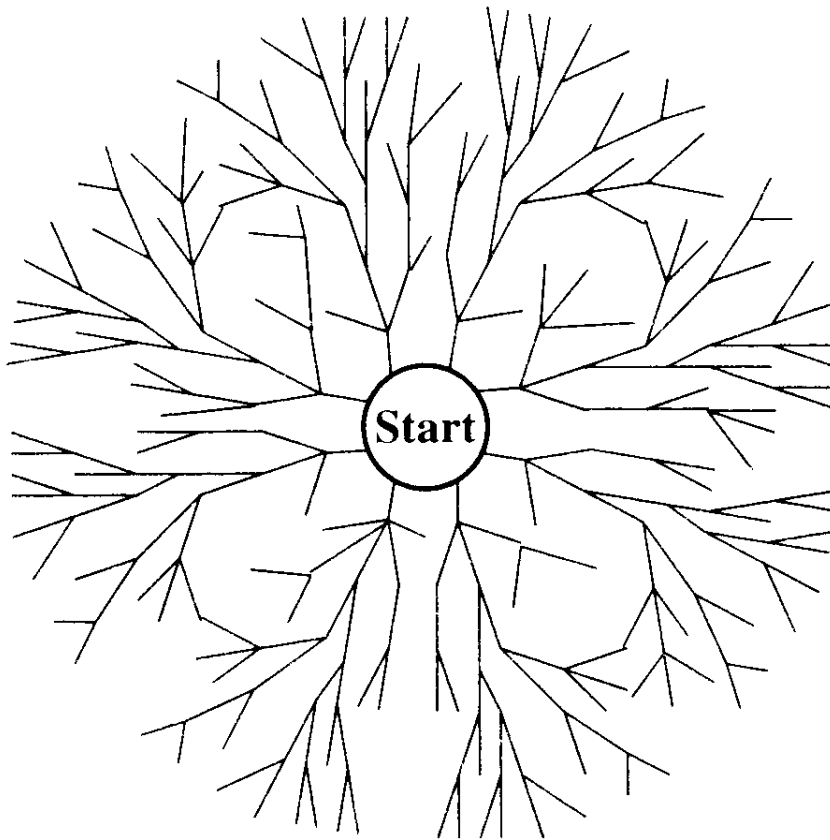
- Speicherbedarf wie Standard-Tiefensuche
- Anzahl erzeugter Zustände insgesamt:

$$t * v + (t-1) * v^2 + (t-2) * v^3 + \dots + 1 * v^t = O(v^t)$$

- betrifft Zeitbedarf
- Wiederholungen schlagen kaum zu Buche
- Merkmale von Tiefen- und Breitensuche kombiniert
  - geringer Speicherbedarf
  - Lösung wird gefunden

# Bidirektionale Breitensuche

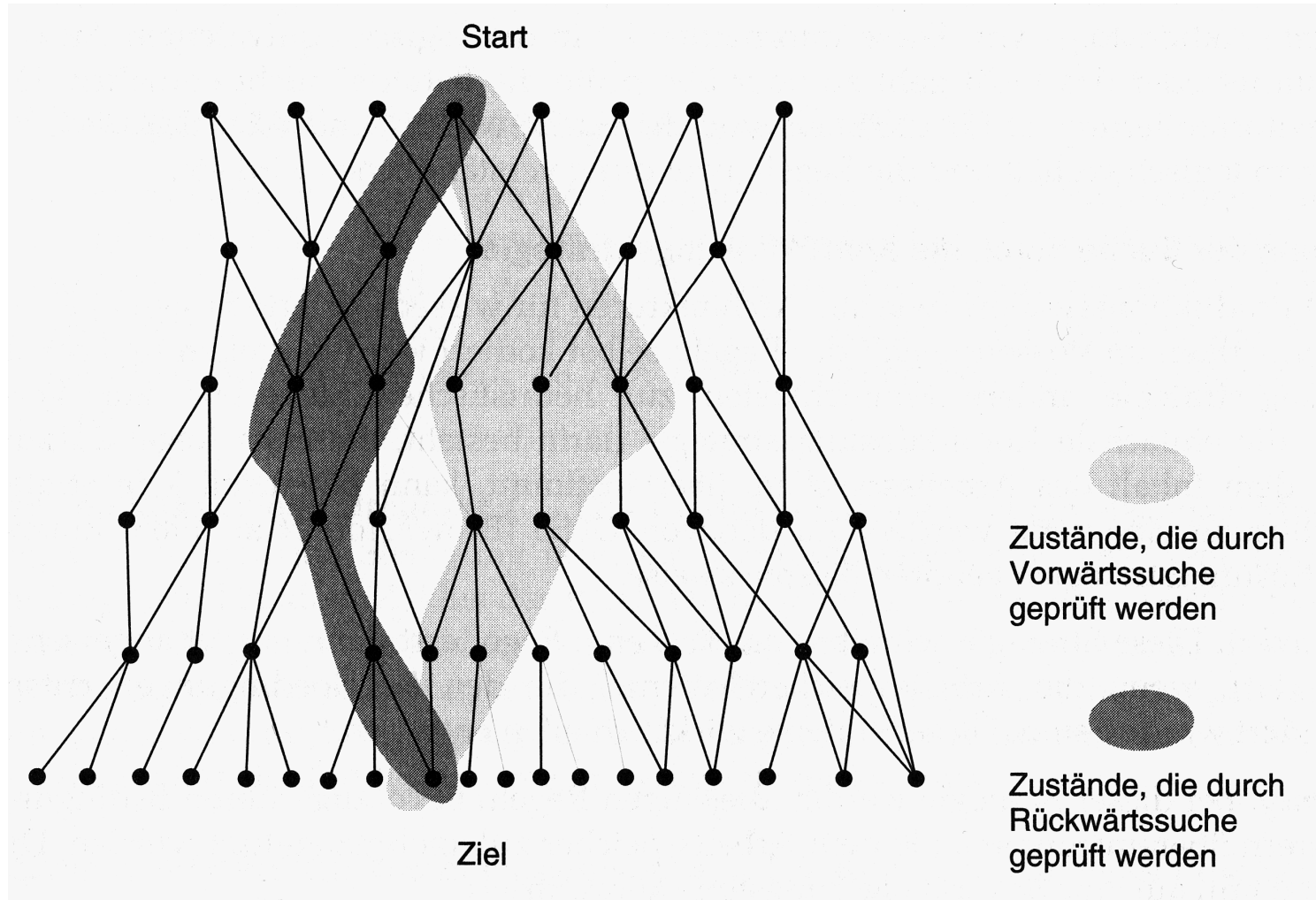
- Zwei Breitensuchen parallel



# Diskutieren Sie!

- Weshalb macht eine bidirektionale Tiefensuche wenig Sinn?
- Wie bewerten Sie die bidirektionale Breitensuche?

# Problem der bidirektionalen Tiefensuche



# Erläuterungen

- Es besteht die Gefahr, dass die beiden Suchbäume aneinander vorbei laufen.

# Bewertung der bidirektionalen Breitensuche

- Vollständig und kürzester Weg nur für Breitensuche
- Zahl der erzeugten Zustände:

$$2*v + 2*v^2 + 2*v^3 + \dots + 2*v^{t/2} = O(v^{t/2})$$

- betrifft Speicher- und Zeitbedarf
- Nicht immer anwendbar:
  - wenn Schritte rückwärts nicht möglich sind
  - wenn mehrere Zielzustände existieren
  - wenn nicht ein Lösungspfad, sondern die konkrete Ausprägung des Zielzustands gesucht wird

# Vollständigkeit und Optimalität

- Ein Suchverfahren ist optimal, wenn immer die beste Lösung gefunden wird
  - beste: kostengünstigste / der kürzeste Weg
- Ein Suchverfahren ist vollständig, wenn jede existierende Lösung grundsätzlich gefunden wird.

# Vergleich uninformierter Suchstrategien

	Breiten -suche	Bidirek- tionale Suche	Tiefen- suche	Beschränkte Tiefensuche	Iterative Ver- tiefung
<b>Vollständig</b>	ja	ja	nein	nein	ja
<b>Kürzester Weg</b>	ja	ja	nein	nein	ja
<b>Speicherbedarf</b>	$O(v^{t+1})$	$O(v^{t/2})$	$O(v^*t_{\max})$	$O(v^*t_{\text{grenz}})$	$O(v^*t)$
<b>Zeitbedarf</b>	$O(v^{t+1})$	$O(v^{t/2})$	$O(v^{t_{\max}})$	$O(v^{t_{\text{grenz}}})$	$O(v^t)$



# S1: Lernziele

- V1: Erläutern können, was Suchen kennzeichnet
- V2: Eigenschaften, Vor- und Nachteile der verschiedenen uninformatierten Suchverfahren erklären können

## S2: Informierte Suche

- Was bedeutet „informierte“ / „heuristische“ Suche?
- A, A\* - Algorithmus
  - Merkmale, Arbeitsweise
  - Eigenschaften: Zulässigkeit, Monotonie
  - Verbesserungen: IDA\*, RBFS, SMA\*
- Sonderfall: Keine Berücksichtigung der bisherigen Kosten
  - Optimistisches Bergsteigen
  - Bergsteigen mit Backtracking
  - Gierige Bestensuche

## S2: Lernziele

- V1: Eigenschaften, Vor- und Nachteile, Arbeitsweise der folgenden Suchverfahren erläutern können, sie untereinander vergleichen können und Alternativen nennen können, um mit Nachteilen umzugehen:
  - A, A\*, IDA\*, RBFS, SMA\*
  - Optimistisches Bergsteigen, Bergsteigen mit Backtracking, Gierige Bestensuche

# Diskutieren Sie!

- Worin liegt der Nachteil aller uninformatierten Suchverfahren?
- Welche Information wird nicht genutzt?

