Intelligente Systeme

Einführung

Prof. Dr. Rudolf Kruse Georg Ruß
Christian Moewes

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

Arbeitsgruppe Computational Intelligence
Institut für Wissens- und Sprachverarbeitung
Fakultät für Informatik
Otto-von-Guericke Universität Magdeburg

Heuristische Suche

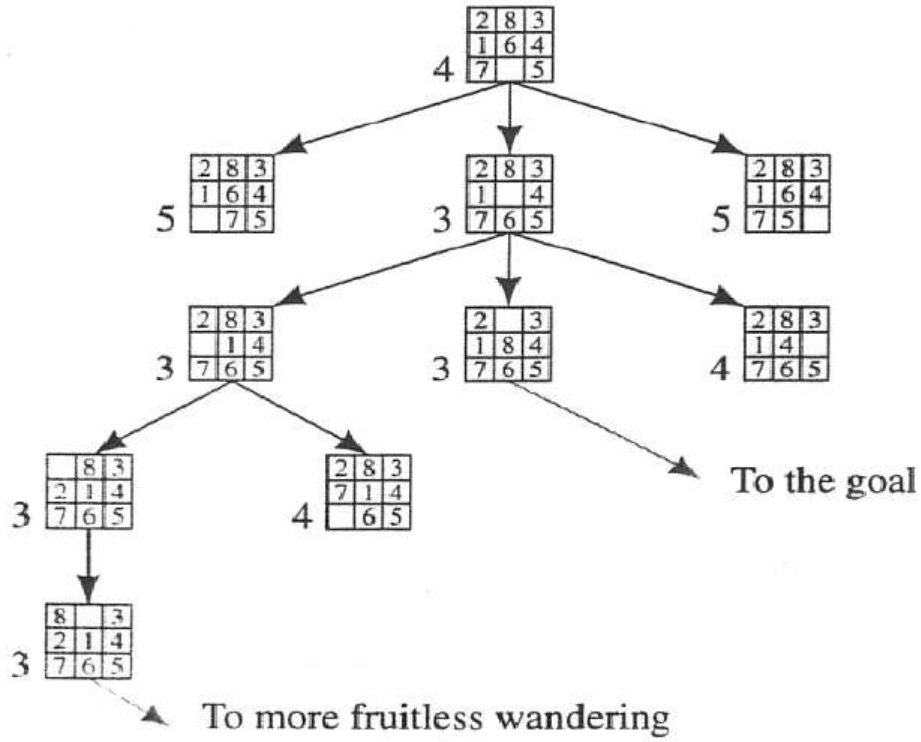
Heuristische Suche

Idee

- Wir nutzen eine (heuristische) Bewertungsfunktion $\hat{f}$, um zu entscheiden, welcher Knoten genauer untersucht werden sollte.
  - z.B. Bewertung von Stellungen beim Schach: niedrige Werte beschreiben "gute" Knoten
- Der Knoten mit dem kleinsten Wert wird untersucht. (Hier: Es werden alle Nachfolger dieses Knotens bestimmt.)
- Es wird abgebrochen, wenn ein Zielknoten erreicht ist.
- *Anschaulich:* Die Funktion $\hat{f}$ soll die Länge bzw. die Kosten eines Weges zum Ziel schätzen. Somit wird der vermutlich günstigste nächste Schritt gewählt.

