INF FAKULTÄT FÜR INFORMATIK

Intelligente Systeme

Heuristische Suchalgorithmen

Prof. Dr. R. Kruse **C. Braune**

{rudolf.kruse,christian.braune}@ovgu.de

Institut für Intelligente Kooperierende Systeme
Fakultät für Informatik
Otto-von-Guericke-Universität Magdeburg

Warum heuristische Suchalgorithmen?

Suchprobleme werden häufig durch eine Baumsuche gelöst.

Eine uninformierte Suche muss im schlechtesten Fall alle Knoten des Baumes expandieren.

Unter Ausnutzung von Problemwissen (Mutmaßungen/Heuristiken) kann der Rechenaufwand meistens reduziert werden.

Übersicht

1. **Bestensuche**
   Greedy-Suche

2. **A*-Algorithmus**

Bestensuche

Idee: nutze Bewertungsfunktion für jeden Knoten

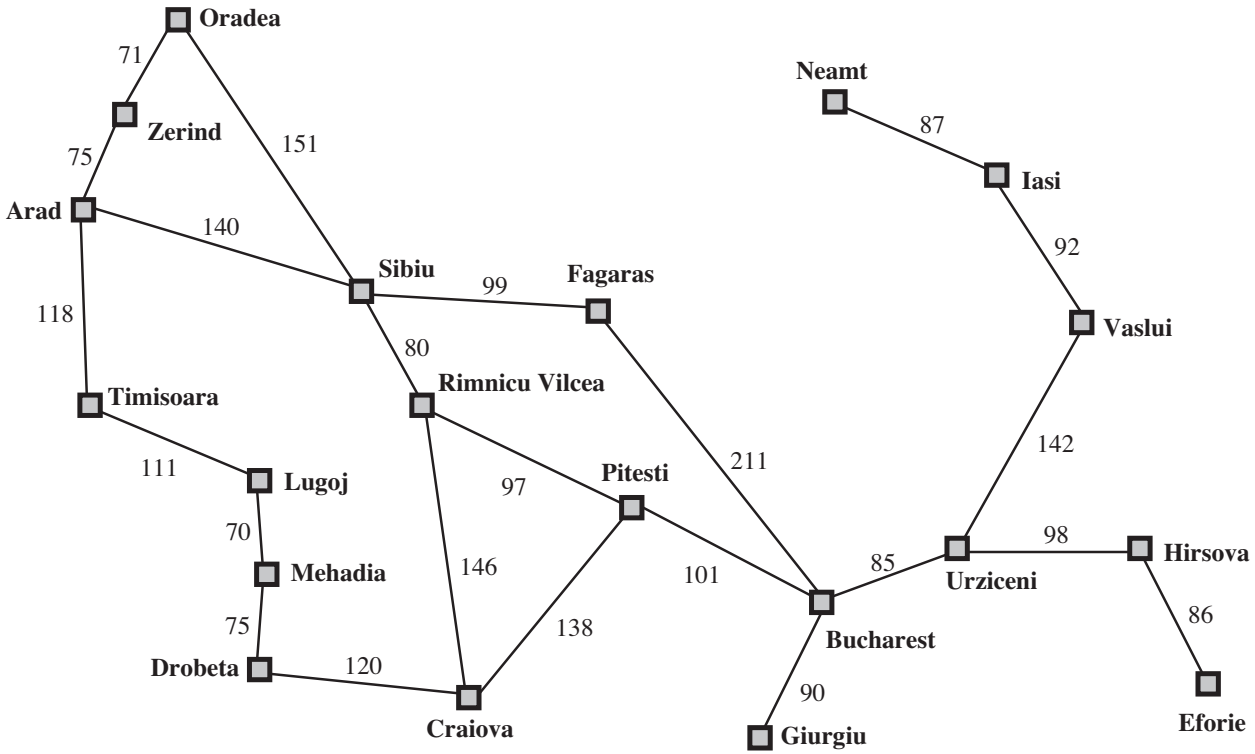
Schätzung, wie „wünschenswert/begehrt“ Knoten ist