- Wissen über Qualität der Zustände nicht genutzt

# Wie Information über Zustandsqualität nutzen?

- Wie nah am Zielzustand?
- Also Information über Restkosten
- Denn bisherige Kosten sind ja bekannt
- Restkosten können nur geschätzt werden
  - Genaue Berechnung bedeutet vollständige Suche, also keine Ersparnis
  - Schätzung durch Heuristiken
- Schwierigkeit: Beste Heuristik finden

# Was ist eine Heuristik?

- Daumenregel
- Vereinfachte Betrachtung

# Verschiedene Heuristiken für das 8-Puzzle

<table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td></td><td>7</td><td>5</td></tr></table>	2	8	3	1	6	4		7	5	5	6	0
2	8	3										
1	6	4										
	7	5										
<table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table>	2	8	3	1		4	7	6	5	3	4	0
2	8	3										
1		4										
7	6	5										
<table><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>1</td><td>6</td><td>4</td></tr><tr><td>7</td><td>5</td><td></td></tr></table>	2	8	3	1	6	4	7	5		5	6	0
2	8	3										
1	6	4										
7	5											
	Anzahl Spielsteine auf falschen Positionen	Summe der Entfernungen, in denen sich Spielsteine auf falschen Positionen zueinander befinden	2 x Anzahl direkter Spielsteintausch									

1	2	3
8		4
7	6	5

Ziel

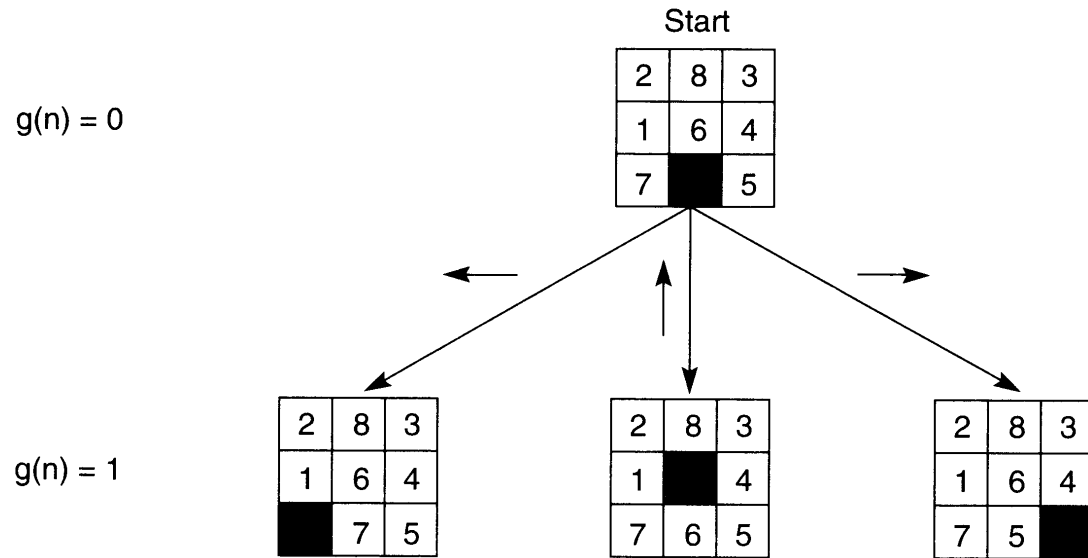


# A-Algorithmus

Gesamtkosten  $f(n)$  eines Knotens  $n$  =  
bisherige Kosten  $g(n)$   
+ geschätzte Restkosten  $h(n)$

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

# Kosten bei Heuristik „Anzahl falsche Positionen“



Werte von  $f(n)$  für jeden Zustand, **6**

**4**

**6**

wobei:

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

$g(n)$  = tatsächliche Entfernung von  $n$   
zum Startzustand, und

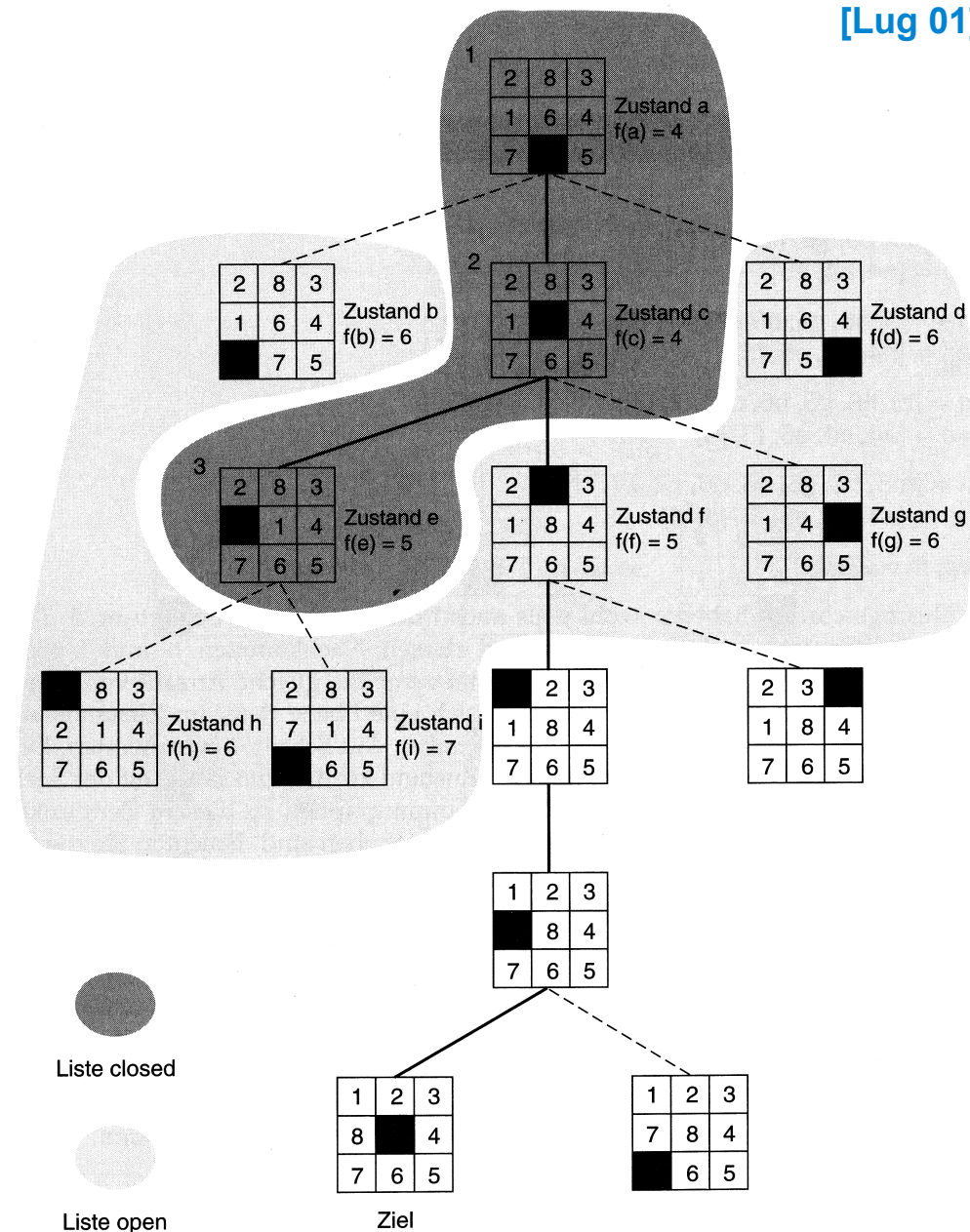
$h(n)$  = Anzahl Spielsteine auf falschen  
Positionen.

1	2	3
8		4
7	6	5

Ziel

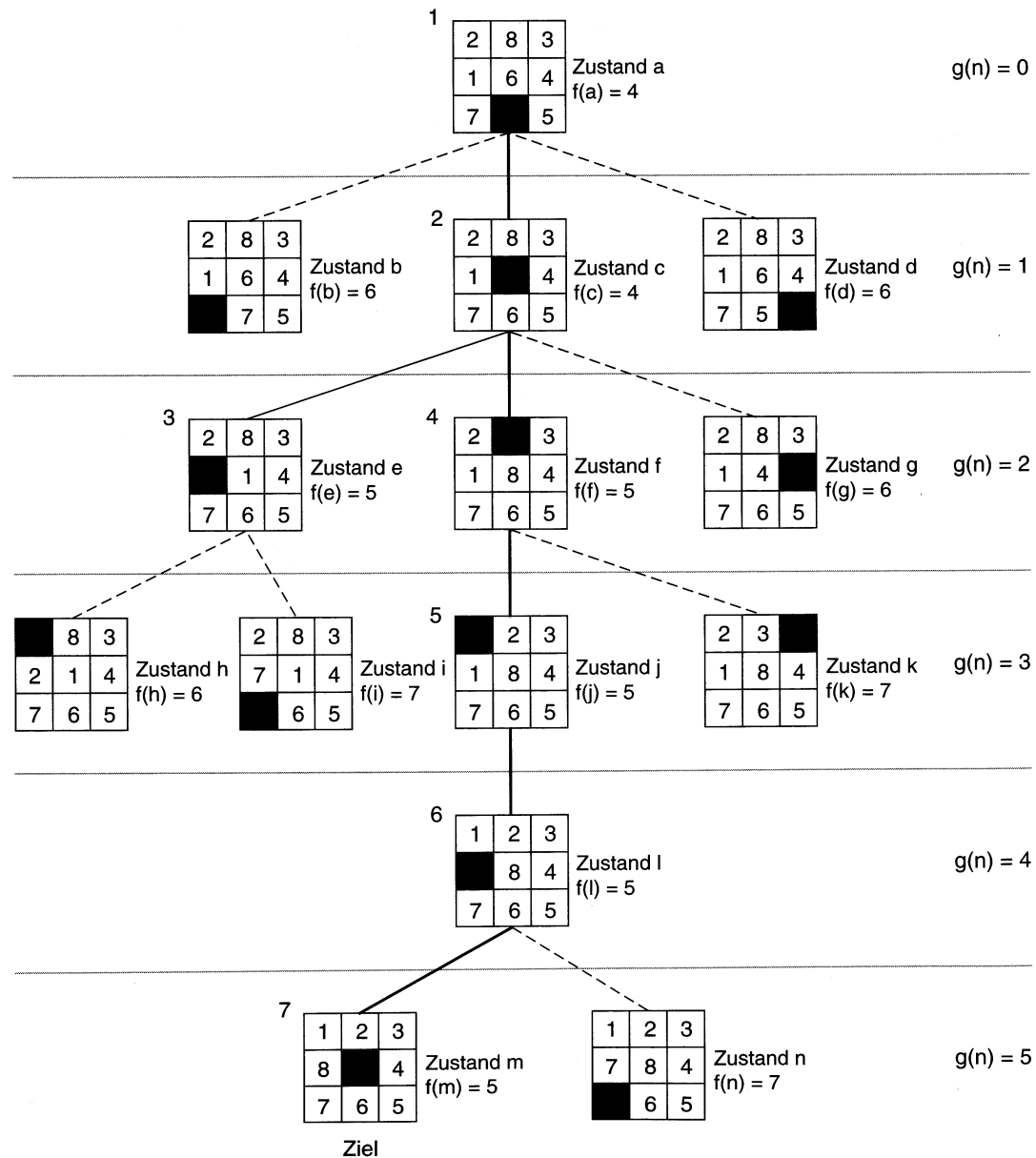
# Nach 3 Schritten

- Geschlossene Knoten:  
Alle Kindknoten sind bekannt
- Offene Knoten:  
Noch nicht expandiert,  
also Kindknoten nicht  
bekannt
- Grau unterlegte Mengen  
betreffen die Situation  
nach 3 Schritten