Beispiel: 8-Puzzle

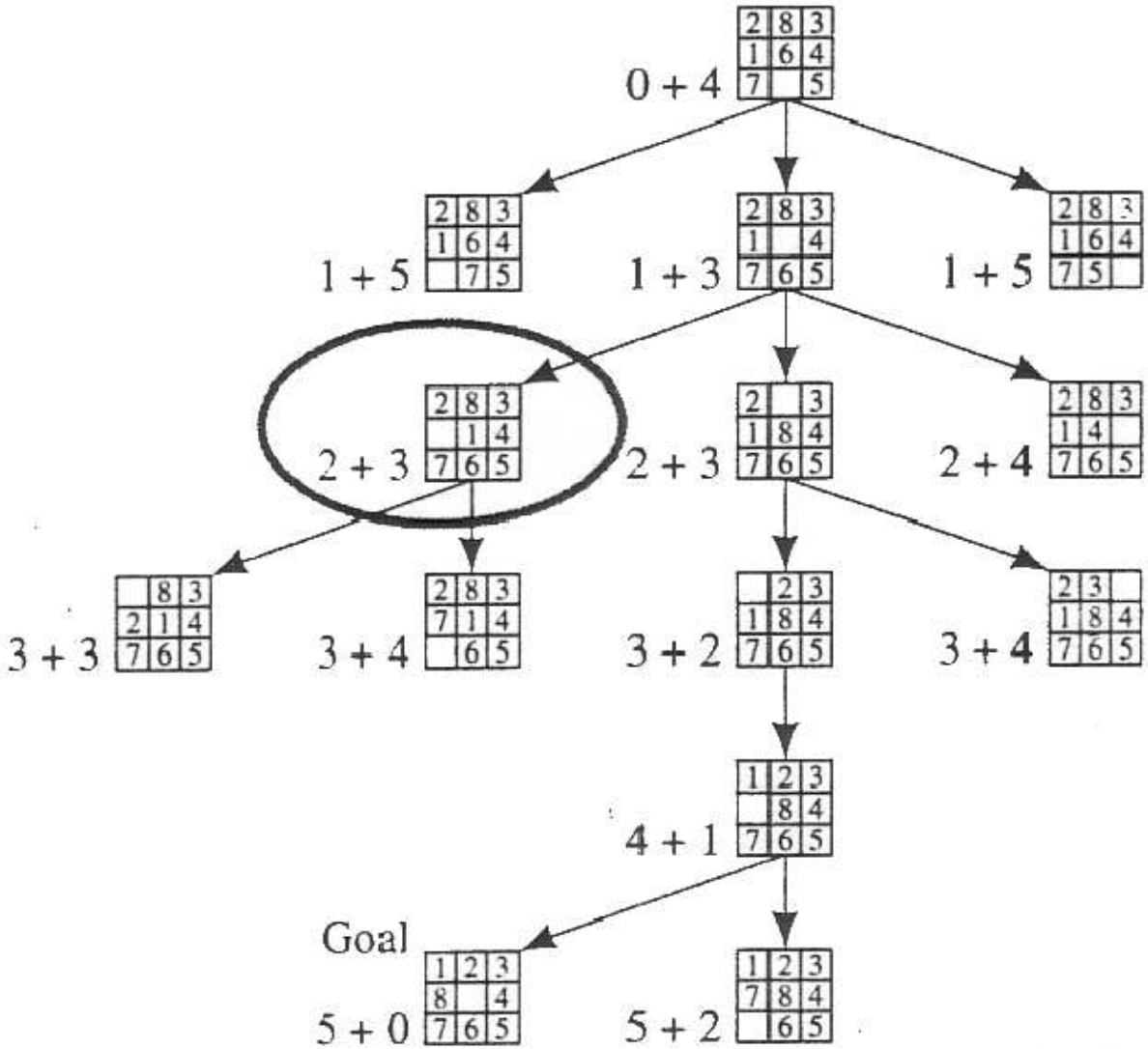
- $\hat{f}(n) =$ Anzahl der falsch plazierten Steine im Vergleich des Knotens mit dem Zielknoten
- allerdings ungünstig, da bei entsprechender Wahl des nächsten zu expandierenden Knotens nur der linke Pfad verfolgt wird



Beispiel: 8-Puzzle

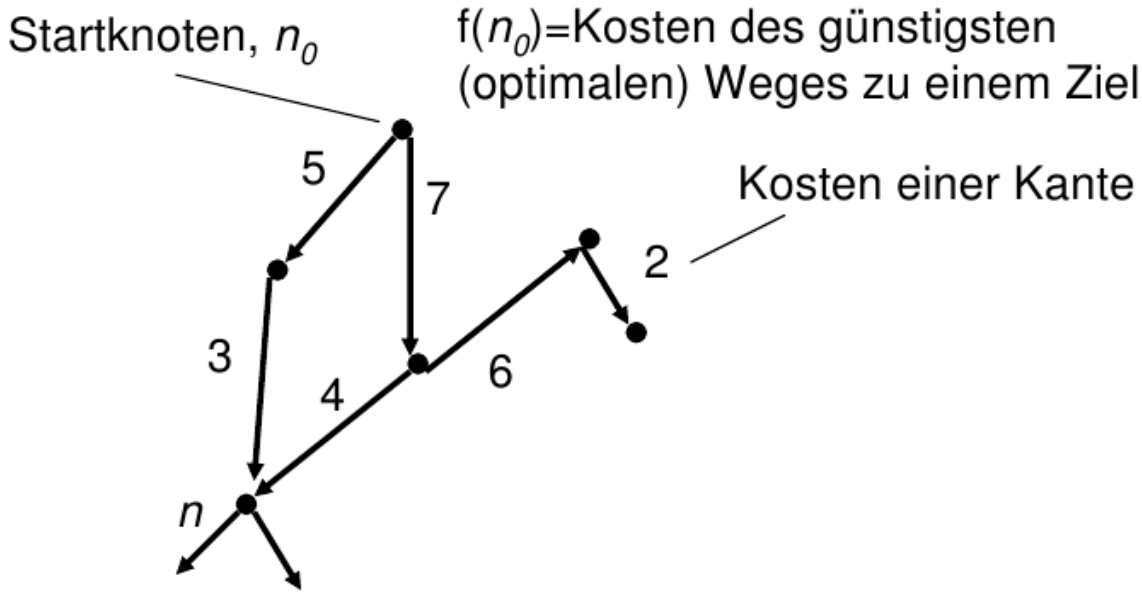
- $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$
- $\hat{g}(n)$: Tiefe von n, d.h. Länge des kürzesten Weges vom Start zu n
- $\hat{h}(n)$: Anzahl der falsch plazierten Steine (wie oben)
- Besser: beide Pfade werden verfolgt