Somit: Expansion des am wünschenswertesten (noch nicht expandierten) Knotens

Implementierung:
fringe = Queue absteigend sortiert nach „Begehrtheit“

Spezialfälle: Greedy-Suche, A*-Algorithmus

Beispiel: Routenplanung 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

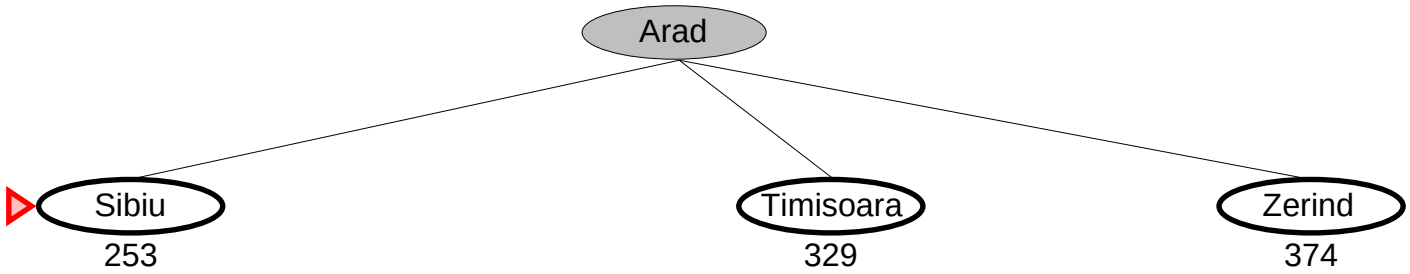
Bewertungsfunktion $h(n)$ (Heuristik):
Schätzung der Kosten von n zum nächsten Ziel

Z.B. $h_{\mathrm{LL}}(n) =$ Luftlinienabstand von n nach Bukarest

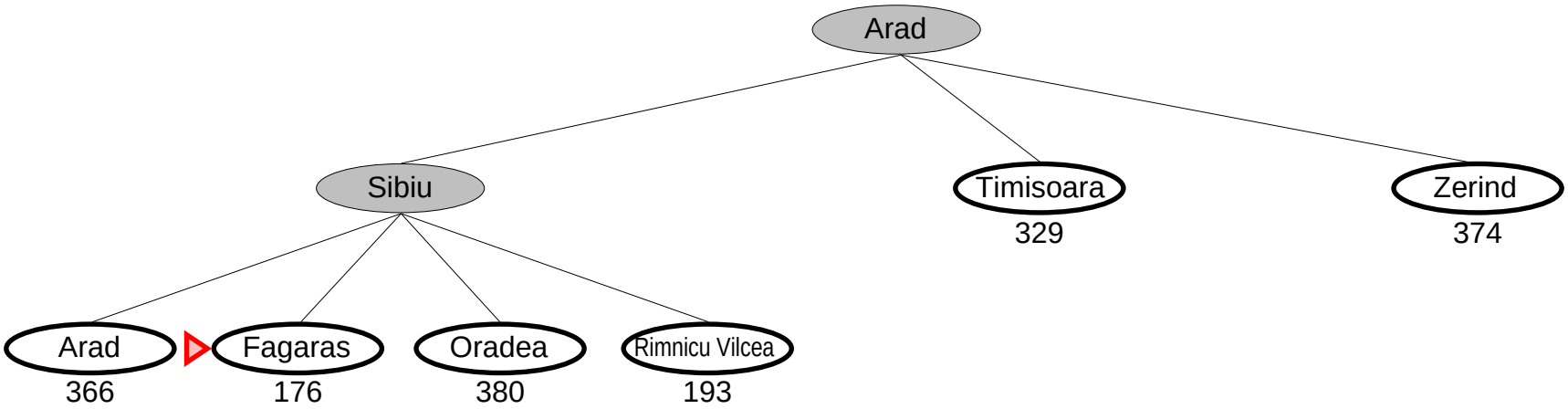
Greedy-Suche expandiert Knoten der am nächsten am Ziel *scheint*