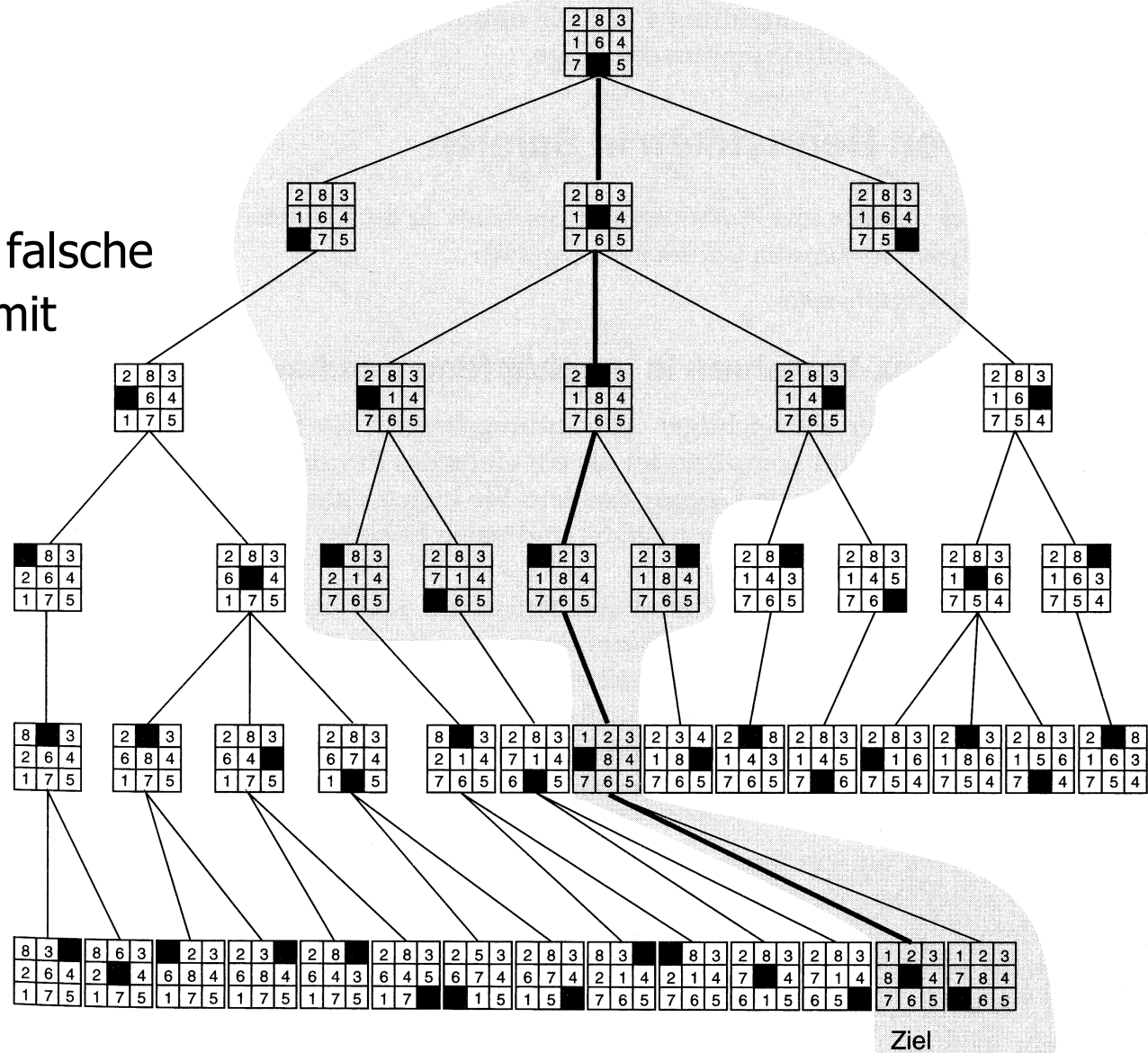
# Vollständiger Baum

- Gestrichelte Linien: Pfade, die nicht weiter untersucht werden, weil die Knoten zu ungünstig bewertet sind



# Vergleich

- Heuristik „Anzahl falsche Positionen“ wird mit Breitensuche verglichen



# Diskutieren Sie!

- Wie wirkt es sich (im ungünstigen Fall) aus, was bedeutet es, wenn die Restkosten
  - potenziell überschätzt werden?
  - nie überschätzt werden?
  - immer auf 0 geschätzt werden?
  - ganz exakt berechnet werden?



# Lösung

- Überschätzung:

Der günstigste Weg zum Ziel wird ignoriert, wenn die Restkosten deutlich zu hoch eingeschätzt werden. So kann es passieren, dass ein anderes, ungünstigeres Ziel gefunden wird. Da auch immer die bisherigen Kosten zu Buche schlagen, kann das Verfahren nicht beliebig lange auf einem ungünstigen Weg bleiben.

- Unterschätzung:

Hohe Unterschätzung bedeutet wenig Information. Es wird wenig zielgerichtet vorgegangen, es werden viele Pfade untersucht. Aber: Die beste Lösung kann nicht verfehlt werden.

- Restkosten = 0:

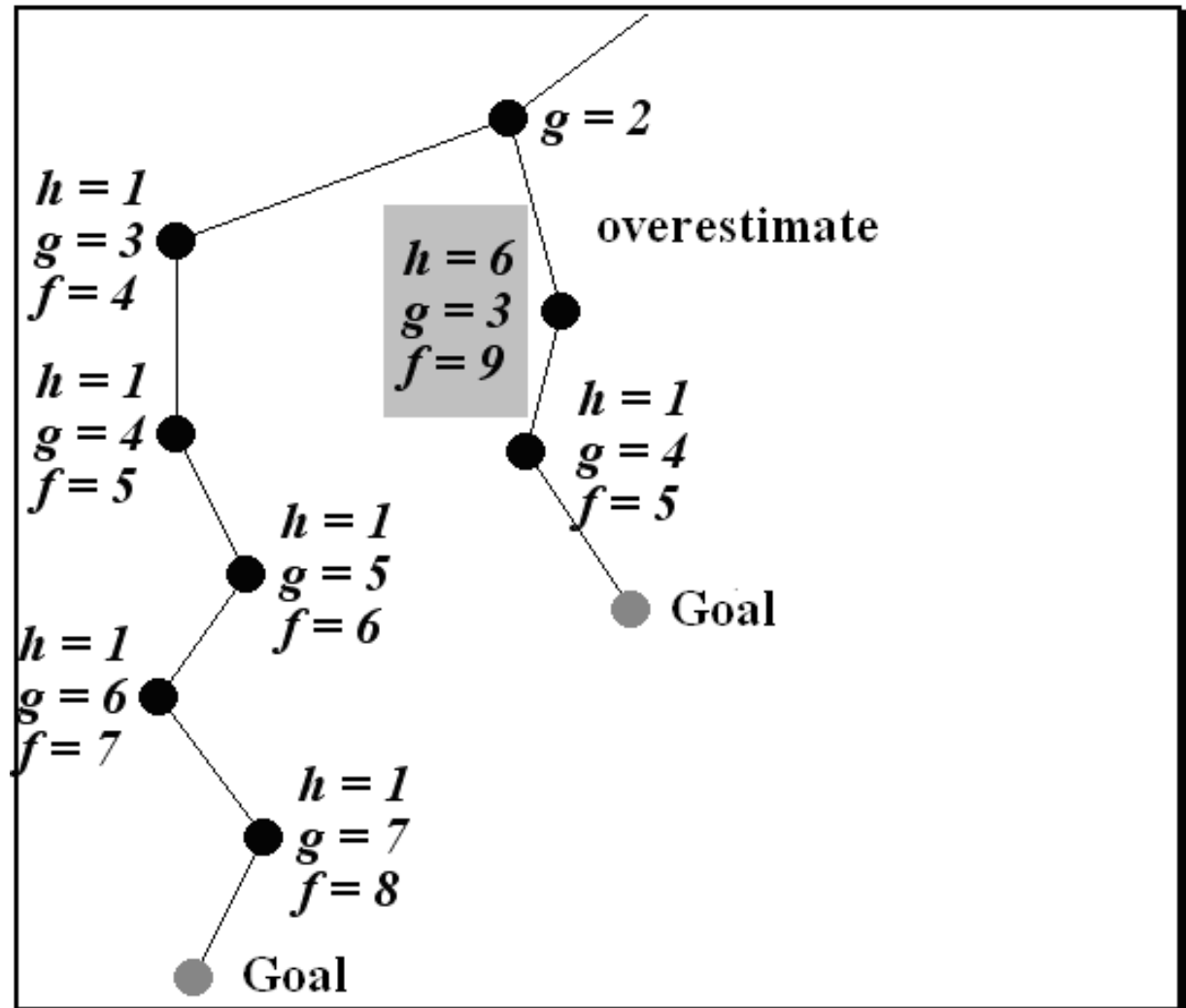
Dies ist der Extremfall der uninformierten Breitensuche

- Restkosten exakt berechnet:

Das setzt eine aufwändige Berechnung voraus, also keine Ersparnis durch Heuristik.

# Überschätzung der Restkosten

- Günstigstes Ziel (rechts) wird verfehlt





# A\*-Algorithmus: Zulässigkeit

- Schätzfunktion / Heuristik ***h*** heißt **zulässig**, wenn sie die Restkosten nicht überschätzt:

$$g(\text{Zielknoten}) - g(n) \geq h(n), \text{ für alle } n, \text{ bzw.}$$

$$h^*(n) \geq h(n), \text{ für alle } n,$$

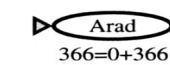
(*h*\*: Funktion der tatsächlichen Kosten)

- **A\*-Algorithmus:** *h* ist zulässig
- A\* ist optimal
- Für zwei zulässige Funktionen ***h*<sub>1</sub>**, ***h*<sub>2</sub>** gilt:

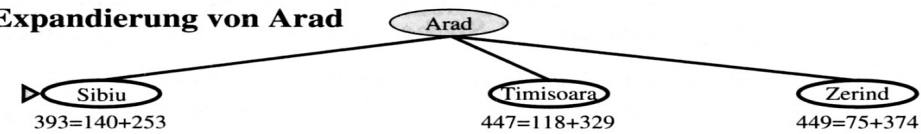
Wenn ***h*<sub>1</sub>(*n*) ≥ *h*<sub>2</sub>(*n*)** für alle ***n***, dann ist ***h*<sub>1</sub>** informierter

