INF FAKULTÄT FÜR INFORMATIK

Intelligente Systeme

Heuristische Suchalgorithmen

Prof. Dr. R. Kruse C. Moewes G. Ruß

{kruse,cmoewes,russ}@iws.cs.uni-magdeburg.de

Institut für Wissens- und Sprachverarbeitung

Fakultät für Informatik

Otto-von-Guericke Universität Magdeburg

Übersicht

Wiederholung: Baumsuche

Algorithmus 1 TREE-SEARCH

Eingabe: Problembeschreibung *problem*, Rand *fringe*
Ausgabe: Lösung oder Fehler

```
 1: seq ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
 2: while true {
 3:     if fringe is empty {
 4:         return failure
 5:     }
 6:     node ← REMOVE-FRONT(fringe)
 7:     if GOAL-TEST(problem, STATE(node)) {
 8:         return node
 9:     }
10:     fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
11: }
```

⇒ diverse Suchstrategien möglich durch verschiedene Reihenfolgen der Knotenexpansion

Bestensuche

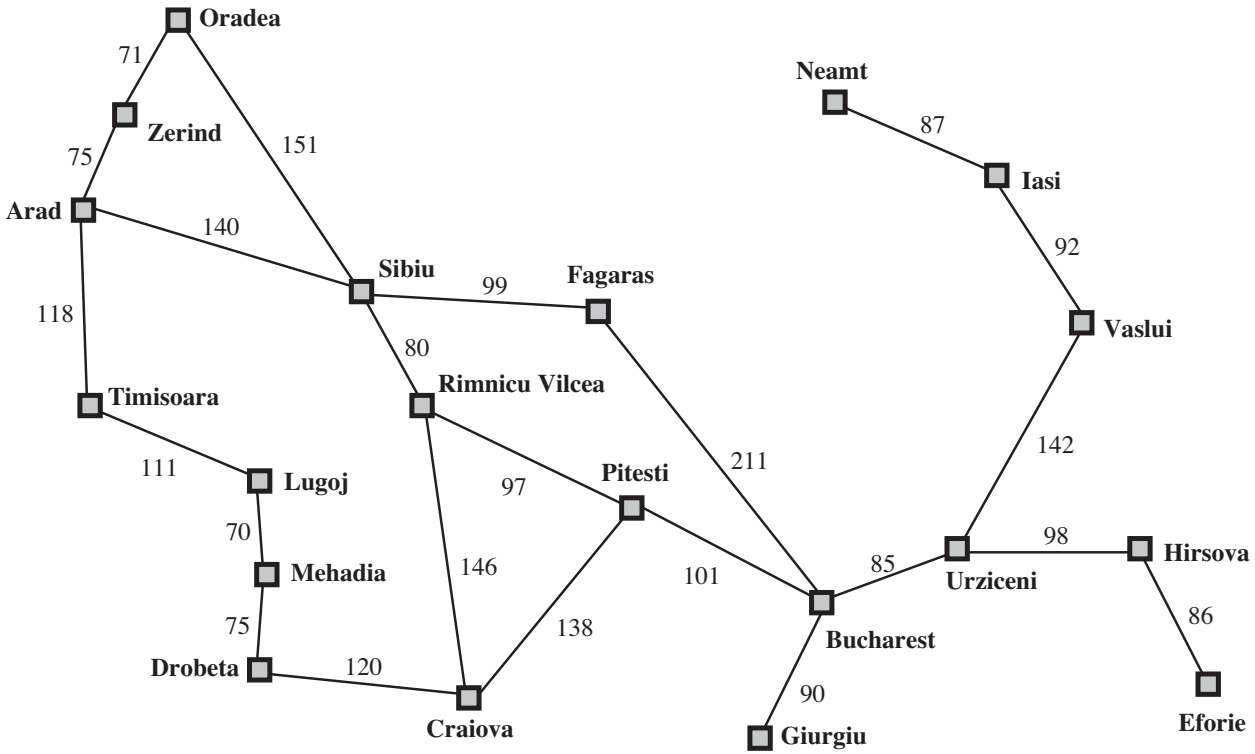
Idee: nutze Bewertungsfunktion für jeden Knoten

- Schätzung, wie „wünschenswert/begehrt“ Knoten ist

$\Rightarrow$ Expansion des am wünschenswertesten (noch nicht expandierten) Knotens

- Implementierung:
  fringe = Queue absteigend sortiert nach „Begehrtheit“
- Spezialfälle: Greedy-Suche, A*-Algorithmus

Beispiel: Rumänien mit Schrittkosten in km



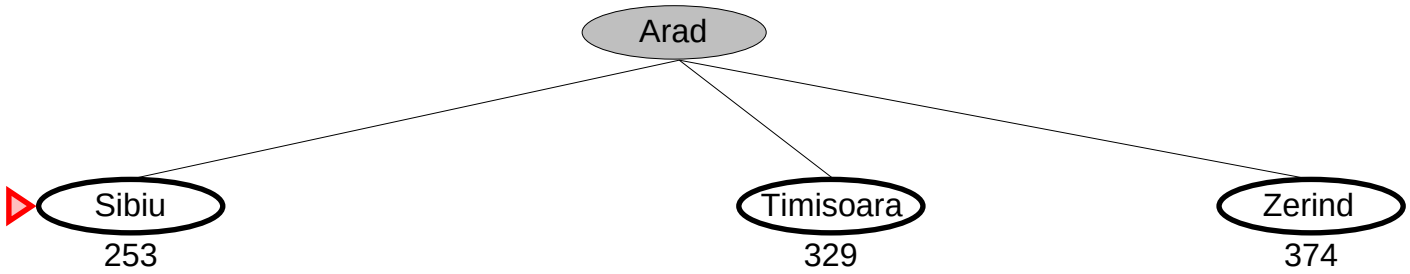
Luftlinie nach Bukarest	
Arad	366
Bukarest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy-Suche

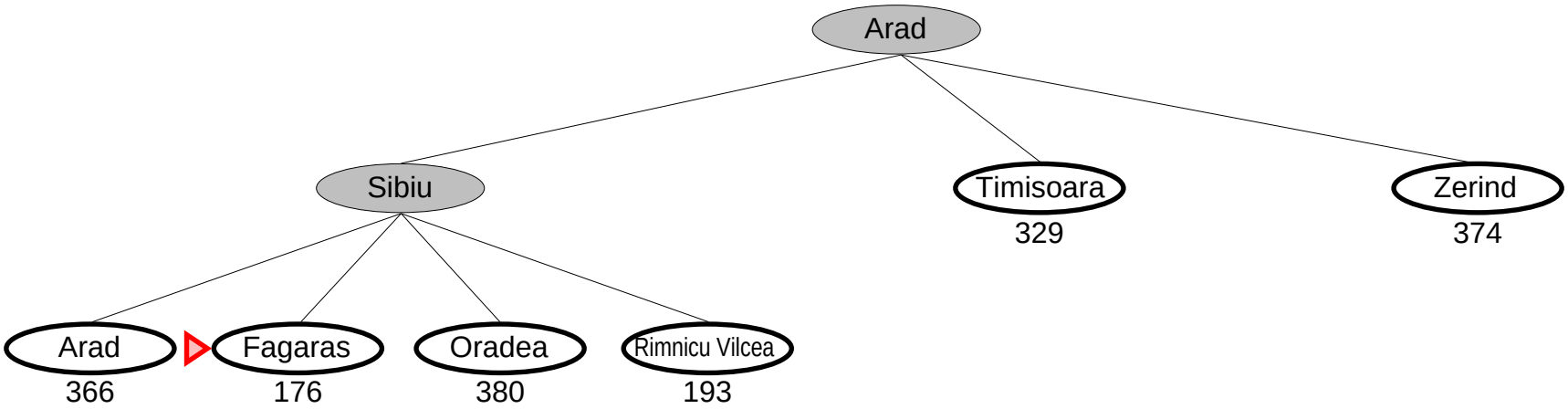
- Bewertungsfunktion $h(n)$ (Heuristik):
  Schätzung der Kosten von n zum nächsten Ziel
- z.B. $h_{\mathrm{LL}}(n) =$ Luftlinienabstand von n nach Bukarest
- Greedy-Suche expandiert Knoten der am nächsten am Ziel *scheint*

Beispiel: Greedy-Suche

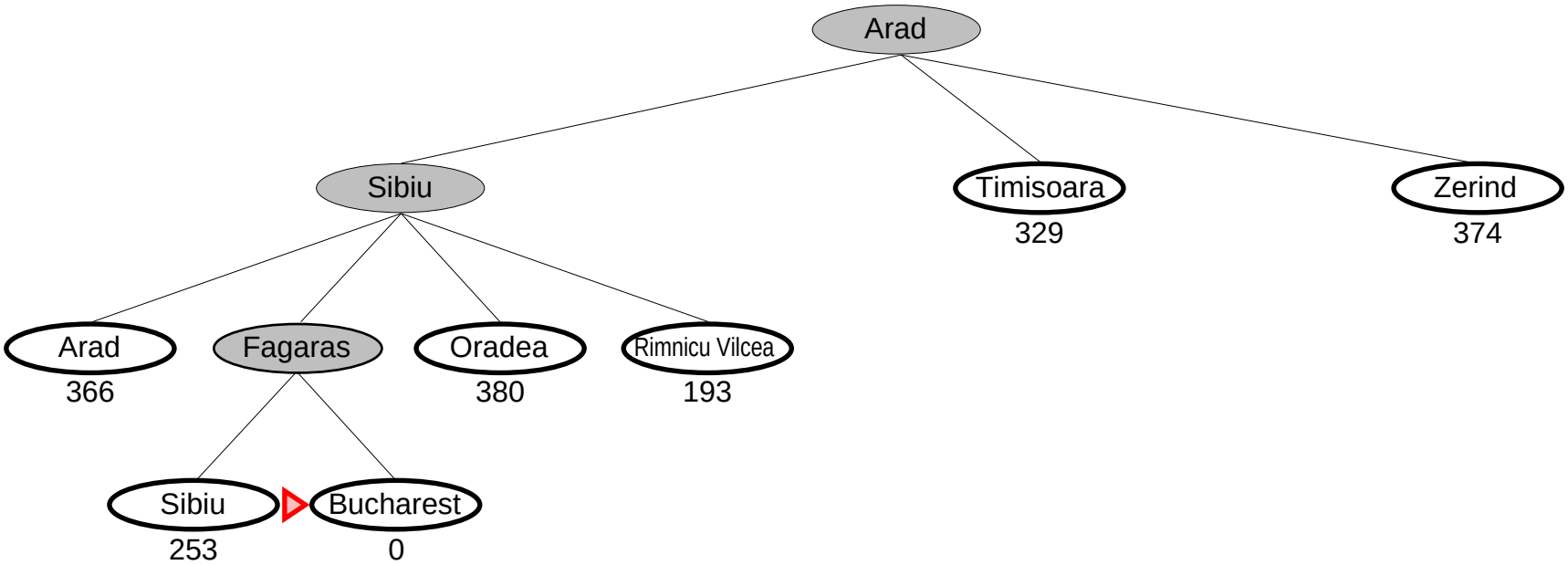
Beispiel: Greedy-Suche



Beispiel: Greedy-Suche



Beispiel: Greedy-Suche



Greedy-Suche: Eigenschaften

- vollständig:
  - nein, kann in Schleifen hängenbleiben, z.B.
    Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$ ...
  - ja, für endliche Räume bei Vermeidung sich wiederholender Zustände im Pfad
- Zeit: $O(b^m)$, mit guter Heuristik drastische Verbesserung
- Speicher: $O(b^m)$ (behält jeden Knoten im Speicher)
- optimal: nein