Beispiel: 8-Puzzle



A*-Algorithmus

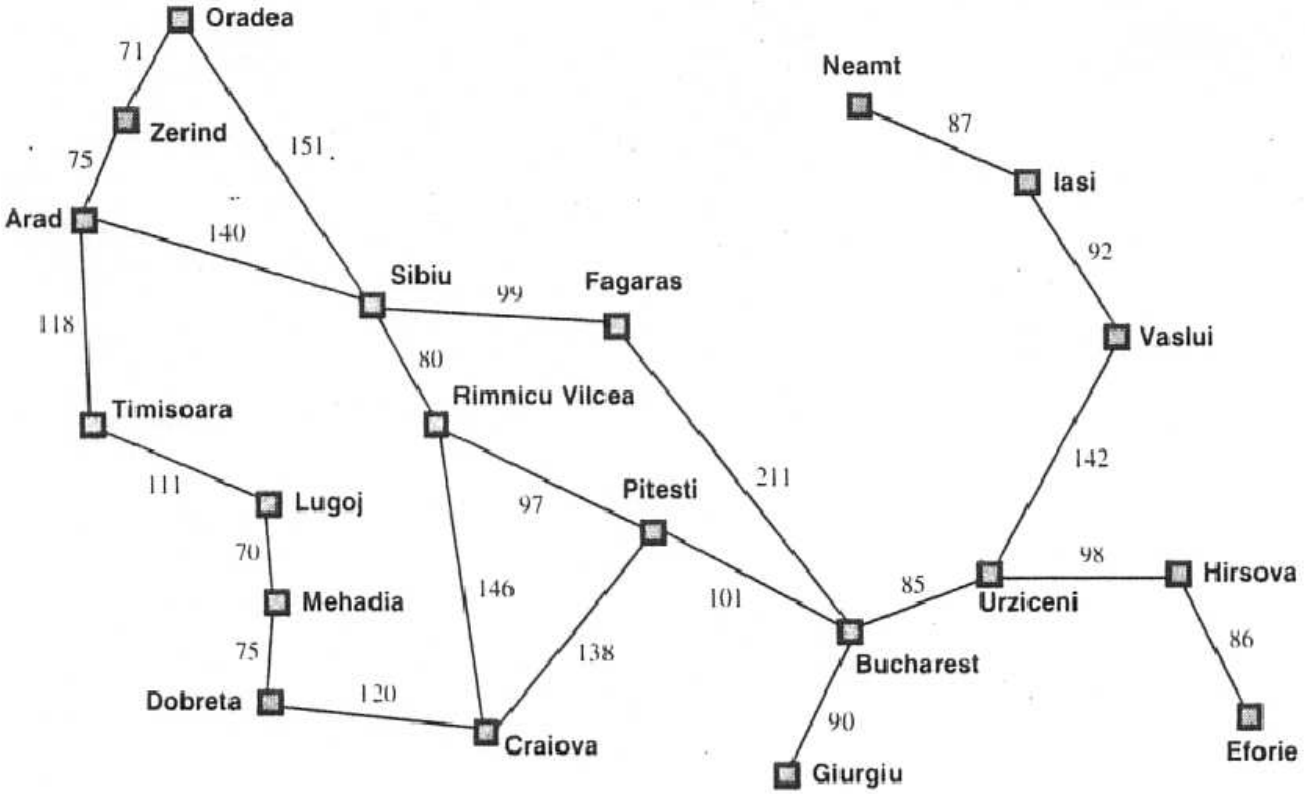
Notation



- $f(n) = g(n) + h(n)$: Kosten des günstigsten Weges zu einem Ziel, wenn man über Knoten n geht
- $g(n)$: Kosten des besten Weges von n_0 nach n (im vorherigen Beispiel: 8)
- $h(n)$: Kosten des besten Weges von n zu einem Ziel
- $\hat{f}, \hat{g}, \hat{h}$ sind Schätzwerte für f, g und h
- $\hat{f} = \hat{g} + \hat{h}$

A*-Algorithmus

Beispiel: Luftlinie



Straight–line distance to Bucharest	
Arad	366
Bucharest	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

A*-Algorithmus

- Ein Suchproblem ist definiert durch:
  - eine Menge Z von Zuständen,
  - eine (endliche) Menge O von Operatoren, die einen Zustand in einen anderen überführen,
  - einen Startzustand $z_0 \in Z$ und
  - eine Menge $T \subseteq Z$ von Endzuständen.
- Die Mengen Z und O legen den Suchraum in Form eines Zustandsgraphen fest:
  - Zustände bilden Knoten,
  - Operationen die Kanten.
  - Der Zustandsgraph ist i. A. nur implizit durch Startzustand und Operationenmenge gegeben.

A*-Algorithmus

- Gesucht ist: Eine Folge von Operationen, die den Startzustand in einen Zielzustand überführt und die minimale Kosten verursacht.
- Der A*-Algorithmus löst das Suchproblem unter Ausnutzung einer Heuristik:
  - Durch die Heuristik werden die Kosten einer Operationenfolge zum Erreichen eines Zielzustandes geschätzt.
  - Mit ihr wird gesteuert, wie der Zustandsgraph durchlaufen wird.