# A\*-Suche: Beispiel

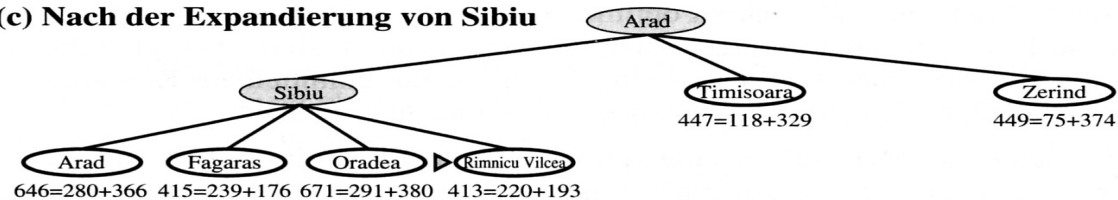
(a) Der Ausgangszustand



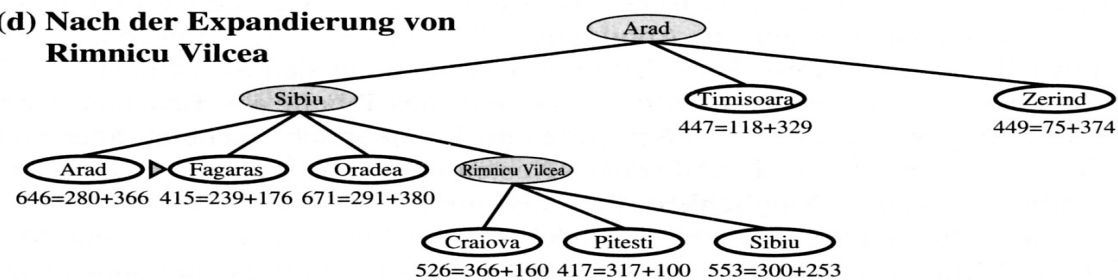
(b) Nach der Expandierung von Arad



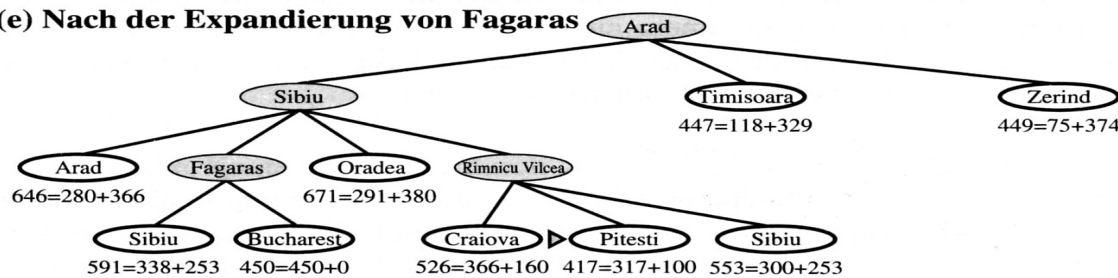
(c) Nach der Expandierung von Sibiu



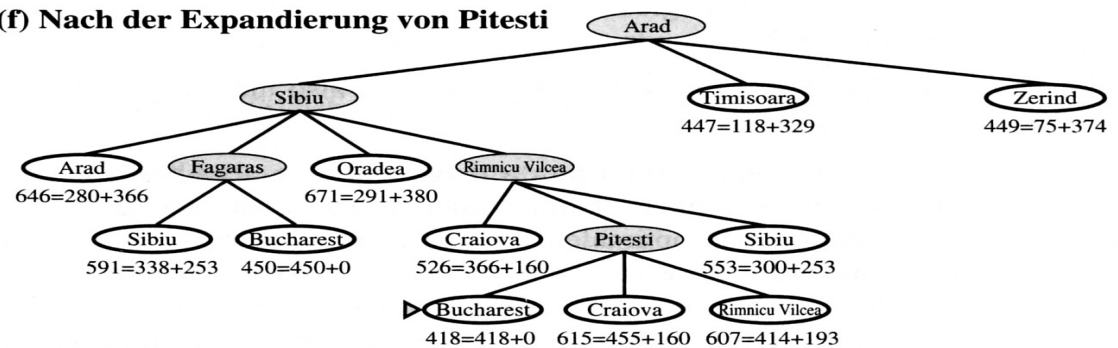
(d) Nach der Expandierung von Rimnicu Vilcea



(e) Nach der Expandierung von Fagaras



(f) Nach der Expandierung von Pitesti



# Monotonie (Konsistenz) der Schätzfunktion

- Heuristik ist monoton (konsistent), wenn die Kosten zwischen **zwei beliebigen Knoten** nie überschätzt werden
  - Das heißt: Die Werte von  $f$  entlang eines Pfades dürfen nicht fallen
- Im Detail:
  - Annahme: Knoten liegt irgendwo auf optimalem Pfad
  - Überschätzung kann bedeuten, dass zunächst nicht-optimaler Pfad zu diesem Knoten gefunden wird
  - Wegen Zulässigkeit wird dann aber später auch der optimale Pfad gefunden
  - Ohne Überprüfung der Wegekosten würde der optimale Pfad keine Berücksichtigung finden
- Zulässige Heuristiken sind meistens auch monoton (Gegenbeispiele schwer zu finden)

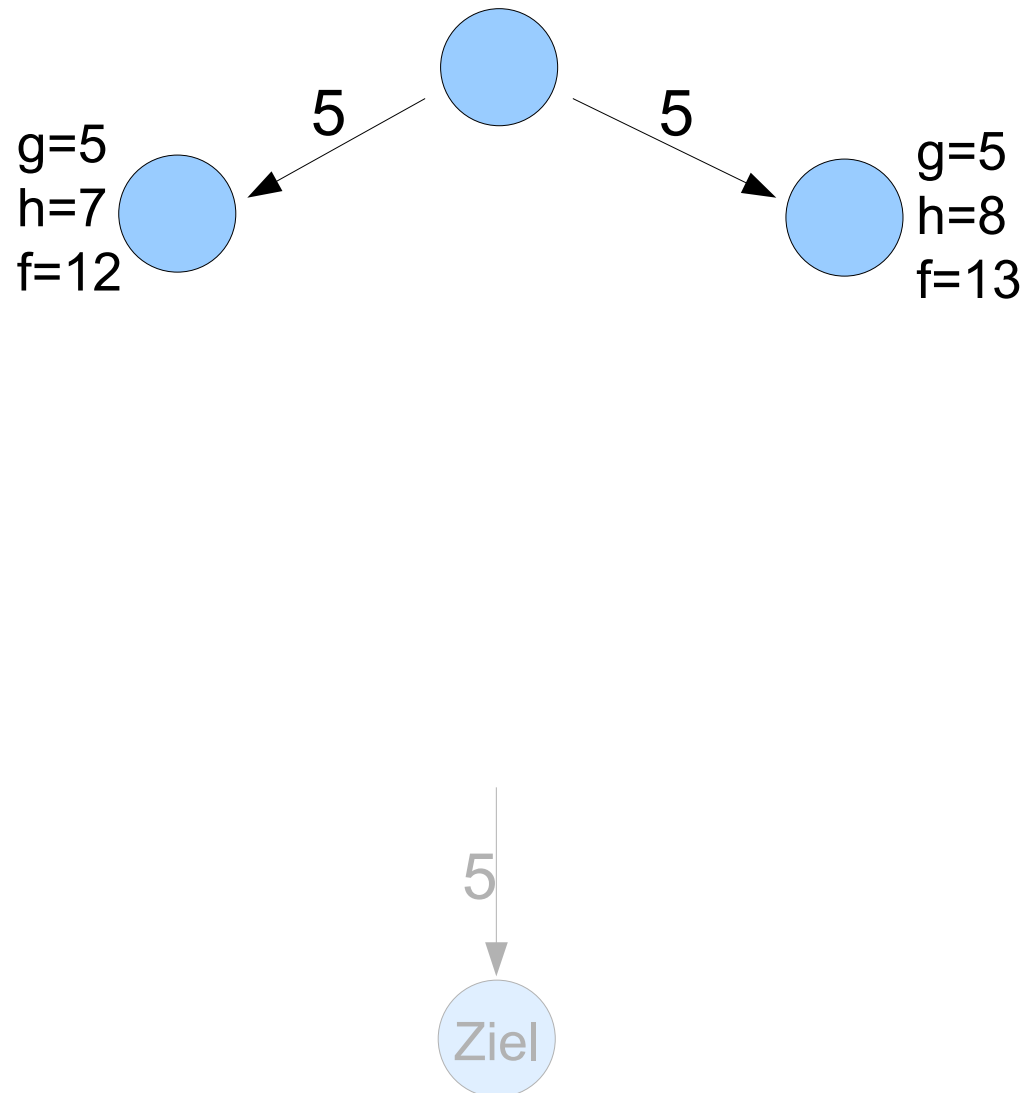
# Erläuterungen

Monotonie bedeutet, dass nicht nur die Kosten bis zum Zielzustand (das wäre Zulässigkeit) nicht überschätzt werden dürfen, sondern außerdem auch die Kosten zu beliegen anderen Zuständen.

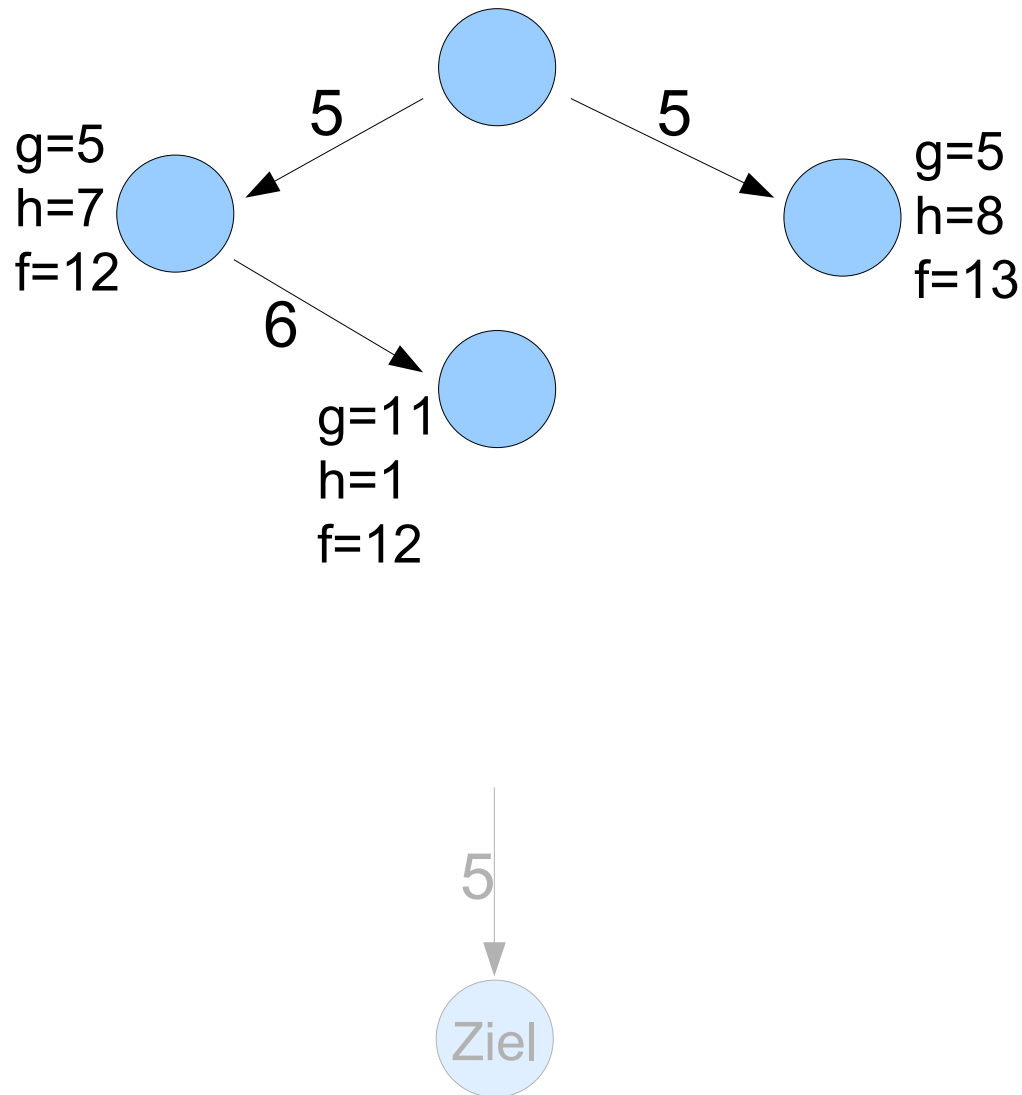
Wenn man sich zu einem Nachfolge-Zustand bewegt, erfährt man ja die tatsächlichen Kosten. In der Gesamtkostenbetrachtung werden dann also die bisher geschätzten Kosten zu diesem Nachfolge-Zustand durch die tatsächlichen ersetzt. Wenn für jede Teilstrecke keine Überschätzung vorliegt, also typischerweise eine leichte Unterschätzung, dann führt das Ersetzen der geschätzten Kosten für eine Teilstrecke durch die tatsächlichen Kosten zu einer Erhöhung der Gesamtkosten. Auf dem Weg vom Startzustand zum Zielzustand erhöhen sich also die Gesamtkosten von Schritt zu Schritt immer weiter. Daher wird eine Heuristik, die auch Teilstrecken nicht überschätzt, als monoton bezeichnet.