Übersicht

A*-Algorithmus

Idee: vermeide Expansion bereits teuer expandierter Pfade
Bewertungsfunktion $f(n) = g(n) + h(n)$

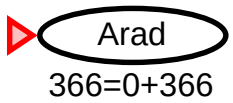
- $g(n)$ bereits aufgenommene Kosten um n zu erreichen
- $h(n)$ geschätzte Kosten von n zum Ziel
- $f(n)$ geschätzte Gesamtkosten des Pfades durch n zum Ziel

A*-Algorithmus benutzt *zulässige Heuristik*

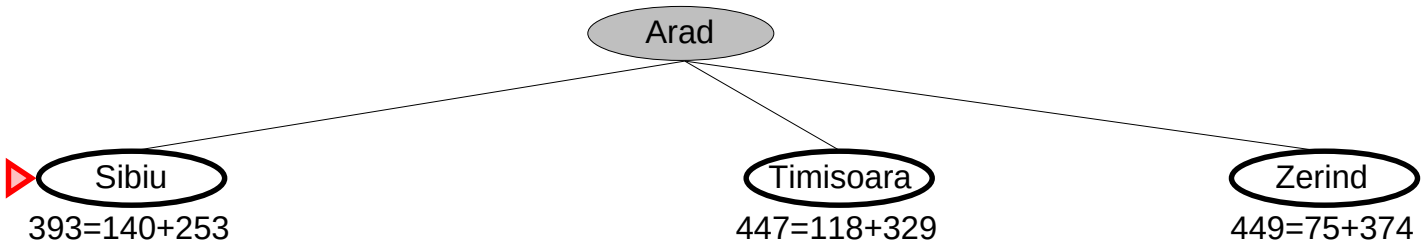
- also, $h(n) \leq h^*(n)$ wobei $h^*(n)$ **wahren** Kosten von n
- auch verlangt: $h(n) \geq 0$, also $h(G) = 0$ für beliebiges Ziel G
- z.B. $h_{\mathrm{LL}}(n)$ überschätzt wirkliche Wegstrecke nie!

Satz: A*-Algorithmus ist optimal.

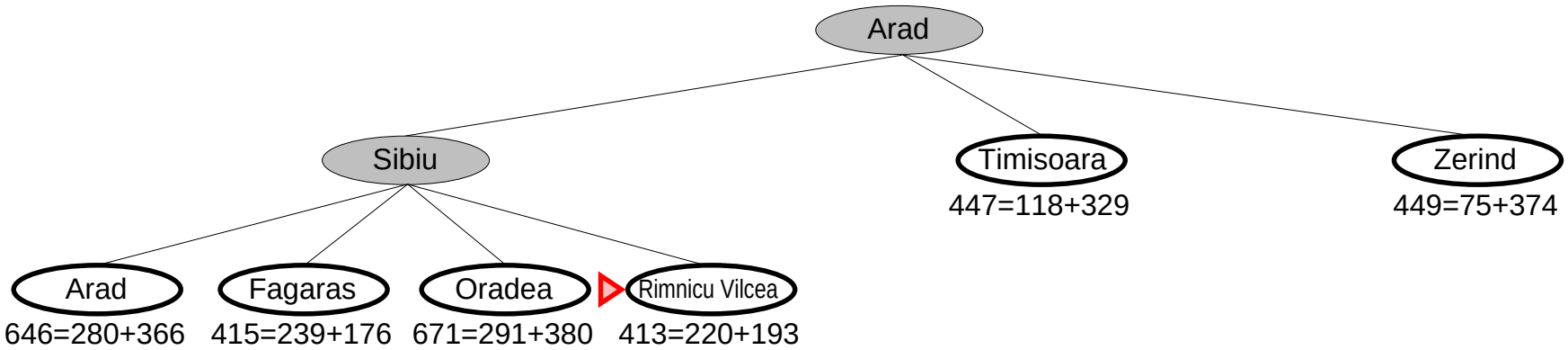
Beispiel: A*-Algorithmus



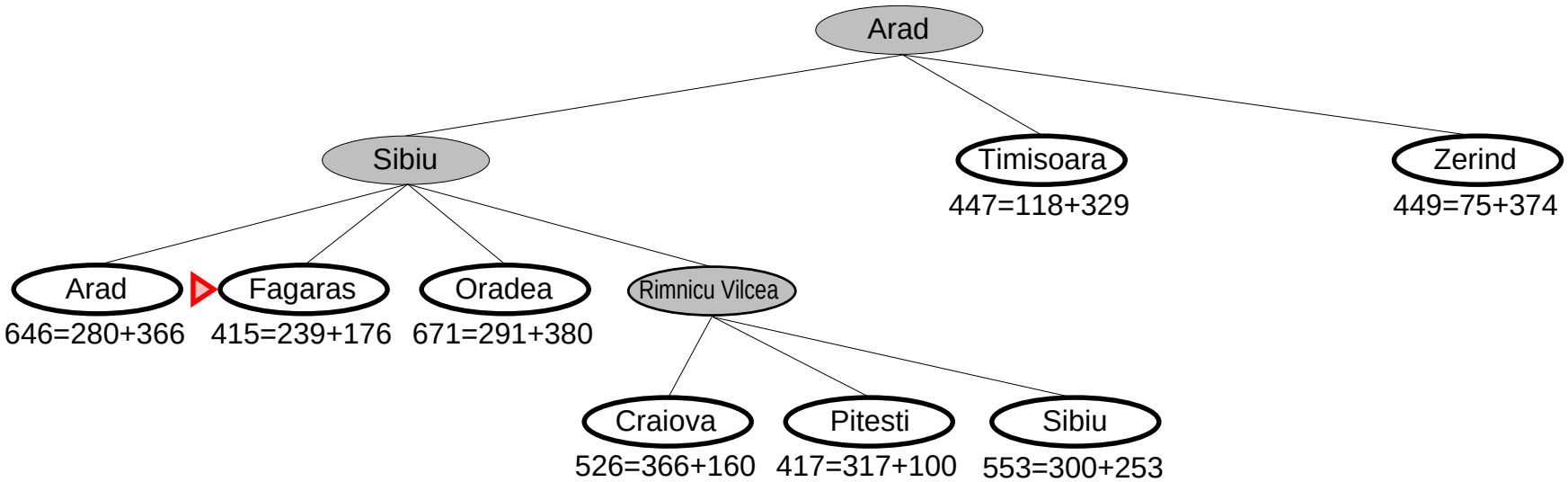
Beispiel: A*-Algorithmus



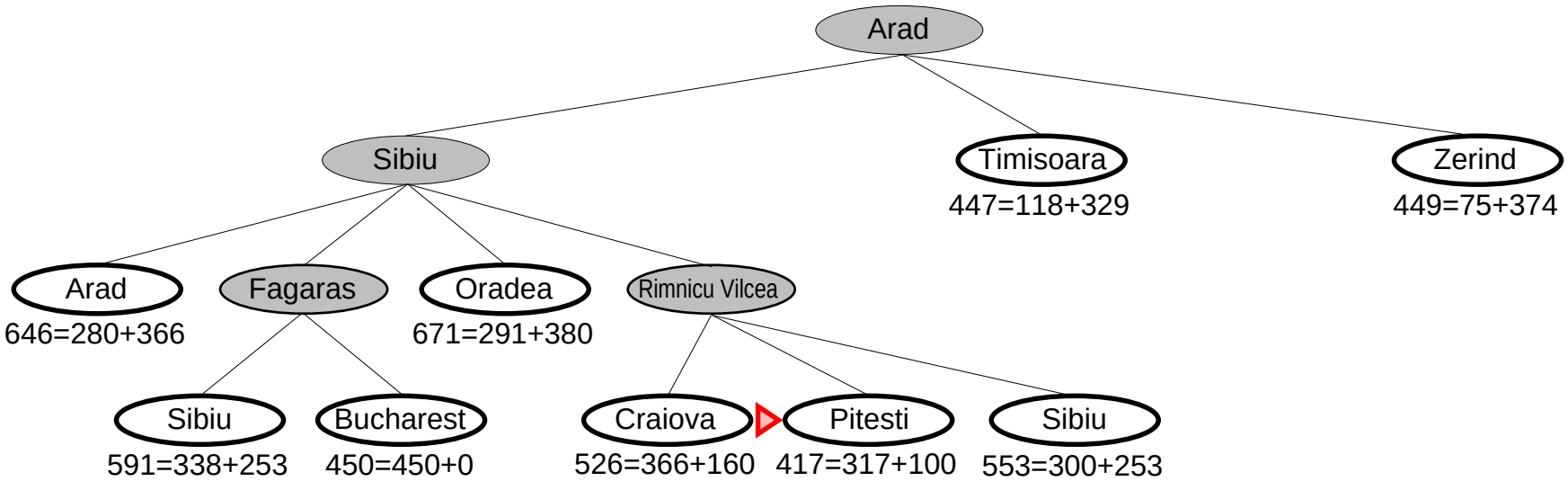
Beispiel: A*-Algorithmus



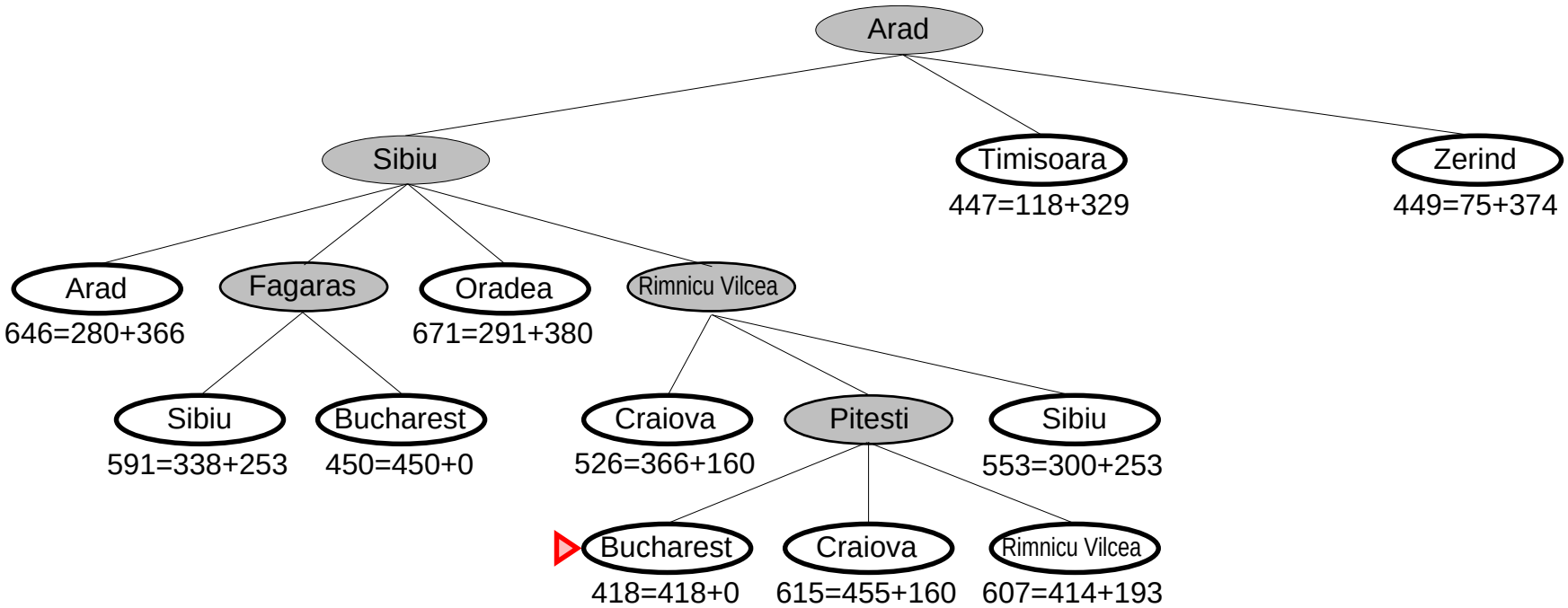
Beispiel: A*-Algorithmus



Beispiel: A*-Algorithmus



Beispiel: A*-Algorithmus



A*-Algorithmus: Gegeben

- Startzustand z_0
- Menge $O = \{o_1, \ldots, o_n\}$ von Operationen:
  liefern zu gegebenem Zustand Nachfolgezustand
  - i.A. nicht alle Operationen auf alle Zustände anwendbar
  - Operation liefert speziellen Wert $\perp$ (undefiniert) statt neuem Zustand, falls nicht anwendbar
- reellwertige Funktion `costs`:
  liefert für jede $o_i \in O$ zugehörigen Kosten
  - u.U. hängen Kosten vom Zustand ab
    (`costs` kann auch zweistellig sein)
- reellwertige Heuristikfunktion h
- Funktion `goal` stellt fest, ob Zustand = Ziel