Beispiel: Greedy-Suche

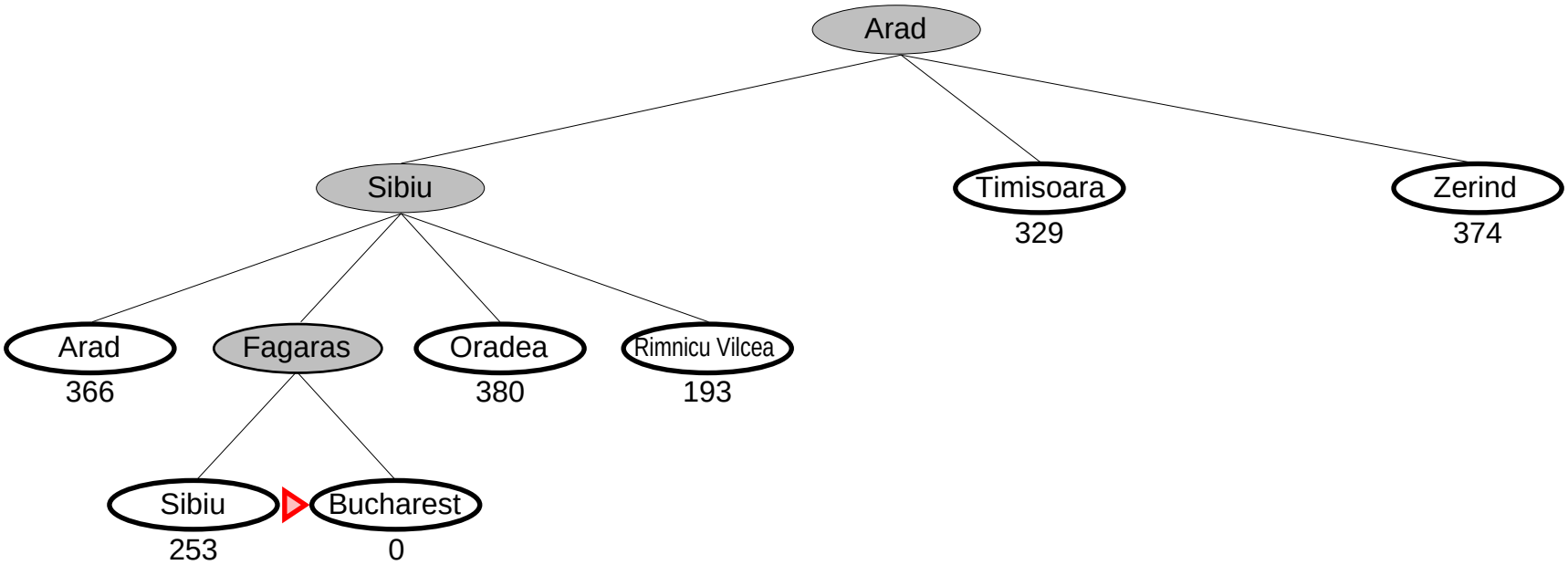
Beispiel: Greedy-Suche



Beispiel: Greedy-Suche



Beispiel: Greedy-Suche



Greedy-Suche: Eigenschaften

Vollständig:

- Nein, kann in Schleifen hängenbleiben, z.B.
  Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow \ldots$
- 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 $f(n) = g(n) + h(n)$

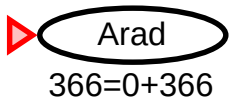
- $g(n)$ bereits aufgenommene Kosten um n zu erreichen
- $h(n)$ geschätzte Kosten von n zum Ziel
- $f(n)$ geschätzte Gesamtkosten des Pfades durch n zum Ziel