Wenn eine Heuristik zwar zulässig, aber nicht monoton ist, können also Teilstrecken überschätzt werden. Dies kann dazu führen, dass zu einem innen liegenden Zustand nicht der optimale (also der kostengünstigste) Pfad gefunden wird. Aufgrund der Zulässigkeit der Heuristik, würde zu diesem Zustand, wenn er denn auf dem optimalen Pfad zum Zielzustand liegt, später auch der optimale Pfad für diese Teilstrecke gefunden werden. Zu diesem innen liegenden Zustand würde also ein zweiter, kostengünstigerer Weg gefunden werden. Dies erfordert ein Update der Kosten zu dem Zustand und nachfolgender Berechnungen. Wenn eine Heuristik nicht monoton, wohl aber zulässig ist, muss man sich also um eine Aktualisierung der Kosten zu den Zuständen aus der Closed-List und ihrer Nachfolger kümmern. Dies ist bei einer monotonen Heuristik nicht nötig.

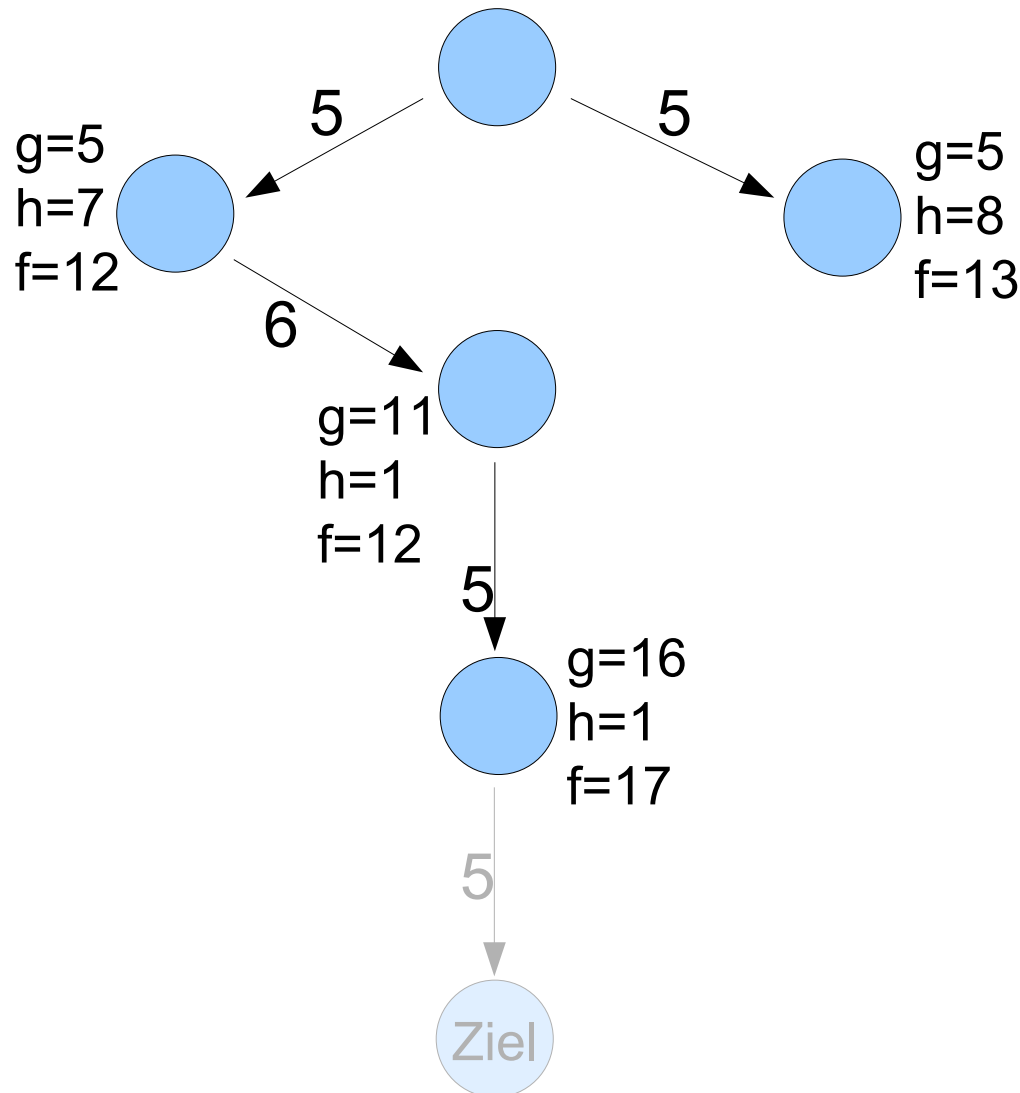
# Beispiel: Zulässige, nicht-monotone Schätzfunktion



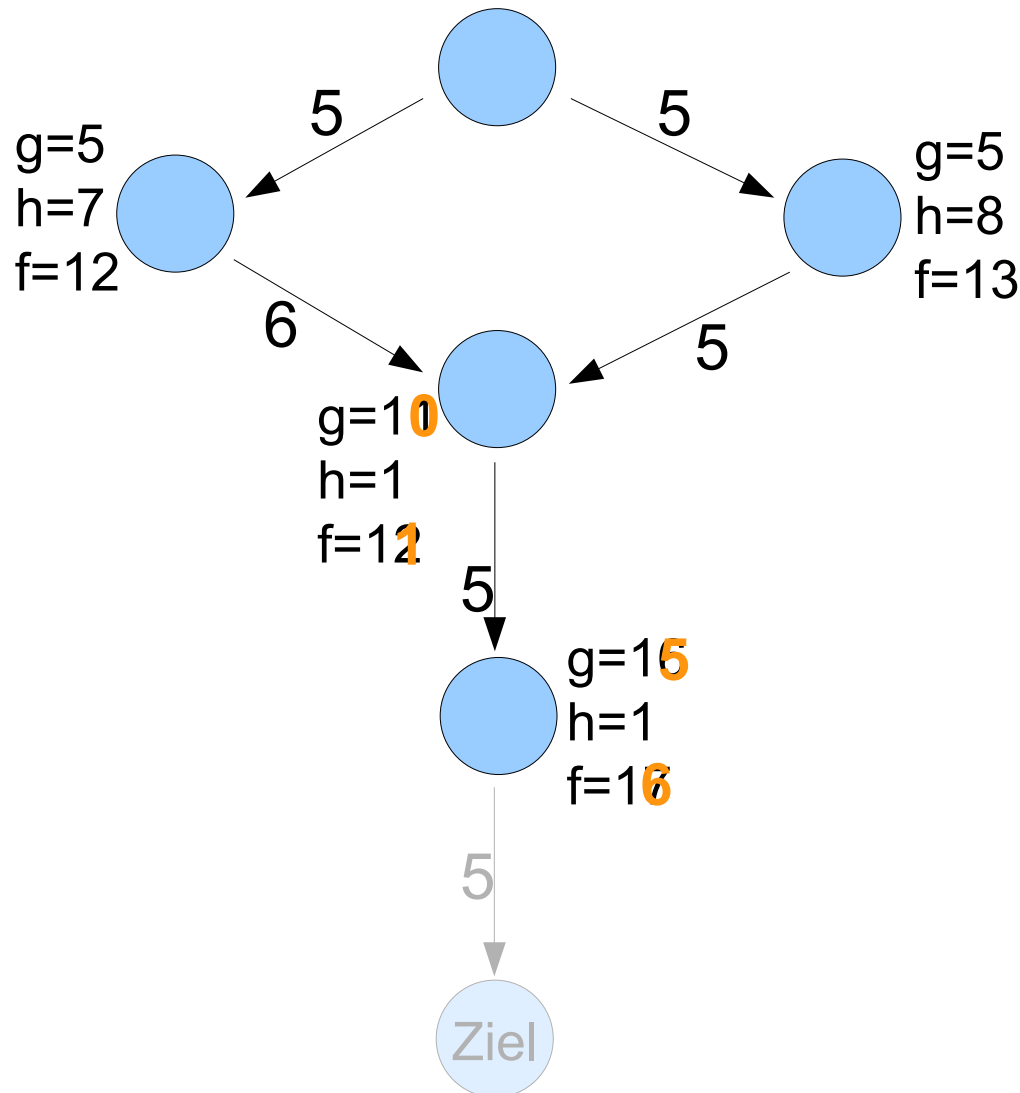
# Beispiel: Zulässige, nicht-monotone Schätzfunktion