A*-Algorithmus: Ablauf I

1. erzeuge (gericht.) Graphen $G = \{V, E\}$ mit $V := \{z_0\}$, $E := \emptyset$
   (G stellt den besuchten Teil des Suchraums und die besten bekannten Wege zum Erreichen eines Zustandes dar)

2. erzeuge Menge `open` mit $\texttt{open} := \{z_0\}$
   (`open` enthält die erreichten Zustände mit noch nicht erzeugten Nachfolgern)

3. erzeuge leere Menge `closed`
   (`closed` enthält die erreichten Zustände mit bereits erzeugten Nachfolgern)

A*-Algorithmus: Ablauf II

4. erzeuge Abbildung $g : V \to \mathbb{R}$ mit $z_0 \mapsto 0$ und sonst undefiniert (sog. Tiefenfaktor g: gibt Kosten der besten gefundenen Operationenfolgen zum Erreichen eines Zustandes von z_0 an)

A*-Algorithmus: Ablauf III

5. erzeuge Abbildung $e : V \to O$ für alle Zustände undefiniert
(e baut Lösung des Problems auf: e gibt an, durch welche Operationen ein Zustand von seinem Vorgänger aus erreicht wird)

6. wähle $z \in \texttt{open}$ mit $z \in \left\{x \mid f(x) = \min_{y \in \texttt{open}} f(y)\right\}$ wobei $f = g + h$
(wähle „erfolgversprechendsten“ Zustand gemäß h)

7. falls $\texttt{goal}(z)$, dann Lösung gefunden $\Rightarrow$ lese Pfad aus G ab

8. entferne z aus open, d.h. $\texttt{open} := \texttt{open} \setminus \{z\}$
(Nachfolger von z im folgenden Schritt erzeugt)

A*-Algorithmus: Ablauf IV

9. für alle $o \in O$:

- $x := o(z)$ und $c := g(z) + \mathrm{costs}(o)$
- falls $x \neq \perp$, dann
  - falls $x \notin \texttt{open} \cup \texttt{closed}$, dann
    - ▷ $\texttt{open} := \texttt{open} \cup \{x\}$, $e(x) := o$, $g(x) = c$
    - ▷ erweitere G durch $V := V \cup \{x\}$ und $E := E \cup \{(z, x)\}$
  - falls $x \in \texttt{open} \cup \texttt{closed}$ und $c < g(x)$, dann
    - ▷ $e(x) := o$ $g(x) = c$
    - ▷ ersetze Vorgänger durch

    $E := (E \setminus \{(a, b) \mid b = x\}) \cup \{(z, x)\}$
    - ▷ falls $x \in \texttt{closed}$, dann prüfe rekursiv alle Zustände, die sich von x erreichen lassen und ersetze Vorgänger ggf. durch günstigere Vorgänger

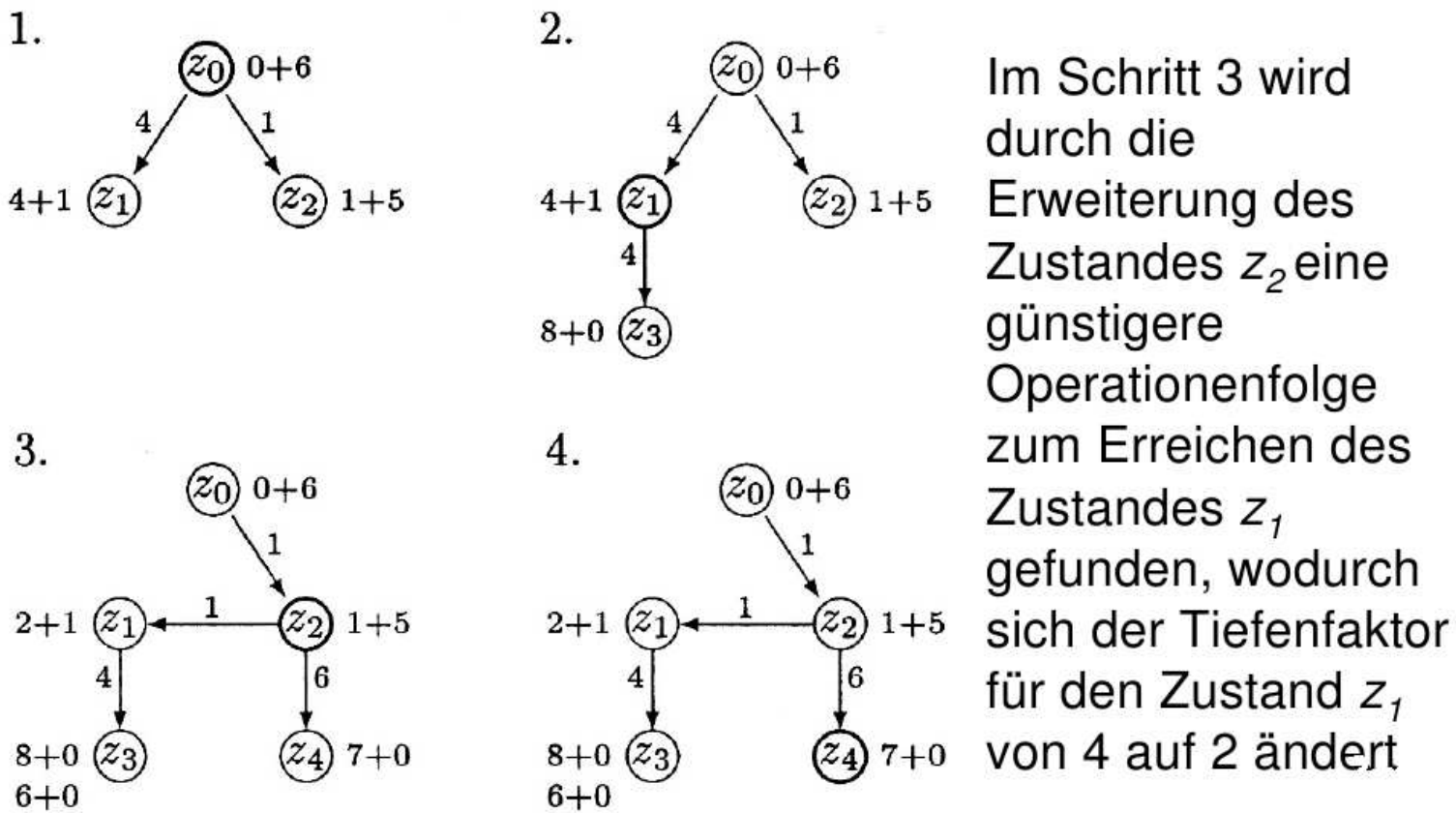
A*-Algorithmus: Ablauf V

10. nimm Zustand z in `closed` auf, also
$\texttt{closed} := \texttt{closed} \cup \{z\}$
(Nachfolger von z im vorhergehenden Schritt erzeugt)

11. falls `open` leer, dann Problem unlösbar $\Rightarrow$ A* bricht ab, andernfalls gehe zu Schritt 6

A*-Algorithmus: Anpassung des Tiefenfaktors

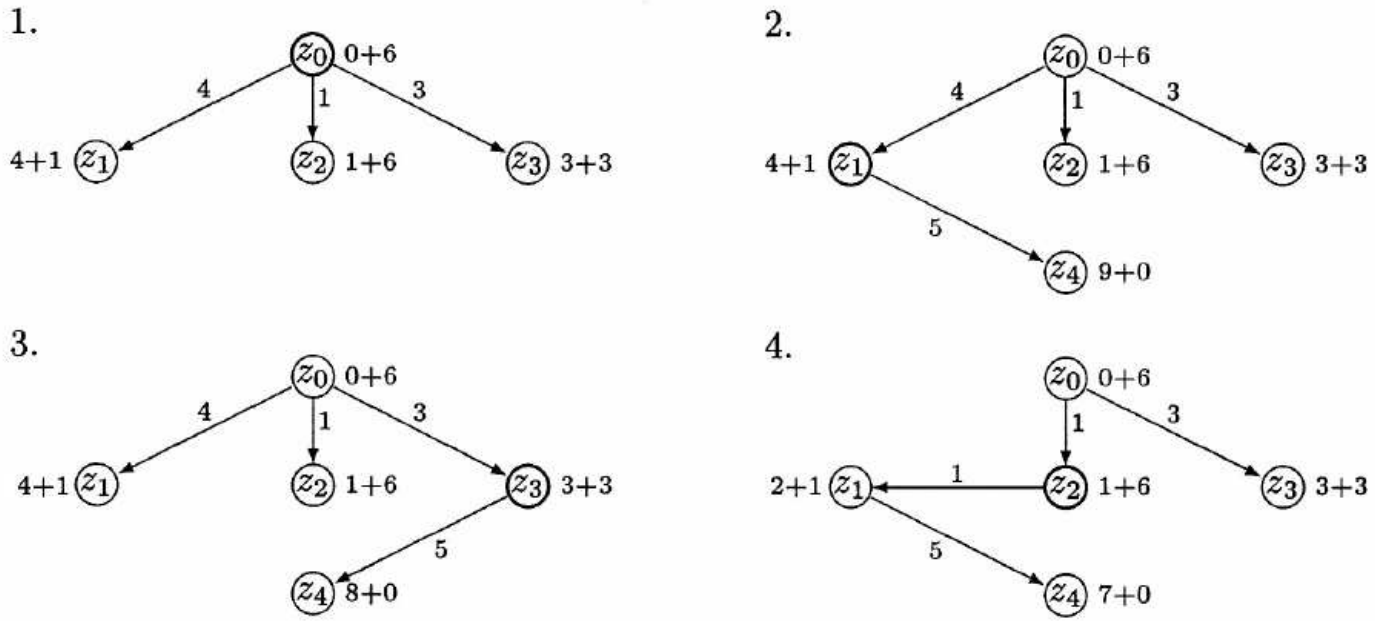
Notwendigkeit der Anpassung von Nachfolgezuständen (**9.**):



Im Schritt 3 wird durch die Erweiterung des Zustandes z_2 eine günstigere Operationenfolge zum Erreichen des Zustandes z_1 gefunden, wodurch sich der Tiefenfaktor für den Zustand z_1 von 4 auf 2 ändert

A*-Algorithmus: Anpassung des Tiefenfaktors

Notwendigkeit der (rekursiven) Anpassung *aller* Zustände (**9.**):



Im Schritt 3 wird durch die Erweiterung des Zustandes z_3 ein günstigerer Knoten z_4 gefunden. Daher wird die Kante (z_1,z_4) entfernt und stattdessen die Kante (z_3,z_4) eingefügt. Die Erweiterung von z_2 in Schritt 4 liefert aber einen kürzeren Weg nach z_1 (und z_4) und erfordert neue Zuordnung der Kante (kein Nachfolger!