A*-Algorithmus

Gegeben (1):

- Der Startzustand z_0.
- Eine Menge $O = \{o_1, \ldots, o_n\}$ von Operationen, die zu einem gegebenen Zustand einen Nachfolgezustand liefern.
  - i.A. sind nicht alle Operationen auf alle Zustände anwendbar.
  - Eine Operation liefert den speziellen Wert $\perp$ (undefiniert) statt eines neuen Zustandes, wenn sie nicht anwendbar ist.

A*-Algorithmus

Gegeben (2):

- Eine reellwertige Funktion `costs`, die für jede Operation in O die ihr zugeordneten Kosten liefert.
  - u.U. hängen Kosten einer Operation davon ab, auf welchen Zustand sie angewandt wird. In diesem Fall kann die Funktion `costs` auch zweistellig sein.
- Eine reellwertige Heuristikfunktion $\hat{h}$, die für einen gegebenen Zustand eine Schätzung der Kosten einer Operationenfolge zum Erreichen eines Zielzustandes liefert.
- Eine Funktion `goal`, mit der sich feststellen lässt, ob ein gegebener Zustand ein Zielzustand ist.

A*-Algorithmus

1. Erzeuge einen (gerichteten) Graphen $G = \{V, E\}$, der nur den Startzustand z_0 enthält, d.h. setze $V := \{z_0\}$ und $E := \varnothing$.
   (In G werden der besuchte Teil des Suchraums und die besten bekannten Wege zum Erreichen eines Zustandes dargestellt.)
2. Erzeuge eine Menge `open`, die nur den Startzustand enthält, d.h. setze `open`$:= \{z_0\}$. (`open` enthält die erreichten Zustände, deren Nachfolger noch nicht erzeugt wurden.)
3. Erzeuge eine leere Menge `closed`. (Die Menge `closed` enthält die erreichten Zustände, deren Nachfolger bereits erzeugt wurden.)

A*-Algorithmus

4. Erzeuge eine Abbildung $\hat{g}$ von der Menge der Zustände in die reellen Zahlen, die für den Zustand z_0 den Wert 0 liefert und sonst undefiniert ist.
   (Die Abbildung ist der sogenannte Tiefenfaktor der Heuristik, der die Kosten der besten gefundenen Operationenfolgen zum Erreichen eines Zustandes vom Startzustand aus angibt.)
5. Erzeuge eine Abbildung e von der Menge der Zustände in die Menge der Operationen, die für alle Zustände undefiniert ist.
   (Die Abbildung e gibt später an, durch welche Operationen ein Zustand von seinem Vorgängerzustand aus erreicht wird. Mit ihr wird die Lösung des Problems aufgebaut.)

A*-Algorithmus

6. Wähle Zustand z aus der Menge open, mit $z \in \{x | \hat{f}(x) = \min_{y \in \text{open}} \hat{f}(y)\}, \text{wobei} \hat{f} = \hat{g} + \hat{h}$
   (Wähle also einen der Zustände, die gemäß Heuristik am erfolgversprechendsten sind.)
7. Falls goal(z) gilt, ist eine Lösung des Problems gefunden. Der Pfad wird aus G abgelesen.
8. Entferne den Zustand z aus der Menge open, d.h. open:=open$\backslash\{z\}$.
   (Die Nachfolger des Zustandes z werden im folgenden Schritt erzeugt.)

A*-Algorithmus

9. Für alle $o \in O$:
   - $x := o(z) \text{und} c := \hat{g}(z) + \text{costs}(o)$
   - Falls $x \neq \perp$:
     - Falls $x \notin \texttt{open} \cup \texttt{closed}$:
       - $\texttt{open} := \texttt{open} \cup \{x\}, e(x) := o, \hat{g}(x) = c$
       - Erweitere G: $V := V \cup \{x\}, E := E \cup \{(z, x)\}$
     - Falls $x \in \texttt{open} \cup \texttt{closed}$ und $c < \hat{g}(x)$
       - $e(x) := o, \hat{g}(x) = c$
       - Ersetze Vorgänger: $E := (E \backslash \{(a, b) | b = x\}) \cup \{(z, x)\}$
       - Falls $x \in \texttt{closed}$: Prüfe rekursiv alle Zustände, die sich von x erreichen lassen und ersetze Vorgänger ggf. durch günstigere Vorgänger.

A*-Algorithmus

10. Füge den Zustand z der Menge `closed` hinzu, d.h. setze $\texttt{closed} := \texttt{closed} \cup \{z\}$.
    (Die Nachfolger des Zustandes z wurden im vorhergehenden Schritt erzeugt.)
11. Falls die Menge `open` leer ist, so ist das Problem unlösbar. Der Algorithmus wird daher ohne Erfolg abgebrochen. Anderenfalls gehe zu Schritt 6.