# Beispiel: Zulässige, nicht-monotone Schätzfunktion



# Beispiel: Zulässige, nicht-monotone Schätzfunktion



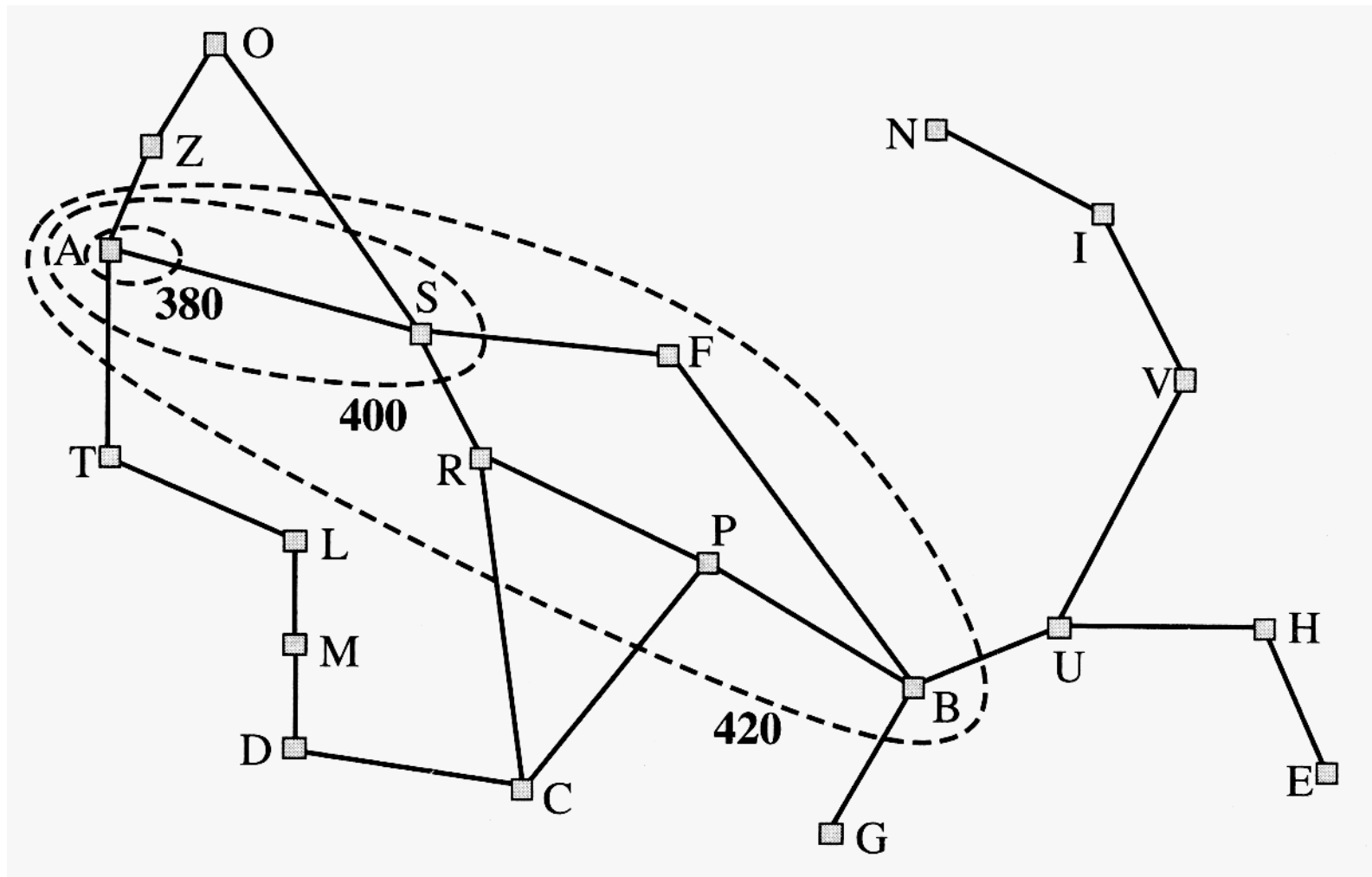


# A\*: Bewertung von Heuristiken

- Bestmögliche Heuristik ist unsinnig
  - Tatsächliche Kosten müssen berechnet werden (die Aufgabe, die eigentlich vereinfacht werden sollte)
- $h(n) = 0$  entspricht Breitensuche
- Kompromiss erforderlich
- Gute Heuristiken haben effektiven Verzweigungsfaktor nahe 1
  - Sei N Anzahl der erzeugten Knoten bis Lösung in Tiefe t gefunden wurde:

$$N+1 = 1 + v_{\text{eff}} + v_{\text{eff}}^2 + v_{\text{eff}}^3 + \dots + v_{\text{eff}}^t$$

# Gute A\*-Heuristik: Geringer Verzweigungsfaktor



# Vergleich verschiedener Heuristiken

- $h_1$ : Anzahl Spielsteine auf falschen Positionen
- $h_2$ : Summe der Entfernungen

$d$	Suchkosten			Effektiver Verzweigungsfaktor		
	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.

# Automatische Erzeugung zulässiger Heuristiken!

- Möglich, wenn Suchproblem in formaler Sprache ausgedrückt wird
- Methode (u. a.): Erzeugung "gelockerter" Probleme, dann Formulierung entsprechender Heuristik
  - Heuristik entspricht der **genauen** Pfadlänge eines gelockerten Problems
  - Beispiel 8-Puzzle: Anzahl Spielsteine auf falschen Positionen ist Pfadlänge, wenn beliebige Sprünge des leeren Feldes erlaubt sind
  - Wenn der Heuristik ein gelockertes Problem zugeordnet werden kann, ist sie zulässig
- Programm ABSOLVER hat beste bekannte Heuristik für 8-Puzzle und erste sinnvolle Heuristik für Zauberwürfel gefunden

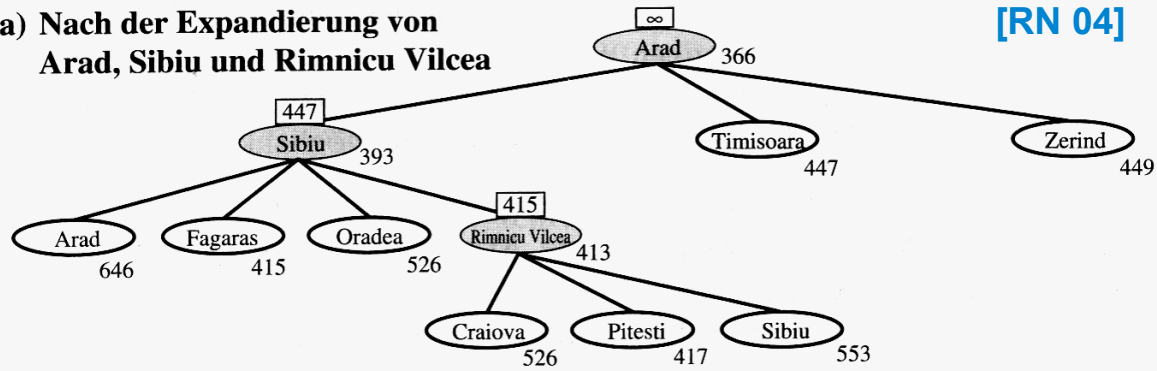
# A\*: Bewertung des Aufwands

- 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
  - RBFS (Recursive Best First Search)
  - IDA\* (Iterative Deepening A\*)
    - Tiefensuche !

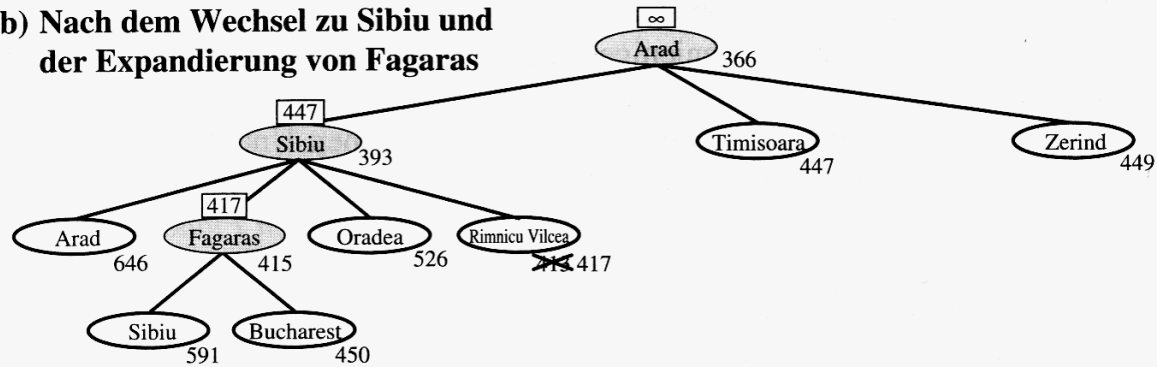
# RBFS

[RN 04]

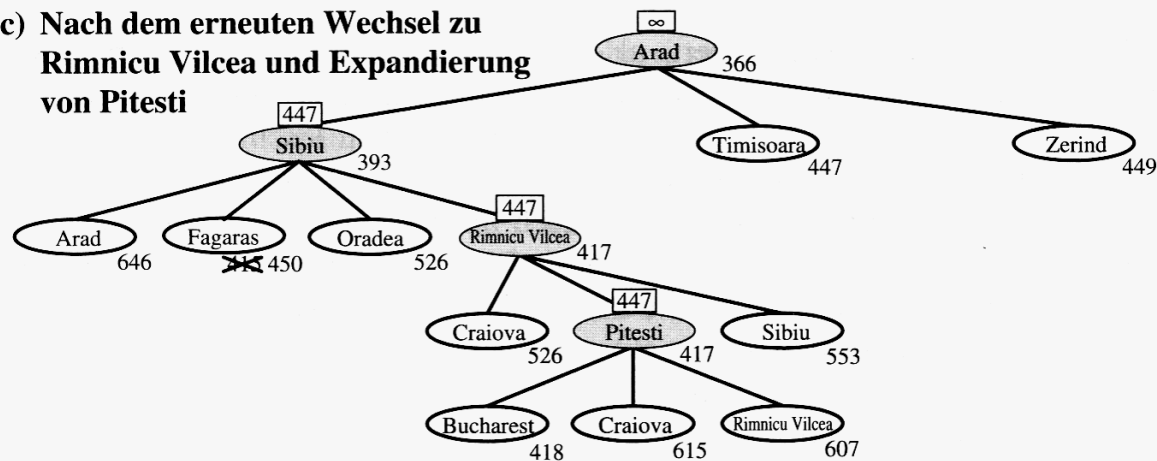
(a) Nach der Expansionierung von Arad, Sibiu und Rimnicu Vilcea