A*-Algorithmus: Eigenschaften

- vollständig: ja, solange wie es unendlich mehr Knoten mit $f \leq f(G)$ gibt
- Zeit: exponentiell in [relativer Fehler in $h \times$ Länge der Lösung]
- Speicher: behält jeden Knoten im Speicher
- optimal: ja, A* kann nicht f_{i+1} expandieren bis f_i beendet

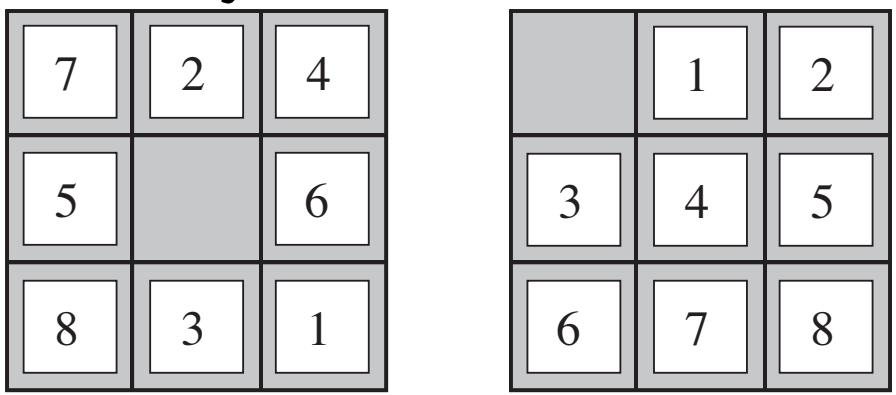
A* expandiert alle Knoten mit $f(n) < C^*$
A* expandiert einige Knoten mit $f(n) = C^*$
A* expandiert keine Knoten mit $f(n) > C^*$

Zulässige Heuristiken

z.B. für 8-Puzzle:

$h_1(n) =$ Anzahl der Plättchen an falscher Position

$h_2(n) =$ Summe der Manhattan-/City-Block-Abstände zw. falscher und gewünschter Position jedes Plättchens



$h_1(S) = 6$, h_1 für Startzustand (6 Plättchen an falscher Position)

$h_2(S) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14$, h_2 für Startzustand

Dominanz

wenn h_1, h_2 zulässig und $h_2(n) \geq h_1(n)$ für alle n, dann $h_2 \succ h_1$ (h_2 **dominiert** h_1)
$\Rightarrow h_2$ ist besser als h_1
typische Suchkosten:

- sei d Tiefe der Lösung mit geringsten Kosten
- für $d = 14$: iterierte Tiefensuche ca. $3,5 \cdot 10^6$ Knoten
  $A^*(h_1) = 539$ Knoten, $A^*(h_2) = 113$ Knoten
- für $d = 24$: iterierte Tiefensuche ca. $54 \cdot 10^9$ Knoten
  $A^*(h_1) = 39135$ Knoten, $A^*(h_2) = 1641$ Knoten

Satz: Gegeben 2 zulässige Heuristiken h_a, h_b.

$$h(n) = \max\{h_a(n), h_b(n)\}$$

ist auch zulässig und dominiert h_a, h_b.

Relaxierte Probleme

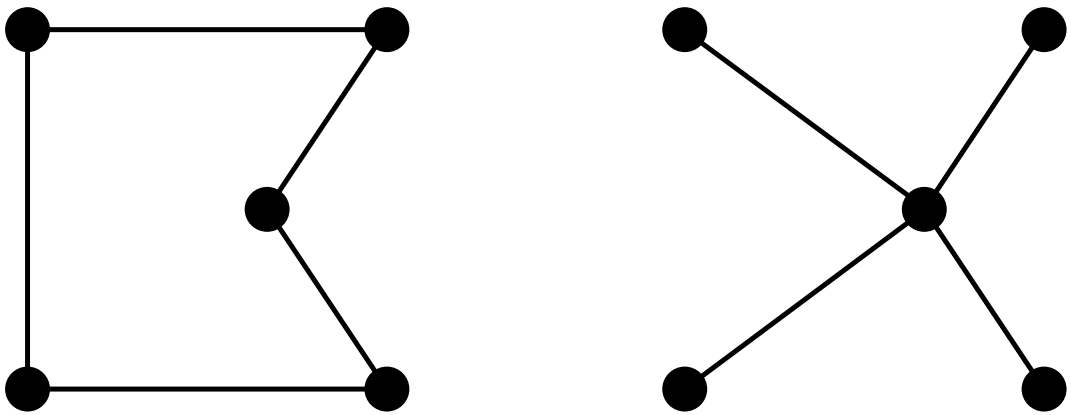
Wie erzeugt man zulässige Heuristiken?
Idee: Konstruktion **exakter** Lösungen einer relaxierten Version des Problems $\Rightarrow$ nutze Kosten dieser Lösung als Heuristik

- falls Regeln des 8-Puzzles relaxiert, sodass Plättchen **überall** hin können, dann kürzeste Lösung mit $h_1(n)$
- falls Regeln des 8-Puzzles relaxiert, sodass Plättchen **zu jedem benachbarten Feld** können, dann kürzeste Lösung mit $h_2(n)$

Kosten der optimalen Lösung eines relaxierten Problems $\not>$
Kosten der optimalen Lösung des realen Problems

Relaxierte Probleme: TSP

Problem des Handlungsreisenden (engl. traveling salesman problem):
Finde kürzeste Rundreise, die alle Städte genau 1x besucht!



minimal aufspannender Baum (Berechnung in $O(n^2)$) ist untere Schranke der kürzesten (offenen) Rundreise

Zusammenfassung

- Heuristikfunktionen schätzen Kosten des kürzesten Pfads
- gute Heuristiken können Suchkosten dramatisch reduzieren
- Greedy-Suche expandiert kleinstes h
  - unvollständig und nicht immer optimal
- A*-Algorithmus expandiert kleinstes $g + h$
  - vollständig und optimal
  - auch optimal effizient (bis auf Unentschieden, für Vorwärtssuche)
- Erzeugung zulässiger Heuristiken durch exakte Lösungen relaxierter Probleme

Übersicht

Spiele vs. Suchprobleme

„unberechenbarer“ Gegner ⇒ Lösung = Strategie, die Zug für jede mögliche gegnerische Antwort spezifiziert

Zeitbegrenzung ⇒ unwahrscheinlich, dass Ziel gefunden wird

⇒ Näherungslösung

Geschichte:

- erste Überlegungen zu spielender Maschine (Babbage 1846)
- Algorithmen für perfektes Spiel (Zermelo 1912, Von Neumann 1944)
- endlicher Horizont, Näherungslösung (Zuse 1945, Wiener 1948, Shannon 1950)
- erstes Schachprogramm (Turing 1951)
- maschinelles Lernen zur Verbesserung der Bewertung (Samuel 1952–57)
- Stutzen erlaubt tiefere Suche (McCarthy 1956)

Arten von Spielen

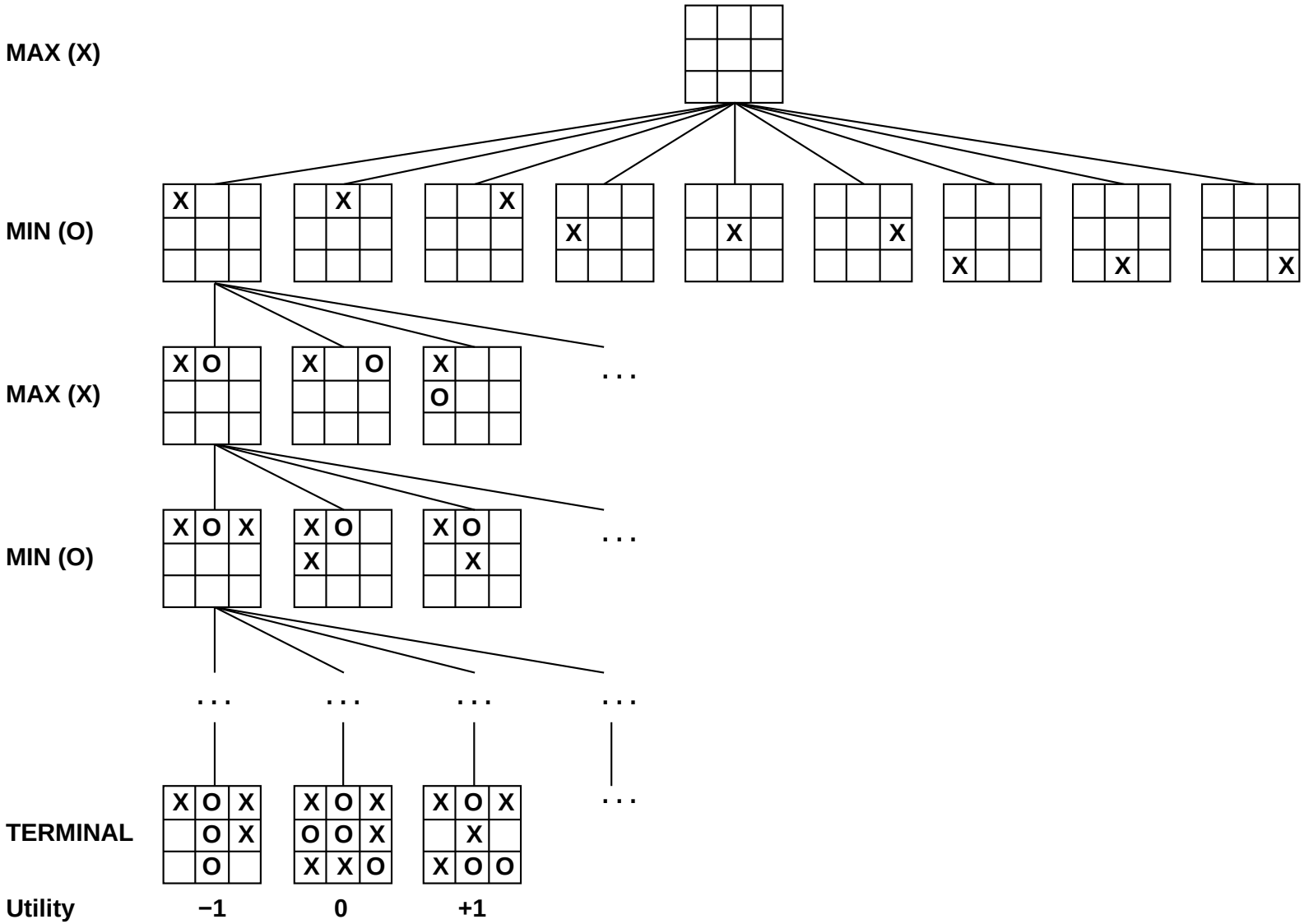
	deterministisch	**Glücksspiel**
vollständige Informationen	Schach, Dame, Go, Othello	Backgammon, Monopoly
unvollständige Informationen	Schiffe versenken, blindes Tic-Tac-Toe	Bridge, Poker, Scrabble, Nuklearer Krieg

blindes Tic-Tac-Toe:

- unvollständige Variante des Standardspiels
- jeder Spieler kann X *und* O setzen
- Gegner erfährt nur welches Feld, aber nicht ob X oder O
- Spieler mit erster Linie mit 3 gleichen Zeichen gewinnt

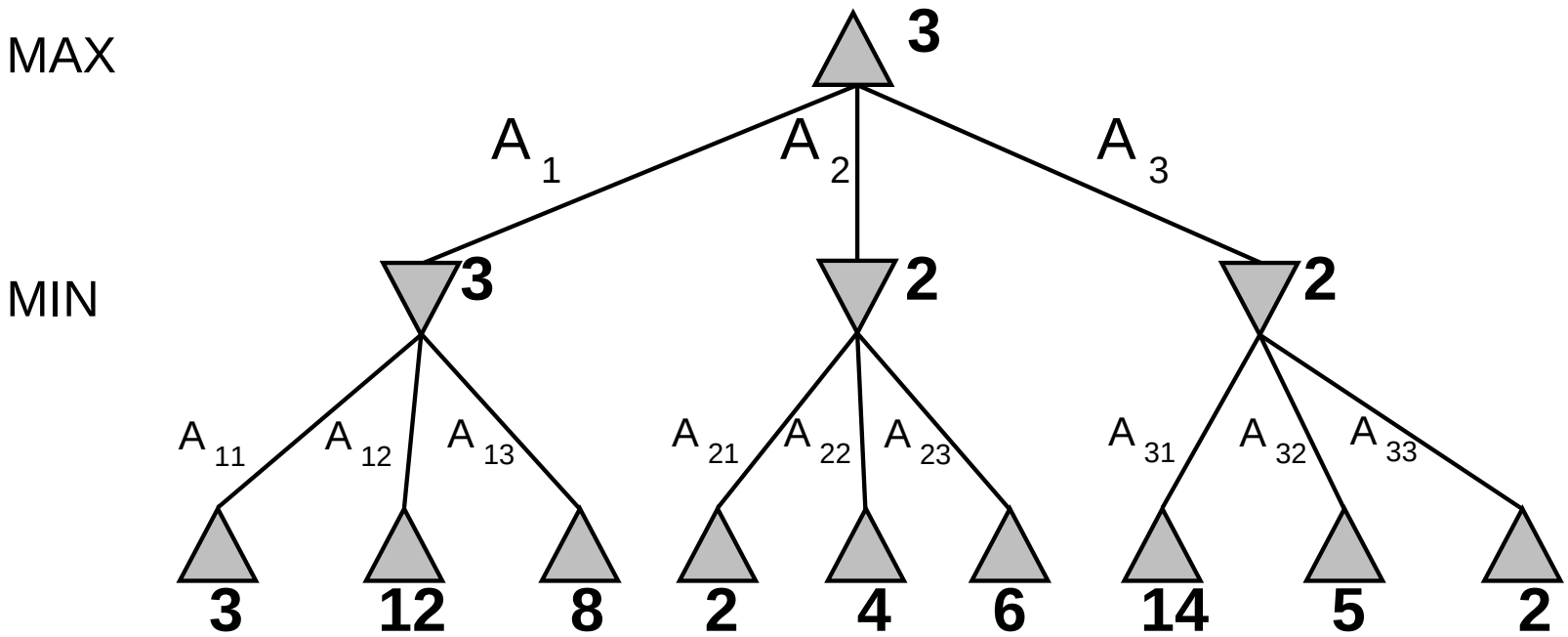
Übersicht

Spielbaum (2 Spieler, deterministisch, Runden)



Minimax

perfektes Spiel für deterministische Spiele mit vollständigen Infos
Idee: wähle Spielzug mit höchstem **Minimax-Wert**
= beste erreichbare Auszahlung gegen besten Spieler
z.B. 2-schichtiges Spiel:



Minimax-Algorithmus

Algorithmus 2 MINIMAX-DECISION

Eingabe: *state*, momentaner Zustand im Spiel
Ausgabe: eine Aktion *action*

1: **return** die Aktion *a* in ACTIONS(*state*), die MIN-VALUE(RESULT(*a*, *state*)) maximiert

Algorithmus 3 MAX-VALUE

```
1: if TERMINAL-TEST(state) {
2:     return UTILITY(state)
3: }
4: v ← −∞
5: for each a, s in SUCCESSORS(state) {
6:     v ←MAX(v,MIN-VALUE(s))
7: }
8: return v
```

Algorithmus 4 MIN-VALUE

```
1: if TERMINAL-TEST(state) {
2:     return UTILITY(state)
3: }
4: v ← ∞
5: for each a, s in SUCCESSORS(state) {
6:     v ←MIN(v,MAX-VALUE(s))
7: }
8: return v
```

MiniMax: Eigenschaften

Zeit- und Speicherkomplexität gemessen anhand von

- b: maximalem Verzweigungsfaktor des Suchbaums
- m: maximaler Tiefe des Zustandsraums (eventuell ∞)

- vollständig: ja, falls Baum endlich

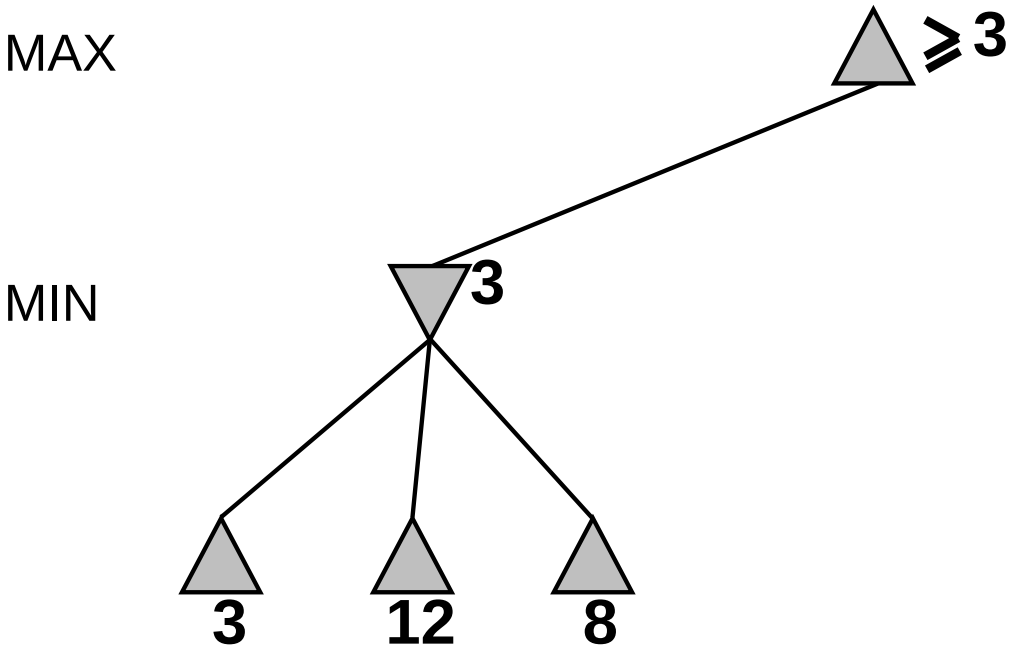
- optimal: ja, gegen optimalen Gegner

- Zeit: $O(b^m)$

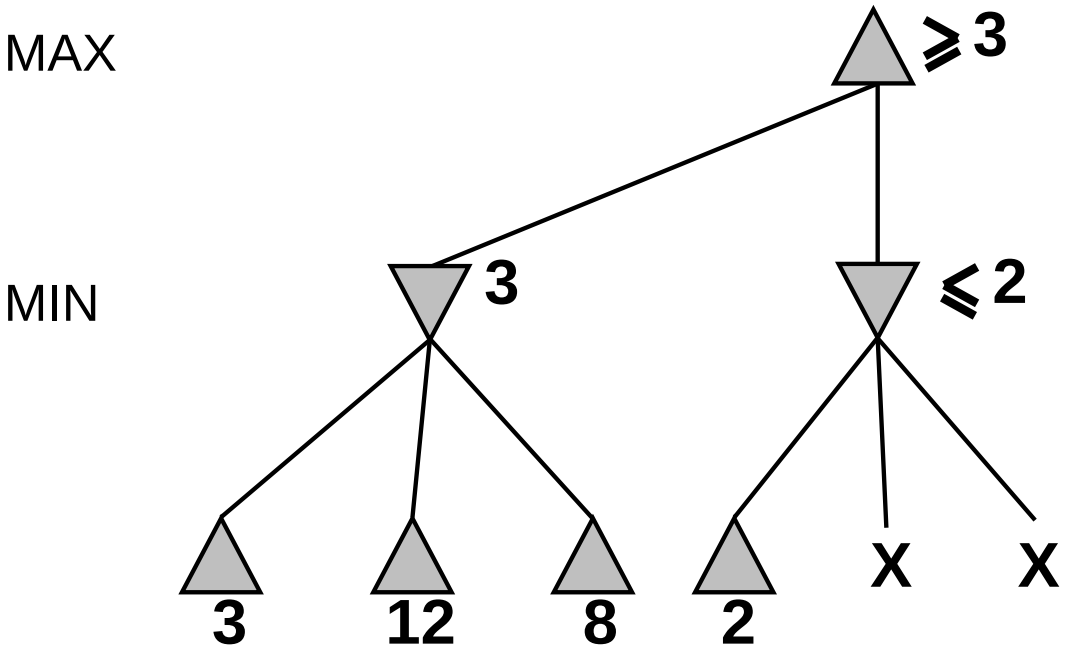
- Speicher: $O(b \cdot m)$ (Tiefensuche)

für Schach: $b \approx 35, m \approx 100$ bei „realistischen“ Spielen
$\Rightarrow$ exakte Lösung absolut nicht berechenbar
Aber: muss jeder Pfad exploriert werden?