A*-Algorithmus benutzt *zulässige Heuristik*

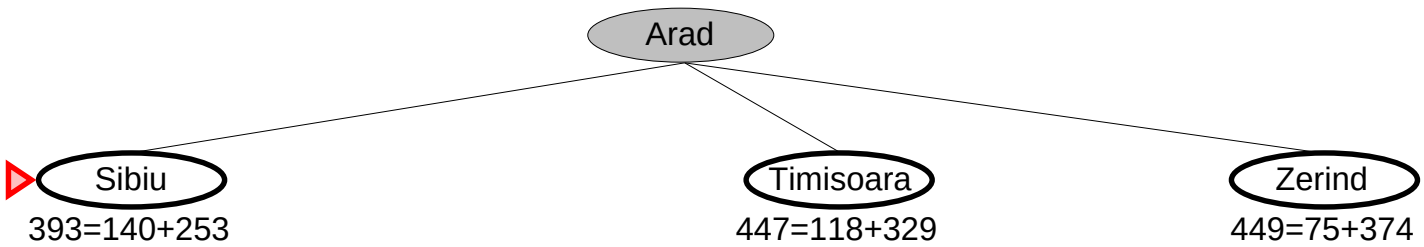
- Also, $h(n) \leq h^*(n)$ wobei $h^*(n)$ **wahren** Kosten von n
- Auch verlangt: $h(n) \geq 0$, also $h(G) = 0$ für beliebiges Ziel G
- Z.B. $h_{\mathrm{LL}}(n)$ überschätzt wirkliche Wegstrecke nie!

Satz: A*-Algorithmus ist optimal.