(b) Nach dem Wechsel zu Sibiu und der Expansionierung von Fagaras



(c) Nach dem erneuten Wechsel zu Rimnicu Vilcea und Expansionierung von Pitesti



# Erläuterung IDA\*

- Zunächst wird nur der Startknoten expandiert
- Der günstigste Wert  $f(n)$  aller Kindknoten wird global gespeichert
- Neustart mit gespeichertem Wert als Schranke:
  - Alle Knoten mit  $f(n) \leq \text{Schranke}$  werden expandiert, und zwar in der Reihenfolge einer Tiefensuche
  - Der günstigste Wert  $f(n)$  eines Blattknotens wird als neue Schranke gespeichert
  - Erneuter Neustart mit neuer Schranke
- Wert der Schranke wächst mit jeder Iteration, da Kosten ja unterschätzt werden
- Achtung: IDA\* ist eine Tiefensuche.
  - Pfade und Knoten, die im Rahmen des Backtracking nicht weiter verfolgt werden, werden gelöscht

# Bewertung RBFS, IDA\*

- Jetzt ist Zeitaufwand limitierend
  - Unnötig wenig Speicherbedarf
- Verbesserung:
  - Verfügbaren Speicher nutzen, erst dann mehr rechnen
  - MA\* (Memory-bounded A\*)
  - SMA\* (Simplified MA\*)



- Arbeitsweise:

- Wie A\* bis Speicher voll ist
- Dann Löschen des schlechtesten Unterbaums mit Übertragung des besten Wertes auf den Vorgänger (wie RBFS)

- Nachteil:

- Bei bestimmten Problemen abwechselndes Untersuchen bzw. Wegschalten konkurrierender Pfade
  - Potenziell hoher Rechenaufwand sobald Speichergrenze erreicht

- Einziger Ausweg:

- Verzicht auf Optimalität oder Vollständigkeit

- Zum Nachteil von SMA\*:

Das Löschen von Teilbäumen und Wechseln zu einem anderen Ast ist ja nur dann hilfreich, wenn der neue Ast wenigstens für einige Schritte weiter expandiert wird. Wenn jedoch bereits nach ein, zwei Schritten klar wird, dass der erste Ast doch der bessere war und nach dem Löschen des zweiten Asts und ein, zwei Schritten festgestellt wird, dass nun der zweite Ast wieder günstiger erscheint, so ergibt sich ein ständiges Löschen und Wiederherstellen von Ästen ohne dass nennenswerter Fortschritt in der Suche erreicht wird.

# Lernen, besser zu suchen?

- Erlernen der besten Suchstrategie ist möglich
- Idee: Auswertung der schrittweisen Expansion des Zustandsbaums aus übergeordneter Sicht
  - Zustandsraum auf Metaebene
  - Was sind die Merkmale der Teilbäume, die vergeblich untersucht wurden?

# Diskutieren Sie!

- Unter welchen Bedingungen könnte es nützlich sein, die bisherigen Kosten nicht zu berücksichtigen?

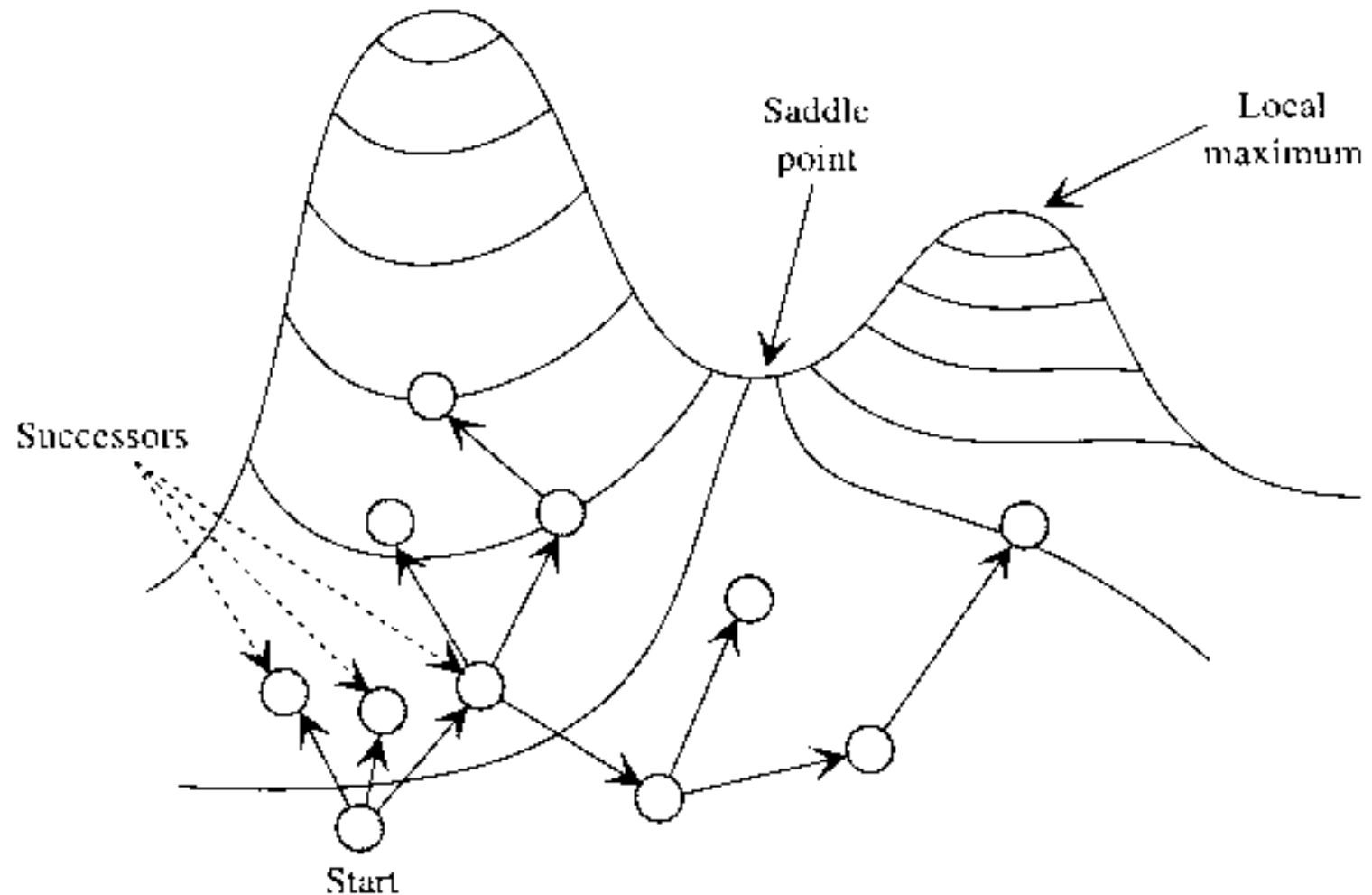
# Sonderfall: $f(n) = h(n)$

- Nützlich
  - für bestimmte Suchprobleme zur Bestimmung von Lösungspfaden???
  - wenn nicht der Pfad, sondern ein Zielzustand gesucht wird
  - für Optimierungsprobleme ( $h(n)$  liefert Wert des Zustands)
- Optimistisches Bergsteigen
- Bergsteigen mit Backtracking
- Gierige Bestensuche

# Optimistisches Bergsteigen (Hill Climbing)

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

# Problem der lokalen Maxima



# Umgang mit lokalen Maxima: Sprünge (1)

- Stochastisches Bergsteigen
  - Für bestimmten Prozentsatz der Schritte:
    - Zufällige Auswahl des Nachfolge-Zustands
    - Zufälliger Sprung
    - Zufälliger Neustart
- Simulated Annealing
  - Zufällige Sprünge mit abnehmender Sprungweite

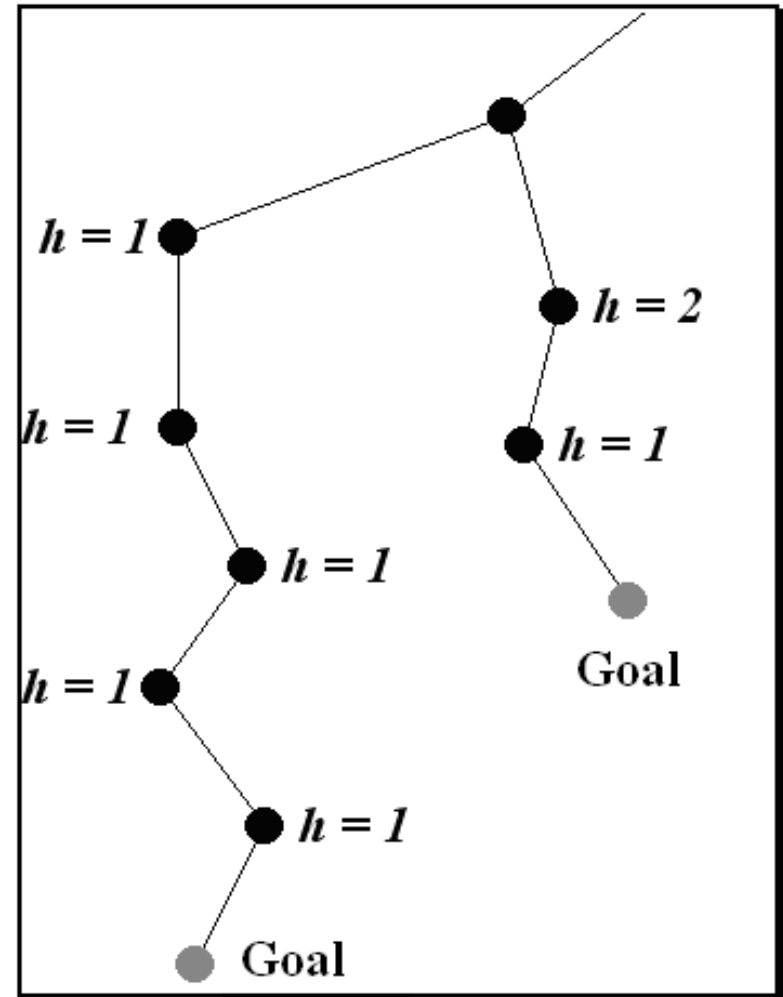


# Umgang mit lokalen Maxima: Sprünge (2)

- Stochastische Strahlsuche
  - Mehrere (k) Startpunkte gleichzeitig
  - Nächste Runde jeweils mit den k besten Nachfolgern
  - Stochastische Strahlsuche: Je schlechter der Wert des Zustands, desto höhere Wahrscheinlichkeit für Sprung
- Variante der stochastischen Strahlsuche: Genetische Algorithmen
  - Kombination von zwei günstigen Zustände zu einem neuen Nachfolger plus zufälliger Mutation
  - Anfangs große Schritte im Zustandsraum (weil Eltern sehr unterschiedlich), später kleinere Schritte (Gemeinsamkeit mit Simulated Annealing)