A*-Algorithmus

Korrektheit

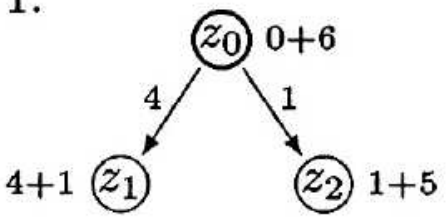
- Unter folgenden Bedingungen ist der A*-Algorithmus korrekt (d.h. liefert die Operationsfolge mit den geringst möglichen Kosten zum Erreichen eines Zielzustandes):
  - Jeder Zustand hat nur endlich viele Nachfolgezustände. (Diese Bedingung ist in der obigen Formulierung des A*-Algorithmus durch die Forderung einer endlichen Menge von Operationen ausgedrückt.)
  - Die Kosten aller Operationen sind größer als ein bestimmter positiver Wert ε.
  - Die Heuristikfunktion $\hat{h}$ überschätzt für keinen Zustand z die Kosten einer Operationenfolge zum Erreichen eines Zielzustandes (eine solche Funktion heißt auch optimistischer Schätzer).

A*-Algorithmus

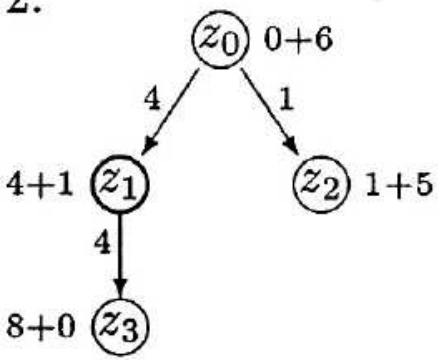
Bemerkungen

Notwendigkeit der Anpassung des Tiefenfaktors von Nachfolgezuständen (Schritt 9):

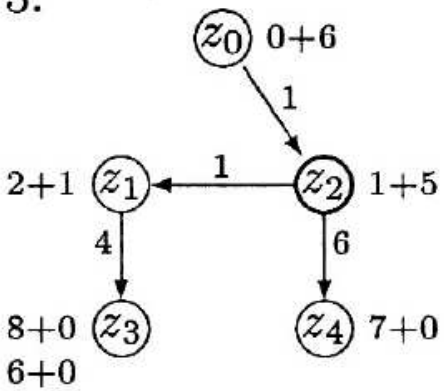
1.



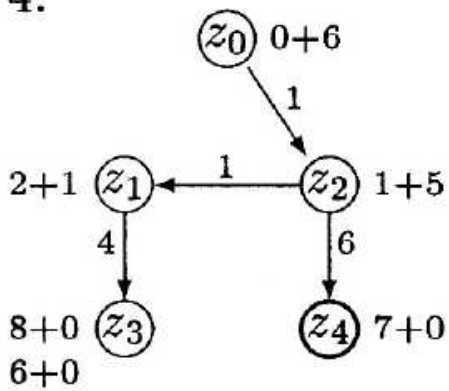
2.



3.



4.