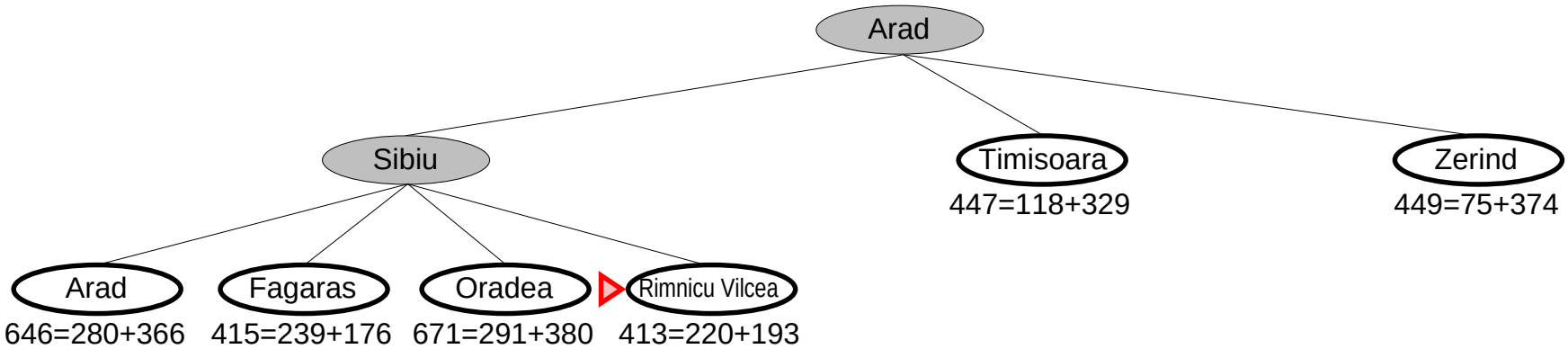
Beispiel: A*-Algorithmus



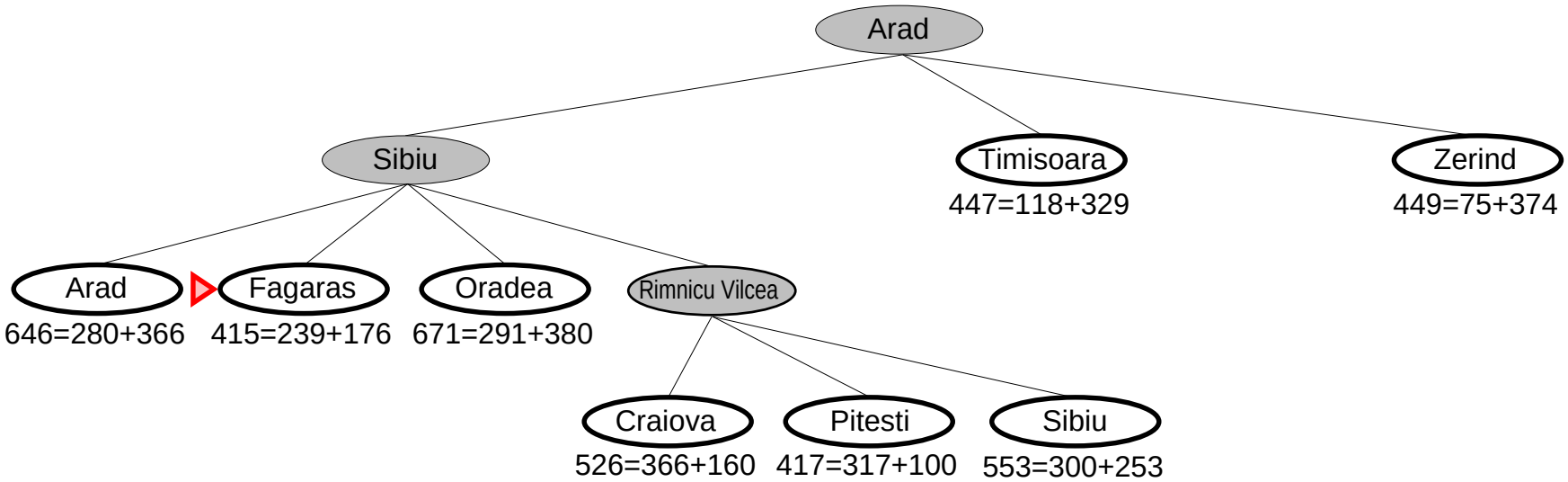
Beispiel: A*-Algorithmus



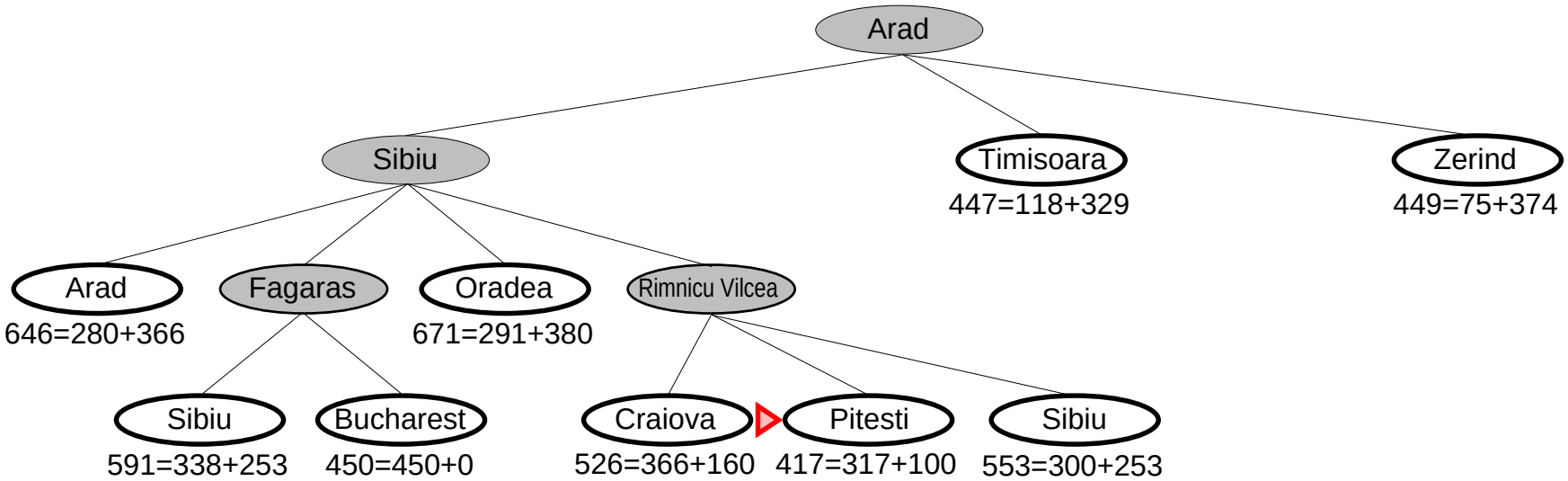
Beispiel: A*-Algorithmus



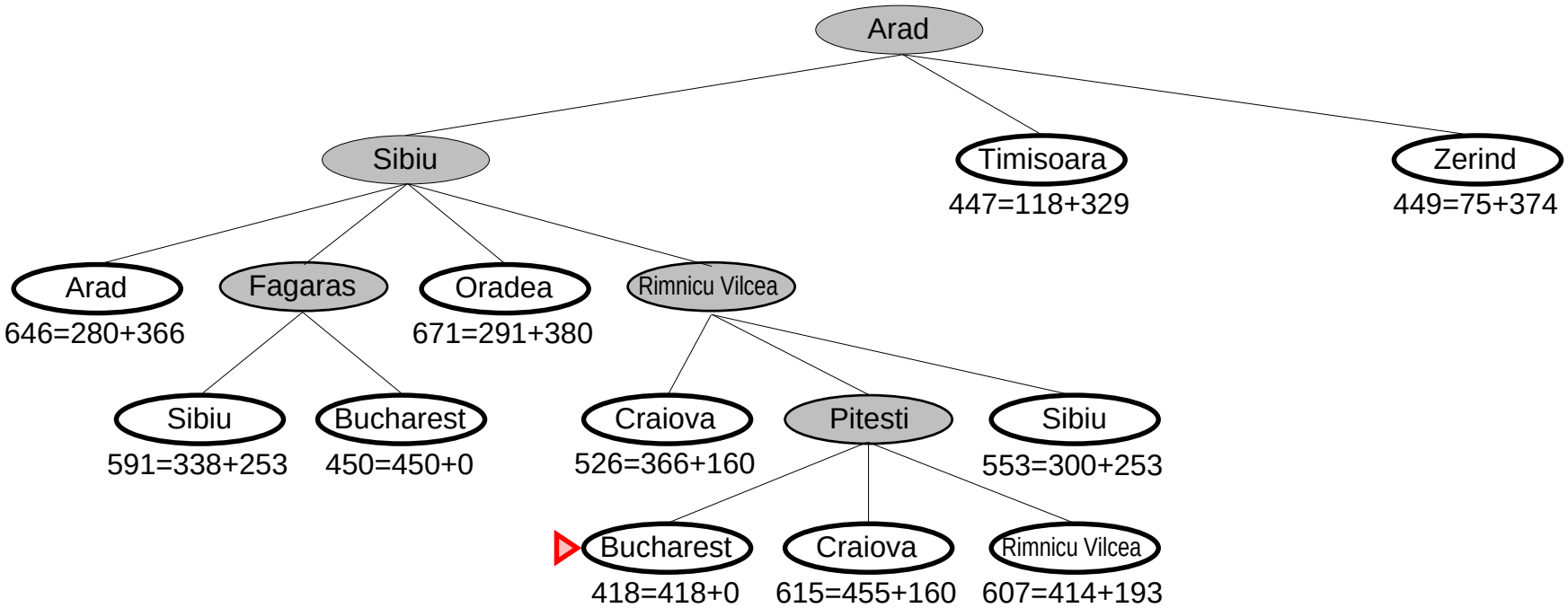
Beispiel: A*-Algorithmus



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 `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)

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 II

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 und somit 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 III

9. für alle $o \in O$:
   - $x := o(z)$ und $c := g(z) + \text{costs}(o)$
   - 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