# Probleme durch Wegfall der bisherigen Kosten

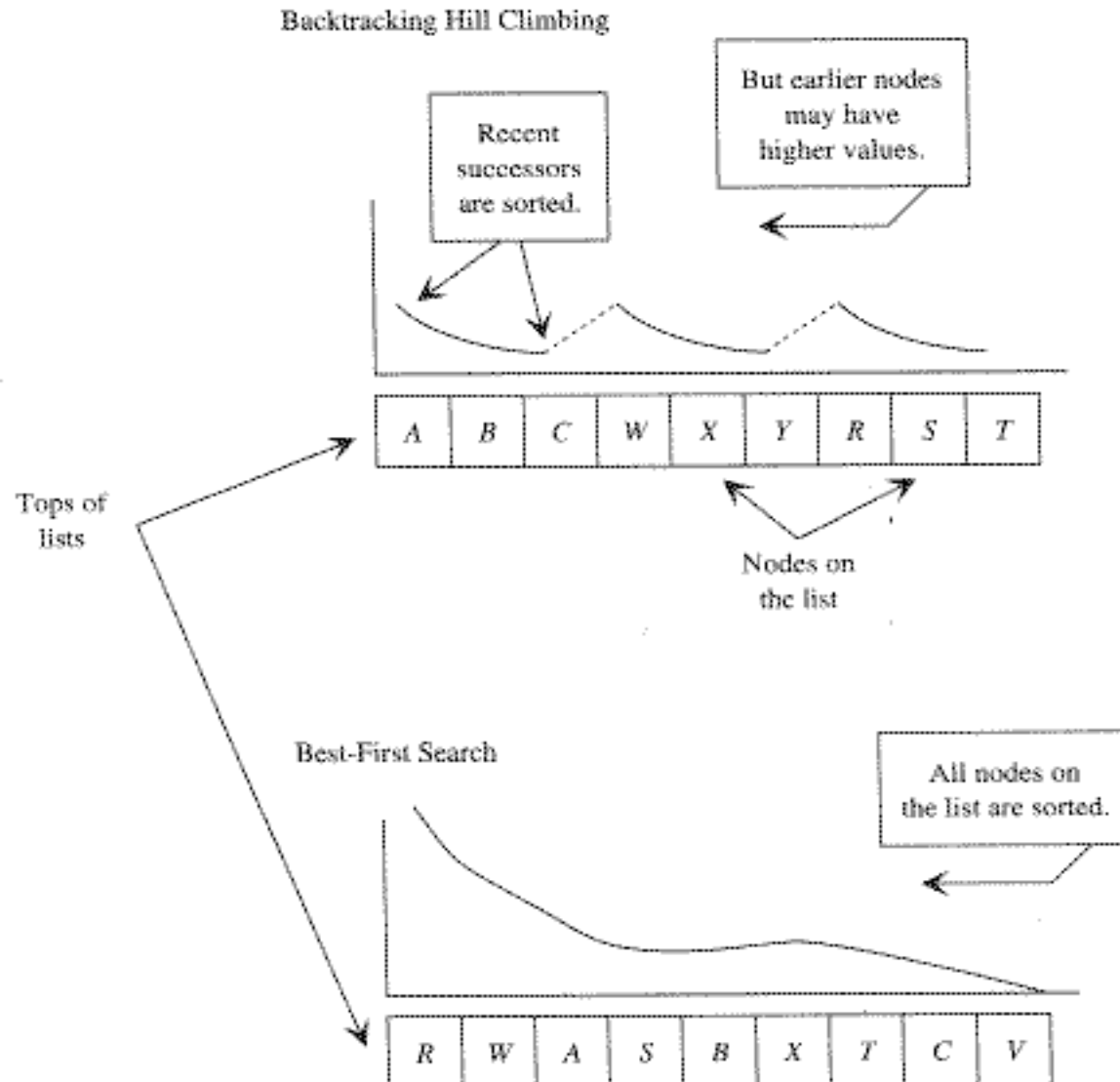
- Optimale Lösung kann verfehlt werden
- Lösung insgesamt kann verfehlt werden (Verlust der Vollständigkeit)
  - unendlich lange Pfade mit guter Bewertung  $h$



# Bergsteigen mit Backtracking

- Informierte Tiefensuche ohne Berücksichtigung der bisherigen Kosten (***g***)
- Geschwister-Knoten jeweils untereinander sortiert
- LIFO-Prinzip für die Gruppen von Geschwister-Knoten
- Backtracking
- Nicht vollständig

# Bergsteigen m. Backtr. vs. Gierige Bestensuche



# Erläuterung

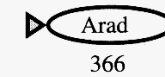
- Bei der gierigen Bestensuche werden also alle Knoten der Open List sortiert, genau wie bei  $A^*$ . Der Unterschied zu  $A^*$  besteht lediglich darin, dass die bisherigen Kosten  $g(n)$  ignoriert werden.
- Bergsteigen mit Backtracking ist ein Tiefensuchverfahren. Die bei der Expansion eines Knotens erzeugten Kindknoten werden also immer vorn in die Open List eingefügt, allerdings im Gegensatz zur uninformierten Suche jeweils untereinander sortiert. So besteht die Open List also aus sortierten Teil-Listen von Geschwisterknoten, zuerst kommen die im Baum am weitesten unten liegenden Geschwisterknoten, dann die auf der Ebene darüber, usw.

# Gierige Bestensuche

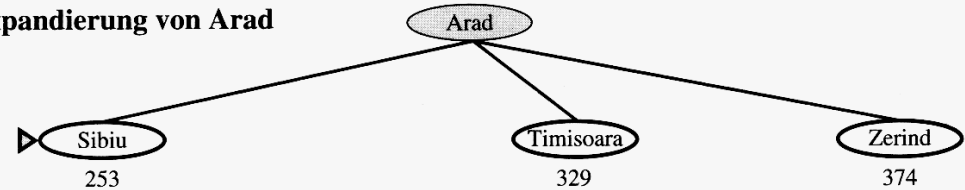
- Keine Berücksichtigung der bisherigen Kosten (***g***)
- Alle Knoten sortiert
  - Also wie  $A^*$  ohne bisherige Kosten
- Nicht vollständig
  - Eine anhaltend zu optimistische Restschätzung eines schlechten Pfades führt dazu, dass dieser Pfad beliebig lange verfolgt wird.

# Gierige Bestensuche: Beispiel

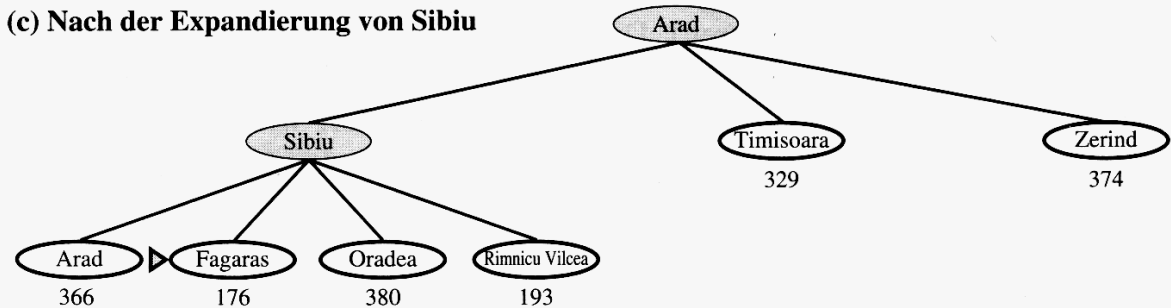
(a) Der Ausgangszustand



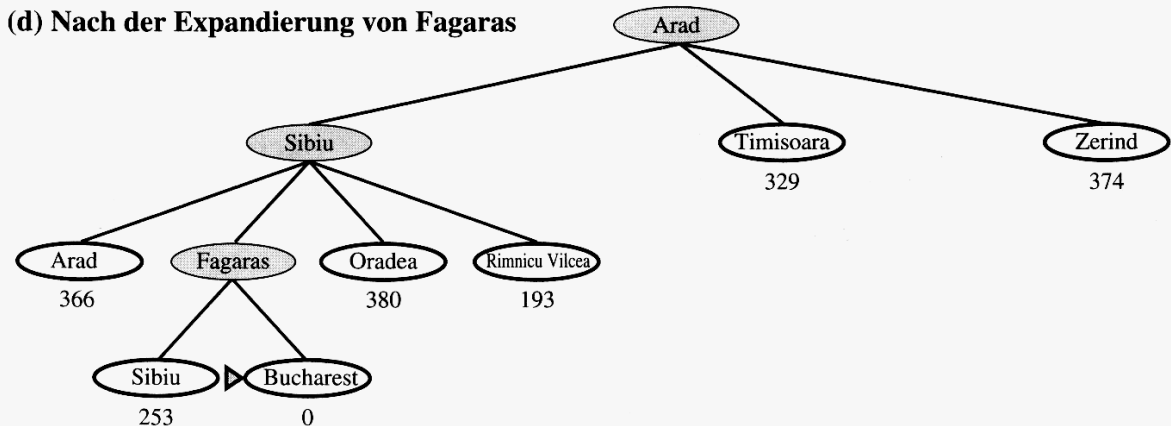
(b) Nach der Expandierung von Arad



(c) Nach der Expandierung von Sibiu



(d) Nach der Expandierung von Fagaras



# Wie kann der Stoff nachbereitet werden?

- Russell, Norvig: "Künstliche Intelligenz – Ein moderner Ansatz", Kapitel 4
  - Sehr ausführlich
  - Gut zu lesen



## S2: Lernziele

- V1: Eigenschaften, Vor- und Nachteile, Arbeitsweise der folgenden Suchverfahren erläutern können, sie untereinander vergleichen können und Alternativen nennen können, um mit Nachteilen umzugehen::
  - A, A\*, IDA\*, RBFS, SMA\*
  - Optimistisches Bergsteigen, Bergsteigen mit Backtracking, Gierige Bestensuche

# S3: Allgemeiner Suchalgorithmus

## S3: Lernziele

- V1: Aufbau eines allgemeinen Suchalgorithmus beschreiben können
- V2: Erklären können, wie Steuerung der Suchstrategie über Sortierung der Open List für die verschiedenen Suchverfahren erfolgt (Gegenstand von Praktikum 3)

# Diskutieren Sie!

- Welche Lösungsbestandteile muss ein allgemeines Suchverfahren aufweisen?

# Allgemeiner Suchalgorithmus: Pseudocode

```
procedure search;  
  paths:= [[start1],[start2],...,[startn]];  
  success:= false;  
  repeat  
    p:= first(paths);  
    success:= goal(first(p));  
    if not success then begin  
      children:= expand(first(p));  
      newpaths:= generate_new_paths(children,p);  
      paths:= insert(newpaths,rest(paths));  
    end;  
  until success;  
  return reverse(p);  
end.
```

# Übung: Expertenpuzzle

## S3: Lernziele

- V1: Aufbau eines allgemeinen Suchalgorithmus beschreiben können
- V2: Erklären können, wie eine Steuerung der Suchstrategie über Sortierung der Open List für die verschiedenen Suchverfahren erfolgt (Gegenstand von Praktikum 3)