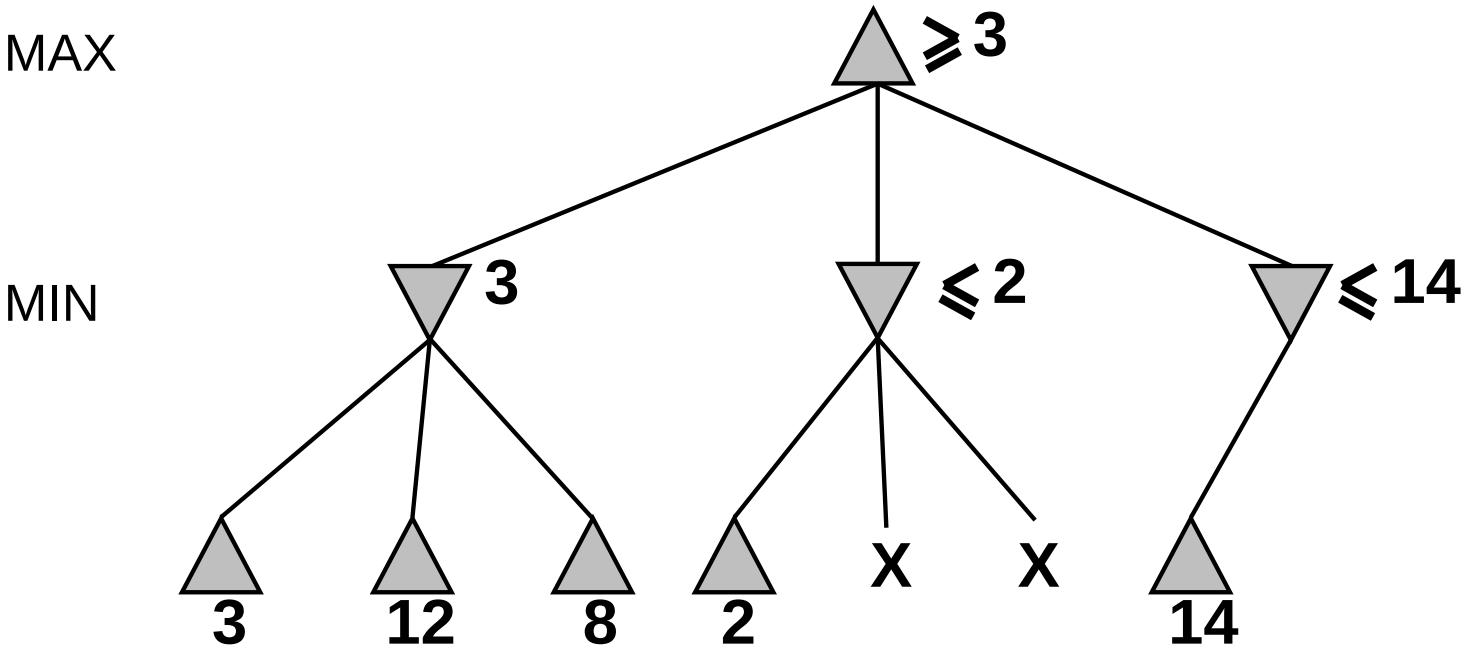
α–β-Stutzen



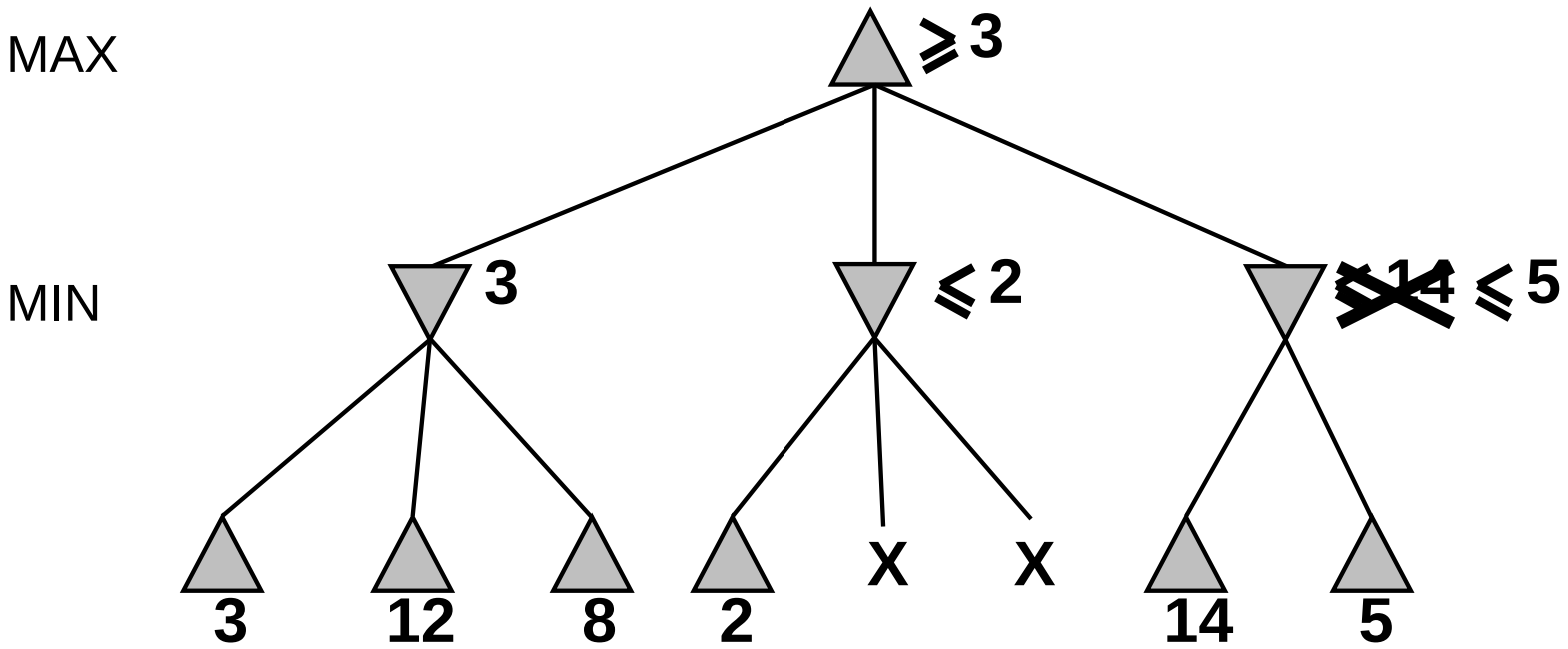
α–β-Stutzen



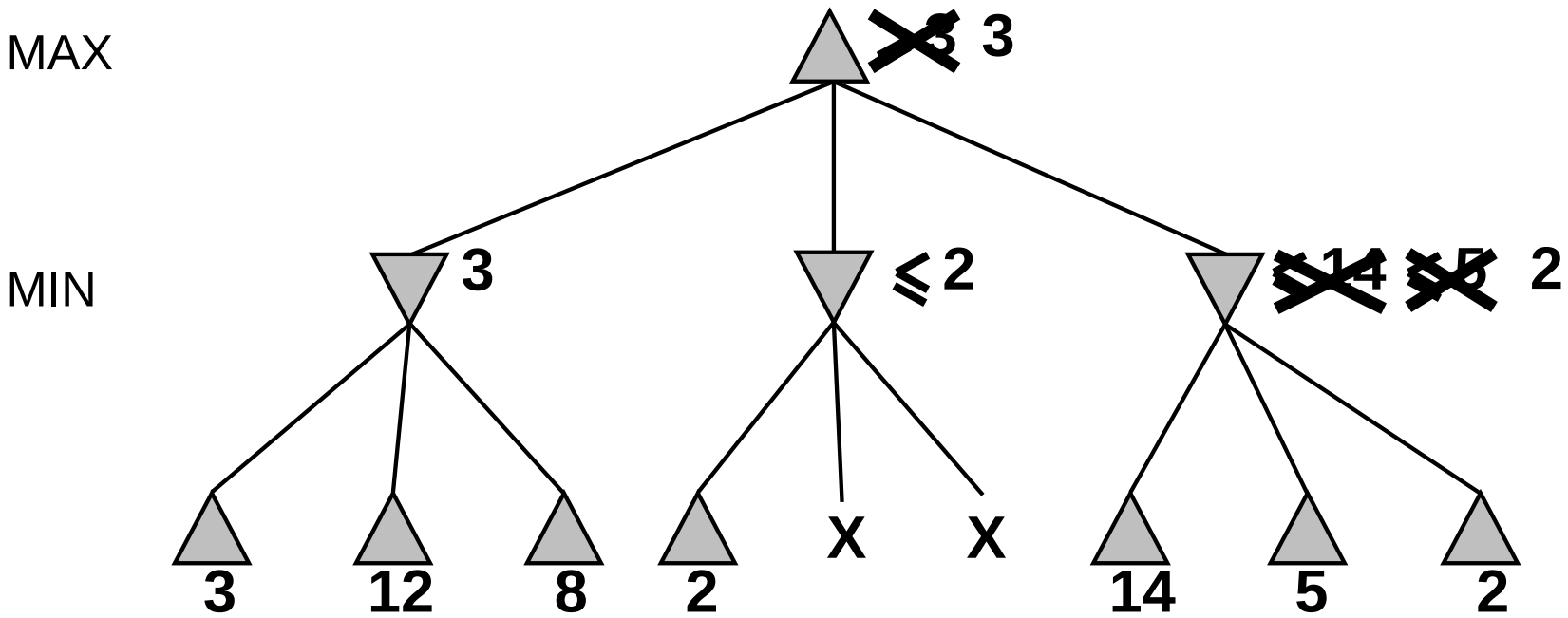
α–β-Stutzen



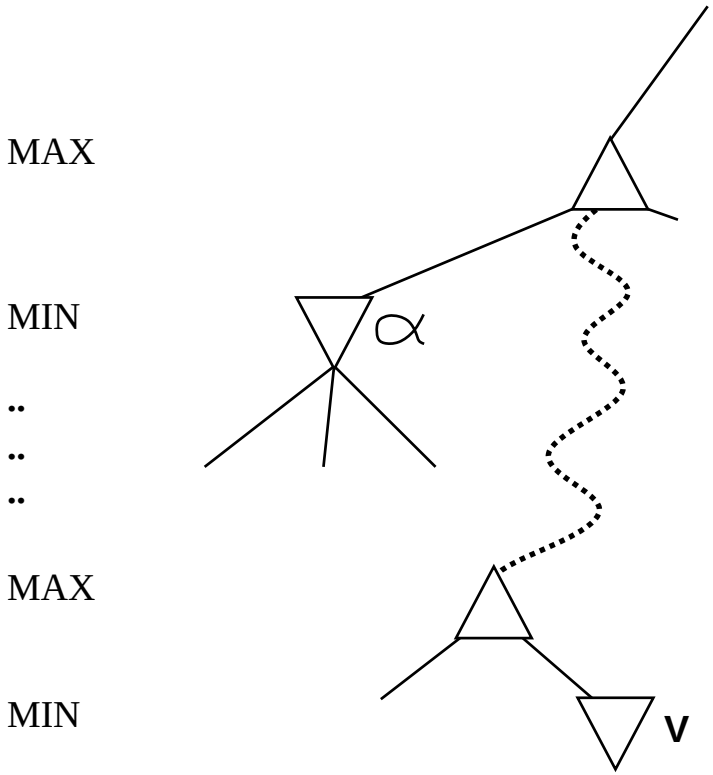
α–β-Stutzen



α–β-Stutzen



Warum heißt es α–β?