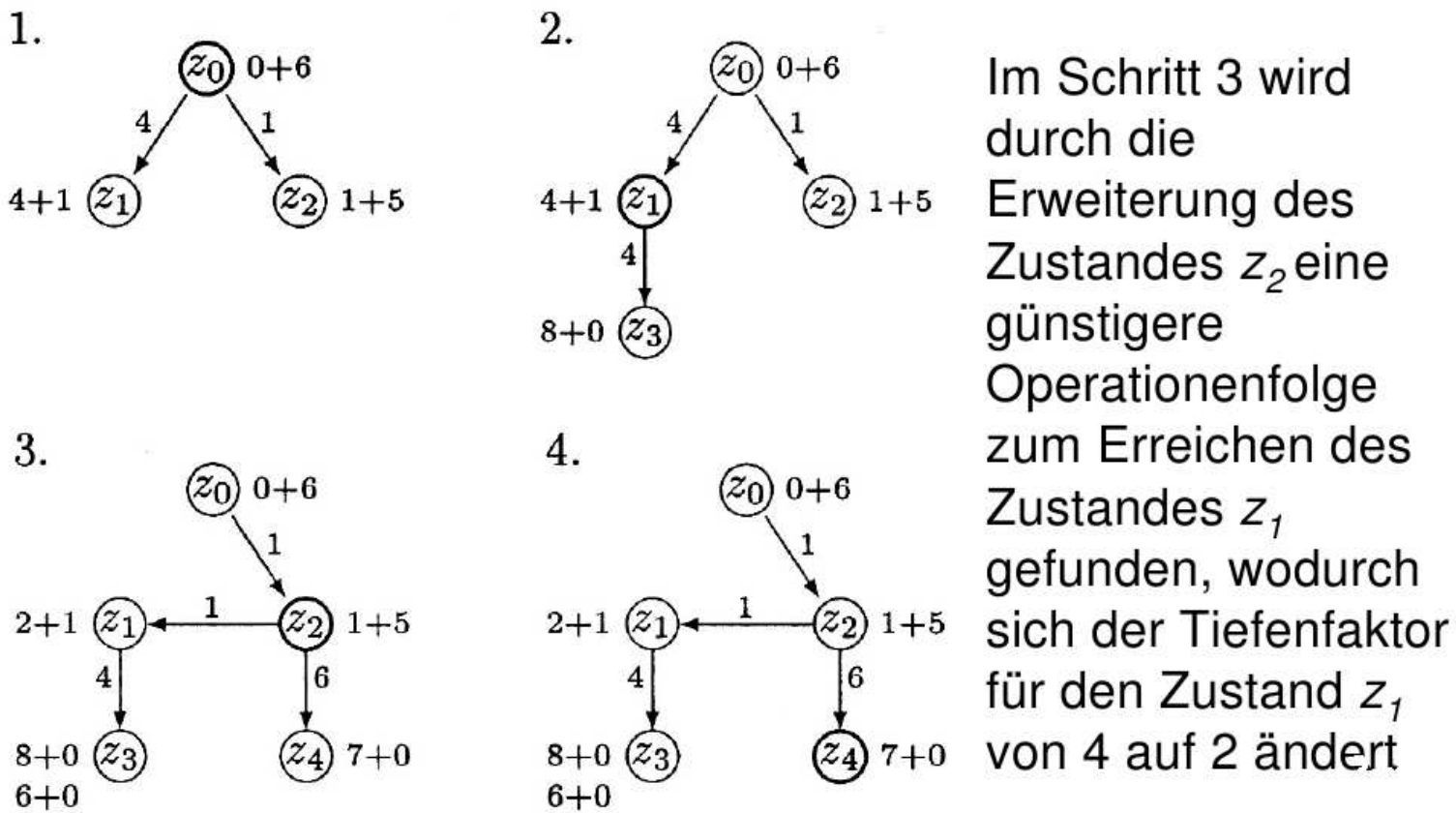
A*-Algorithmus: Ablauf IV

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 und somit bricht A* ab, andernfalls gehe zu Schritt 6