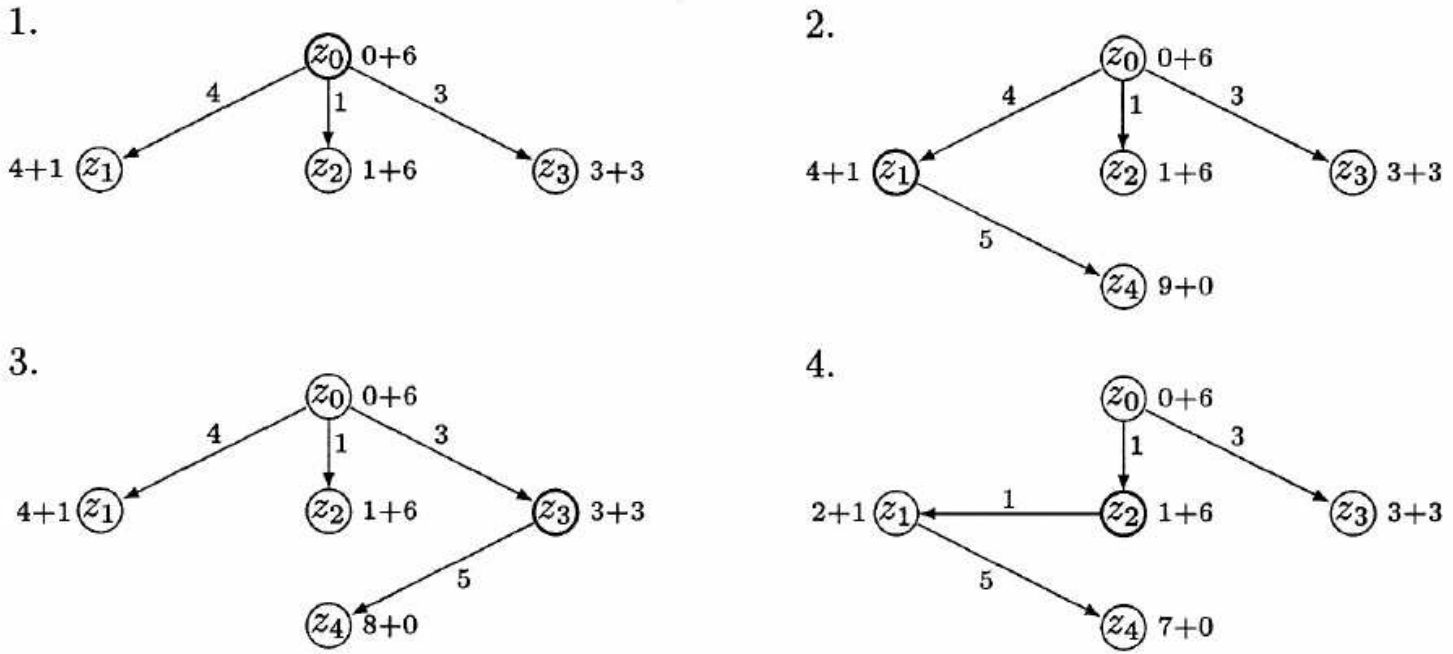


Im Schritt 3 wird durch die Erweiterung des Zustandes z_2 eine günstigere Operationenfolge zum Erreichen des Zustandes z_1 gefunden, wodurch sich der Tiefenfaktor für den Zustand z_1 von 4 auf 2 ändert

A*-Algorithmus

Bemerkungen

Notwendigkeit der (rekursiven) Anpassung des Tiefenfaktors *aller* Zustände (Schritt 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

Verbesserungen

- Es gibt eine ganze Reihe von Möglichkeiten, den A*-Algorithmus zu verbessern, z.B.:
  - Findet man eine Heuristik $\hat{h}_1$ mit $\hat{h}(n) \leq \hat{h}_1(n)$ für alle Nicht-Zielknoten n, so heißt der zugehörige Algorithmus A*$_1$ *more informed* als A*. Der Algorithmus A*$_1$ ist effizienter.
  - Verfeinerungen des A*-Algorithmus sind der Iterative-Deepening-A* und die Recursive Best-First Search

A*-Algorithmus

Beispiele

- Beim 8-Puzzle liefert $\hat{h}(n) = 0$ die (ineffiziente) uniform cost search.
- Für $\hat{h}(n) = w(n) =$ *Anzahl der Steine am falschen Platz* ist die Bedingung des Satzes erfüllt, jedoch ist die Funktion nicht gut geeignet.
- Eine bessere Schätzung ist $\hat{h}(n) = P(n)$, wobei $P(n)$ die Summe der Abstände ist, die jeder Stein von seinem Zielort (ohne störende Steine) hat. (Manhattan- / City-Block-Abstand)
- Beim Routenplaner ist der direkte Abstand $\hat{h}(n) =$ Luftlinie vom Ziel eine gute Schätzung.