α: bester bisher gef. Wert (für $\textsc{Max}$) abseits des aktuellen Wegs
falls V schlechter als α, dann vermeidet $\textsc{Max}$ $V \Rightarrow$ Stutzen des Zweigs
β: analoge Definition für $\textsc{Min}$-Spieler

Der α–β-Algorithmus

Algorithmus 5 ALPHA-BETA-DECISION

```
1: return die Aktion a in ACTIONS(state), die MIN-VALUE(RESULT(a, state)) maximiert
```

Algorithmus 6 MAX-VALUE

Eingabe: *state*, momentaner Zustand im Spiel
α, Wert der besten Alternative für MAX entlang des Pfads zu *state*
β, Wert der besten Alternative für MIN entlang des Pfads zu *state*

```
 1: if TERMINAL-TEST(state) {
 2:     return UTILITY(state)
 3: }
 4: v ← −∞
 5: for each a, s in SUCCESSORS(state) {
 6:     v ←MAX(v,MIN-VALUE(s, α, β))
 7:     if v ≥ β {
 8:         return v
 9:     }
10:     α ← MAX(α, v)
11: }
12: return v
```

Algorithmus 7 MIN-VALUE

```
1: genau wie MIN-VALUE aber mit vertauschten Rollen von α, β
```

α–β-Algorithmus: Eigenschaften

Stutzen hat **keine** Auswirkung auf Endergebnis

- gute Zugordnung verbessert Effektivität des Stutzens
- mit „perfekter“ Ordnung, Zeitkomplexität $= O(b^{m/2})$

$\Rightarrow$ verdoppelt Suchtiefe

- für Schach: immer noch 35^{50} möglich

α–β-Algorithmus: Grenzen

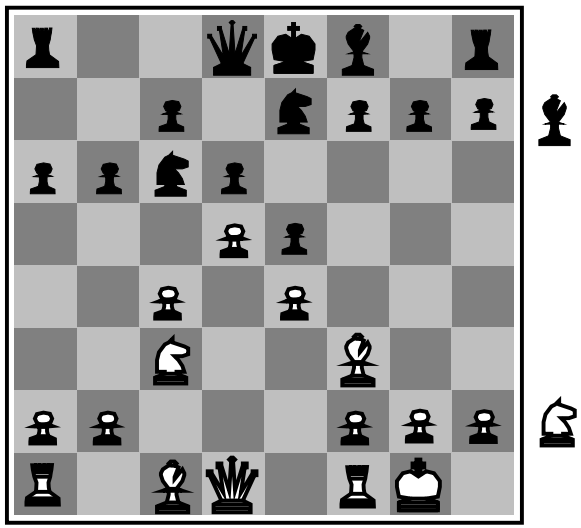
Standard-Ansatz:

- für gewöhnlich: Lösung zu tief im Suchbaum
- Utility mit Werten +1, 0 oder -1 nicht berechnbar
- nutze Cutoff-Test anstelle von Terminal-Test
  z.B. Tiefenbegrenzung (u.U. mit „stiller Suche“—sichtet interessante Pfade tiefer als „stille“)
- nutze Eval (Güteschätzung des Zustands) anstatt Utility:
  Bewertungsfunktion schätzt Begehrtheit einer Stellung

bei 100 Sekunden und 10^4 Knoten/Sekunde:

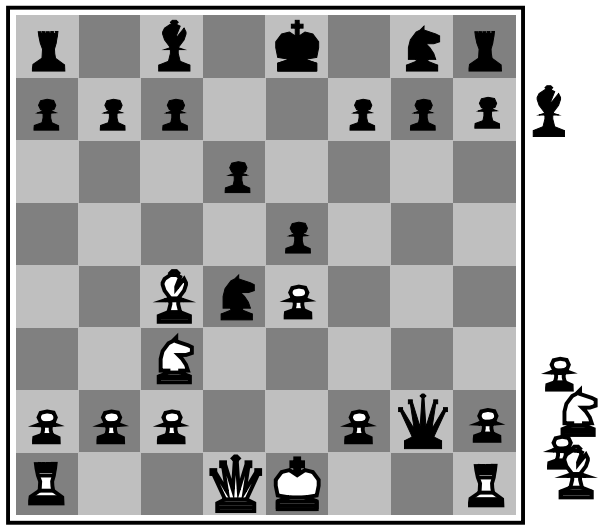
- 10^6 Knoten pro Zug $\approx 35^{8/2}$
- α–β berechnet 8 Halbzüge $\Rightarrow$ ziemlich gutes Schachprogramm

Bewertungsfunktionen



Black to move

White slightly better



White to move

Black winning

für Schach: typischerweise linear gewichtete Summe von n **Merkmalen**

$$\text{Eval}(s) = \sum_{i=1}^{n} w_i \cdot f_i(s)$$

z.B. $f_1(s)$: Differenz von weißen und schwarzen Damen mit $w_1 = 9$

usw.

Deterministische Spiele in der Praxis

Dame:
- 1994 beendete Chinook die 40-jährige Herrschaft des Weltmeisters Marion Tinsley
- Endspieldatenbank mit perfekten Spielen aller Stellungen mit ≤ 8 Steinen ($\geq 443 \cdot 10^9$ Stellungen)

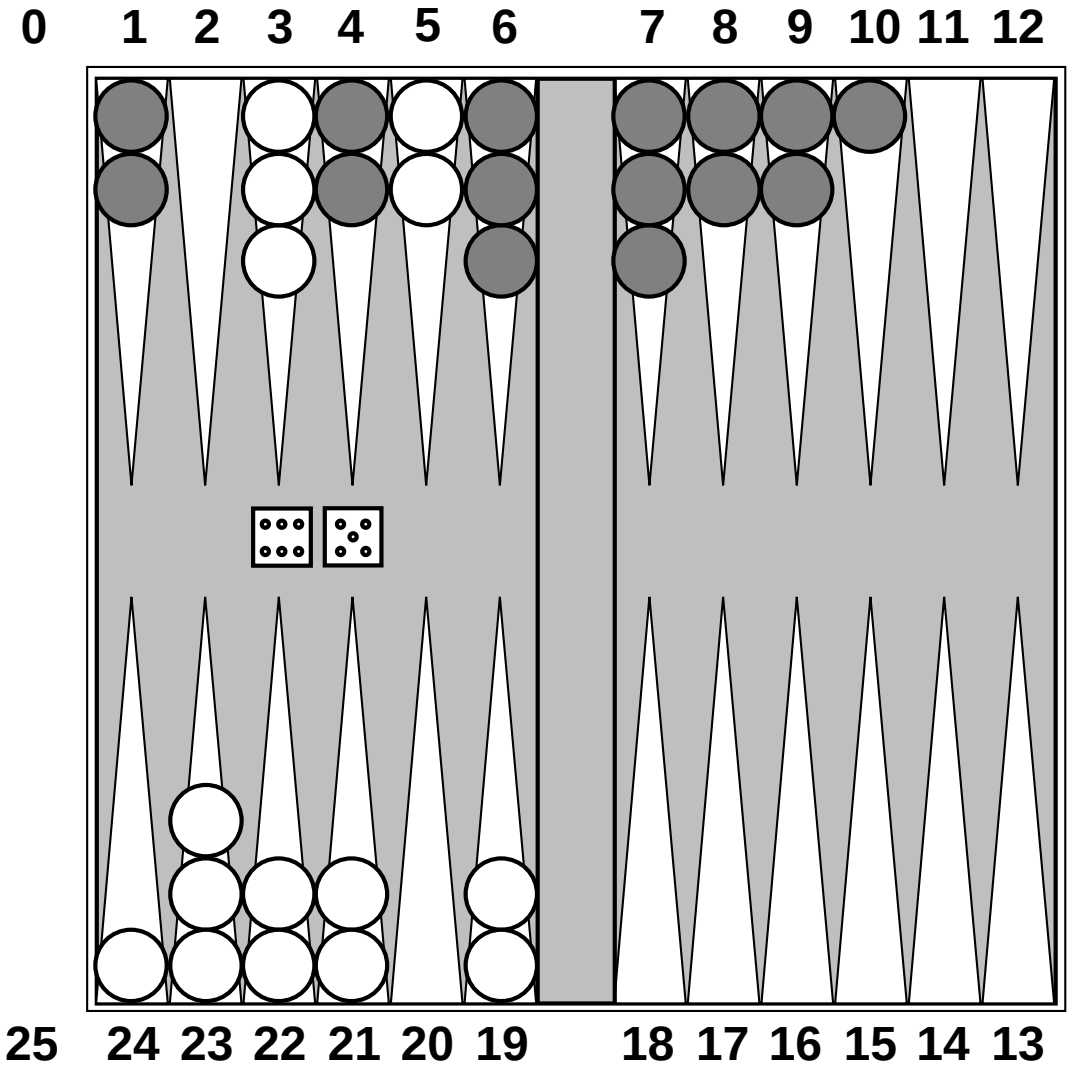
Schach:
- Deep Blue besiegte 1997 Weltmeister Garri Kasparow in 6 Spielen
- $200 \cdot 10^6$ Stellungen/Sekunde, sehr komplizierte Bewertung
- bis zu 40 Halbzüge tief (nicht veröffentlichte Methoden)

Othello: menschl. Meister verweigern (zu guten) Computern Wettbewerb

Go:
- menschl. Meister weigern sich, gegen Computer zu spielen
- $b > 300$, daher Datenbanken mit Mustern für plausible Züge

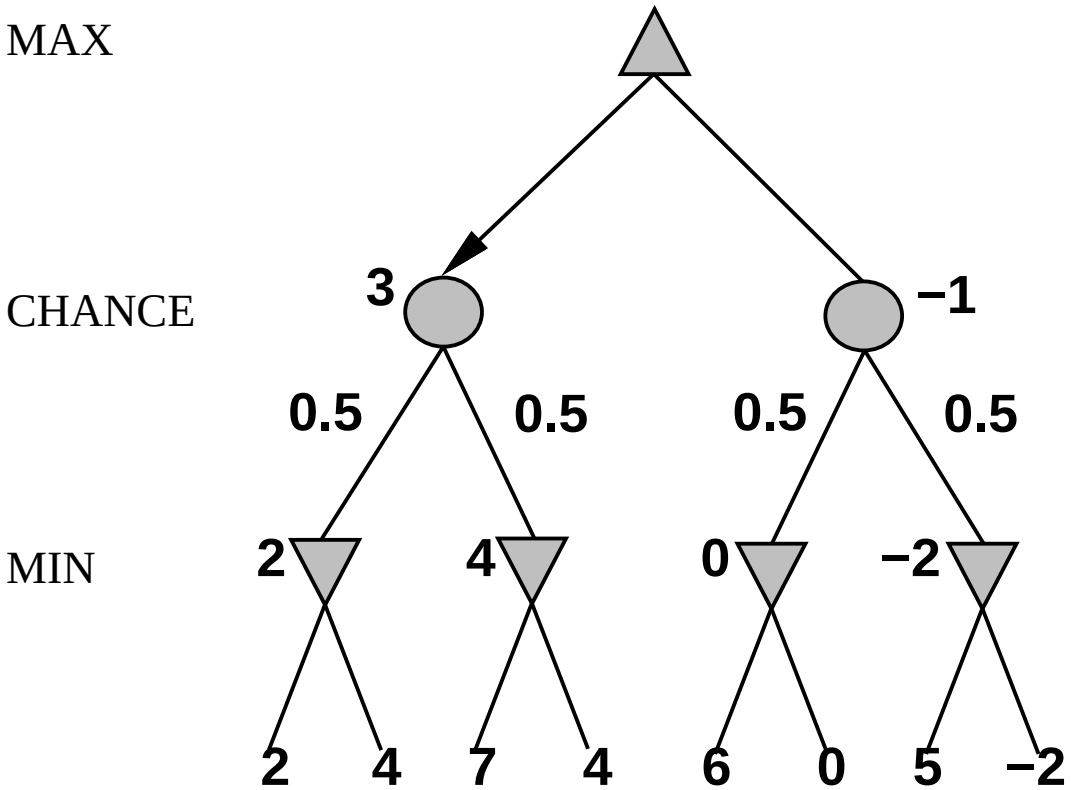
Übersicht

Glücksspiele: Backgammon



Glücksspiele allgemein

Zufall aufgrund von Würfeln, Mischen von Karten, etc.
vereinfachtes Beispiel mit Münzwurf:



Algorithmus für nichtdeterministische Spiele

ExpectiMinimax liefert perfektes Spiel
wie Minimax, nur Zufallsknoten müssen behandelt werden:

```
...
if state is a Max node {
    return highest ExpectiMinimax-Value of Successors(state)
}
if state is a Min node {
    return lowest ExpectiMinimax-Value of Successors(state)
}
if state is a chance node {
    return average of ExpectiMinimax-Value of Successors(state)
}...
```

Nichtdeterministische Spiele in Praxis

jeder Würfelwurf erhöht b: 21 mögliche Würfe mit 2 Würfeln
Backgammon $\approx$ 20 erlaubte Züge (kann 6000 sein mit 1–1 Wurf)

$$\text{Tiefe } 4 = 20 \cdot (21 \cdot 20)^3 \approx 1,2 \cdot 10^9$$

mit steigender Tiefe: Wahrscheinlichkeit sinkt geg. Knoten zu erreichen
$\Rightarrow$ Wert des Vorgriffs wird vermindert
α–β-Stutzen viel weniger effektiv
TDGammon benutzt Tiefe-2-Suche + sehr gute Eval
$\approx$ vgl. Weltmeister

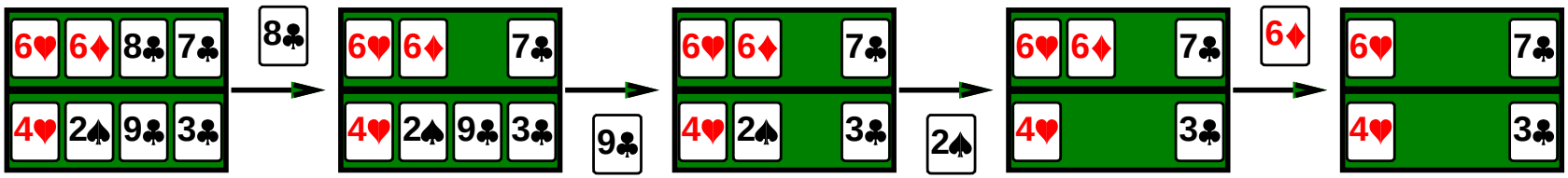
Spiele mit unvollständigen Informationen

z.B. Kartenspiele, bei denen gegnerische Karten anfangs unbekannt
typischerweise: Wahrscheinlichkeiten für jede mögliche Kartenverteilung
Idee: Berechnung der Minimax-Werte jeder Aktion für jede Verteilung
$\Rightarrow$ Wahl der Aktion mit höchstem Erwartungswert über alle Verteilungen
GIB (momentan bestes Bridge-Programm) approximiert Idee durch

- Erzeugung von 100 Verteilungen (basierend auf Ansage)
- Wahl der Aktion, die durchschnittlich die meisten Stiche macht

Beispiel: Bridge

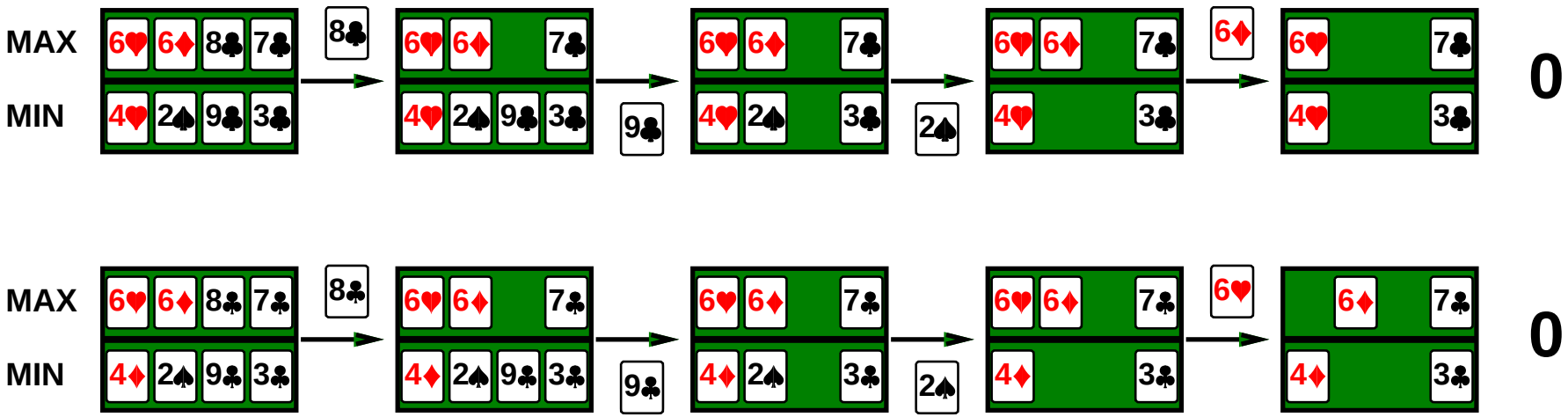
4-Karten-Bridge/Whist/Hearts: MAX spielt zuerst



0

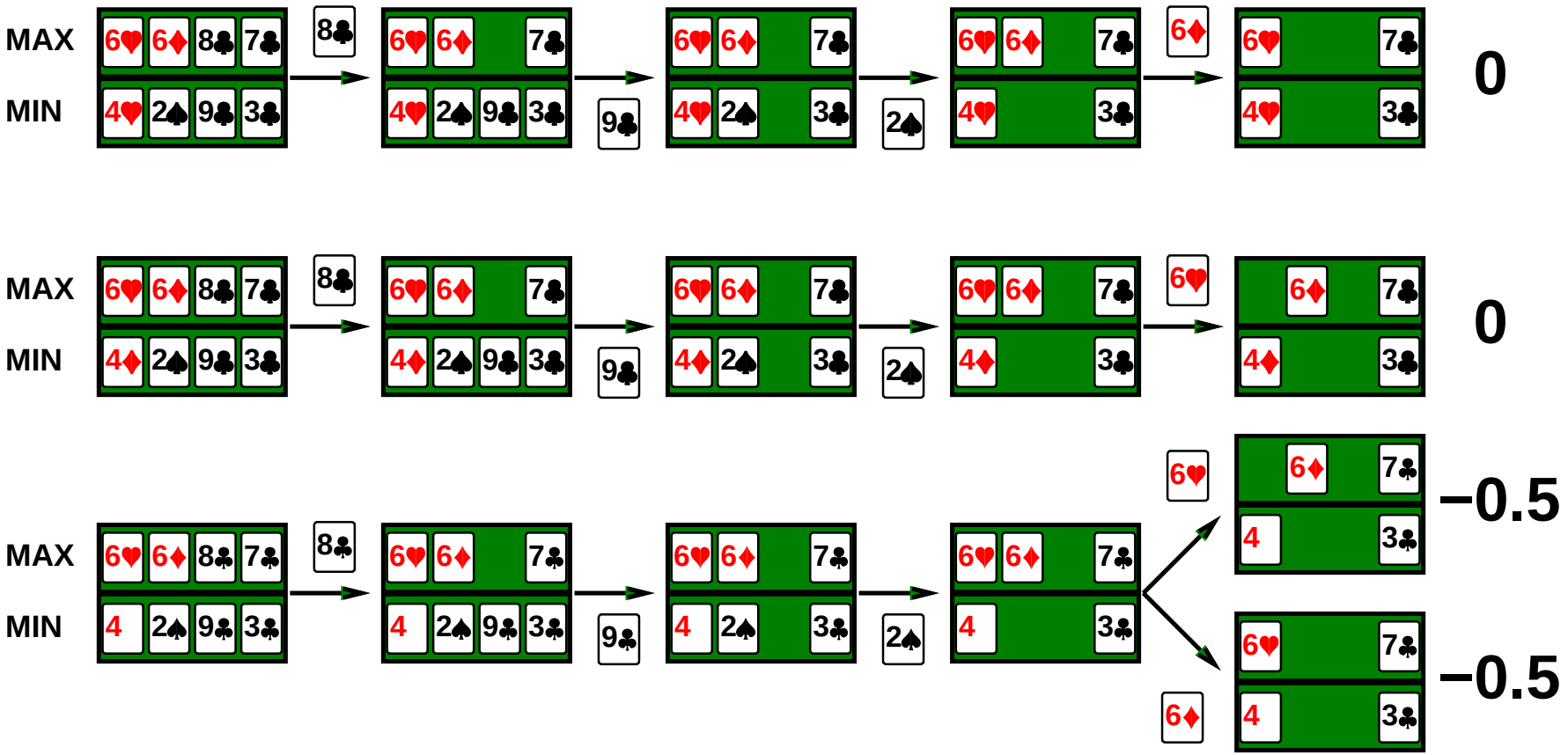
Beispiel: Bridge

4-Karten-Bridge/Whist/Hearts: MAX spielt zuerst



Beispiel: Bridge

4-Karten-Bridge/Whist/Hearts: MAX spielt zuerst



Beispiel

Weg A führt zu einem kleinen Haufen Goldstücke.
Weg B führt zu einer Verzweigung:

- Auf dem linken Abzweig befindet sich ein Haufen Juwelen.
- Auf dem rechten Abzweig wird man von einem Bus überfahren.

Beispiel

Weg A führt zu einem kleinen Haufen Goldstücke.
Weg B führt zu einer Verzweigung:
    Auf dem linken Abzweig befindet sich ein Haufen Juwelen.
    Auf dem rechten Abzweig wird man von einem Bus überfahren.
Weg A führt zu einem kleinen Haufen Goldstücke.
Weg B führt zu einer Verzweigung:
    Auf dem linken Abzweig wird man von einem Bus überfahren.
    Auf dem rechten Abzweig befindet sich ein Haufen Juwelen.

Beispiel

Weg A führt zu einem kleinen Haufen Goldstücke.
Weg B führt zu einer Verzweigung:
    Auf dem linken Abzweig befindet sich ein Haufen Juwelen.
    Auf dem rechten Abzweig wird man von einem Bus überfahren.
Weg A führt zu einem kleinen Haufen Goldstücke.
Weg B führt zu einer Verzweigung:
    Auf dem linken Abzweig wird man von einem Bus überfahren.
    Auf dem rechten Abzweig befindet sich ein Haufen Juwelen.
Weg A führt zu einem kleinen Haufen Goldstücke.
Weg B führt zu einer Verzweigung:
    Durch richtiges Raten findet man einen Haufen Juwelen.
    Durch falsches Raten wird man von einem Bus überfahren.

Saubere Analyse

Intuition, dass Wert einer Aktion Durchschnitt seiner Werte in allen Zuständen ist, ist **falsch**

mit teilweiser Beobachtbarkeit: Wert einer Aktion hängt ab von *Informations- oder Glaubenszustand* des Agenten

Agent kann Baum von Informationszuständen generieren und durchsuchen

führt zu rationalem Verhalten, wie z.B.

- handeln um Informationen zu bekommen,
- signalisieren zu seinem Partner,
- zufällig handeln um Weitergabe von Informationen zu minimieren

Zusammenfassung

Arbeit mit Spielen macht Spaß (ist aber auch gefährlich!)

Illustration verschiedener wichtiger Punkte über KI

- wenn Perfektion unerreichbar, dann approximieren
- gute Idee: nachdenken über das Nachdenken
- Unsicherheit beschränkt Zuweisung von Werten zu Zuständen
- optimale Entscheidungen hängen ab vom Informationszustand, nicht vom wirklichen Zustand

Spiele sind für KI, was Grand Prix für Automobilentwicklung ist!