A*-Algorithmus: Anpassung des Tiefenfaktors

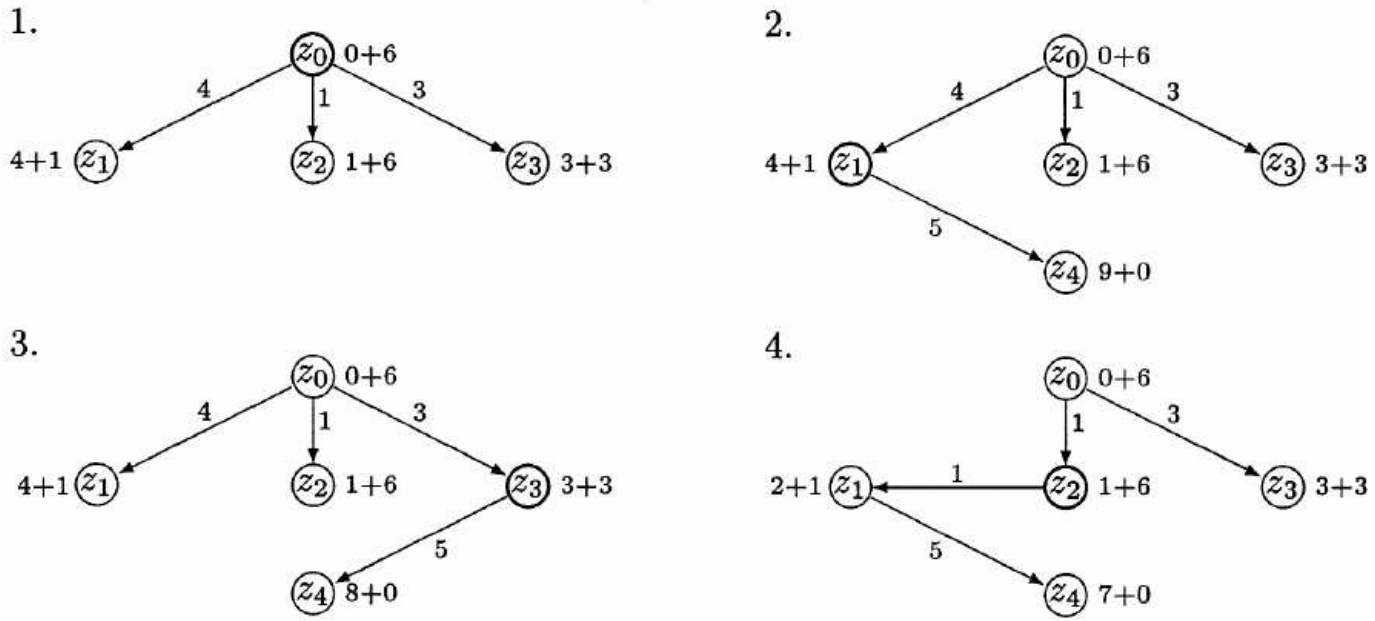
Notwendigkeit der Anpassung von Nachfolgezuständen (**9.**):



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

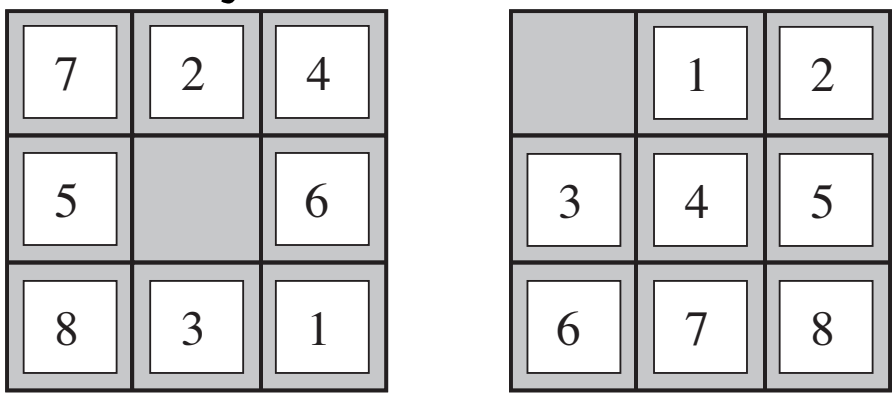
A* expandiert alle Knoten mit $f(n) < C^*$
A* expandiert einige Knoten mit $f(n) = C^*$
A* expandiert keine Knoten mit $f(n) > C^*$

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)
Somit ist h_2 besser als h_1
Typische Suchkosten:

Sei d Tiefe der Lösung mit geringsten Kosten
Für $d = 14$: iterative Tiefensuche ca. $3,5 \cdot 10^6$ Knoten
$A^*(h_1) = 539$ Knoten, $A^*(h_2) = 113$ Knoten
Für $d = 24$: iterative 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 (Relaxierung: Weglassen oder Lockern von Bedingungen in Optimierungsproblemen), somit Ausnutzung der 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

Zusammenfassung

Heuristikfunktionen schätzen Kosten des kürzesten Pfads

Gute Heuristiken können Suchkosten dramatisch reduzieren

Greedy-Suche expandiert Knoten mit kleinstem h

- Unvollständig und nicht immer optimal

A*-Algorithmus expandiert Knoten mit kleinstem $g + h$

- Vollständig und optimal
- Auch optimal effizient (bis auf Unentschieden, für Vorwärtssuche)

Erzeugung zulässiger Heuristiken durch exakte Lösungen relaxierter Probleme