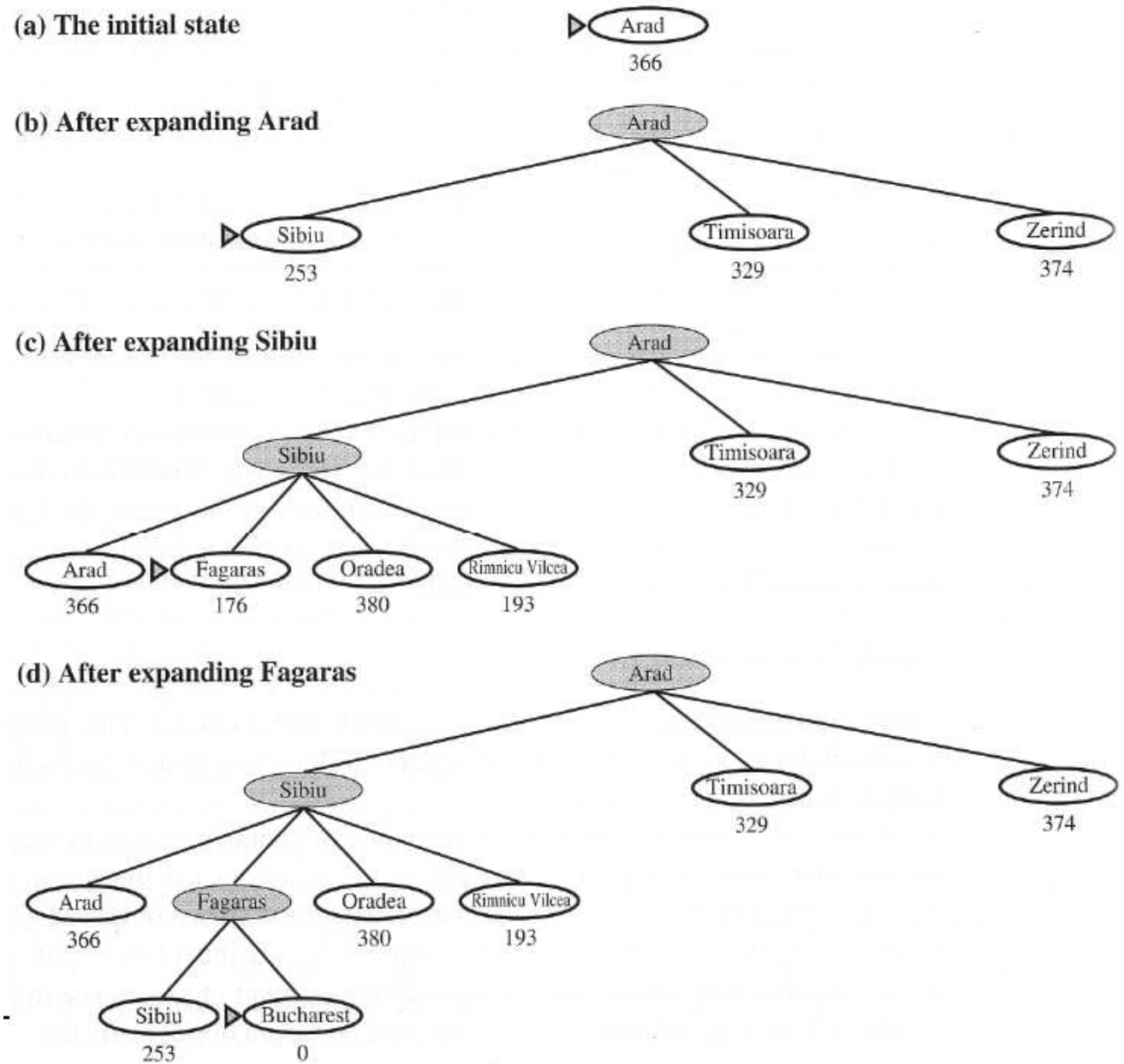
A*-Algorithmus

Effizienz

Die Effizienz von A* hängt ab von

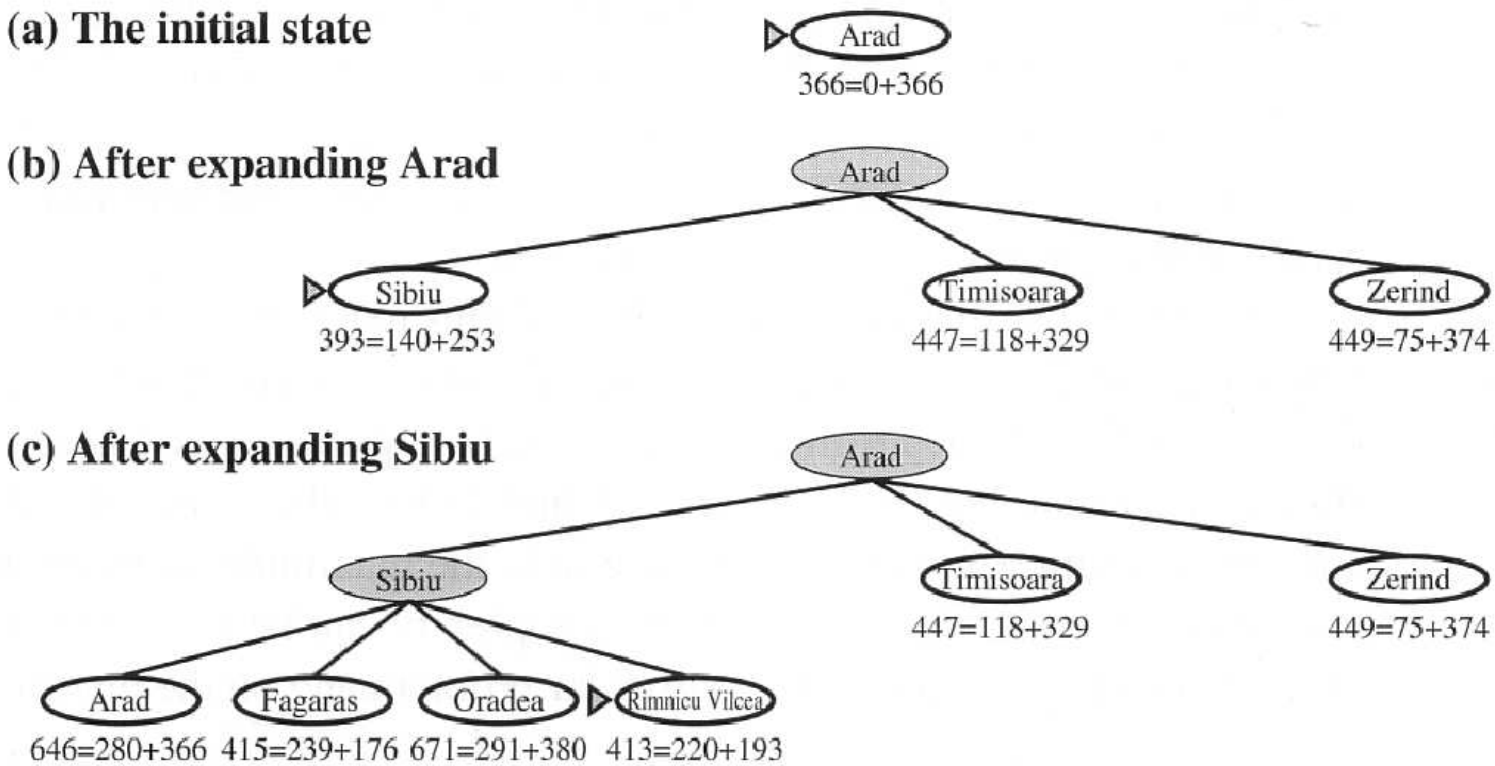
- den Kosten des gefundenen Weges,
- der Anzahl der Knoten, die untersucht wurden, um den Weg zu finden und
- der Komplexität der Berechnung von $\hat{h}$.

A*-Algorithmus

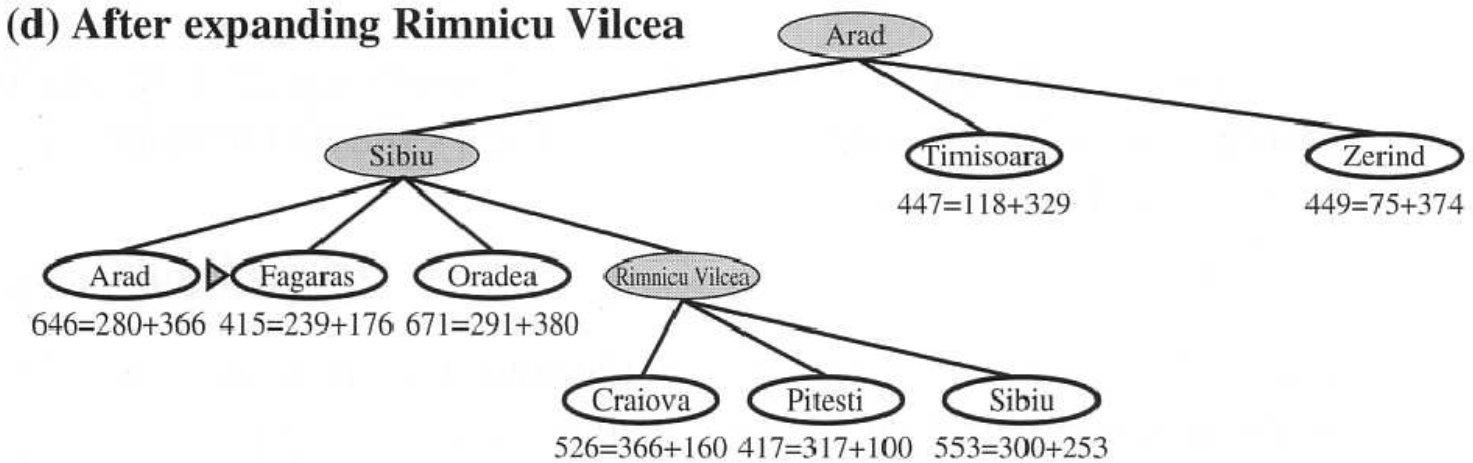


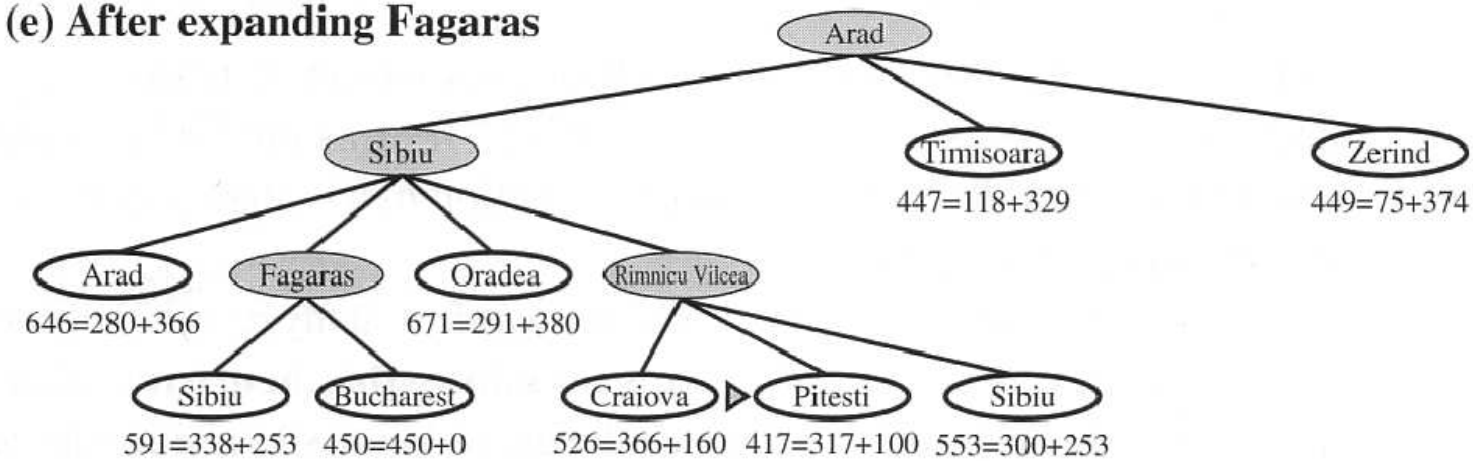
A*-Algorithmus

Knoten sind mit $\hat{f} = \hat{g} + \hat{h}$ beschriftet

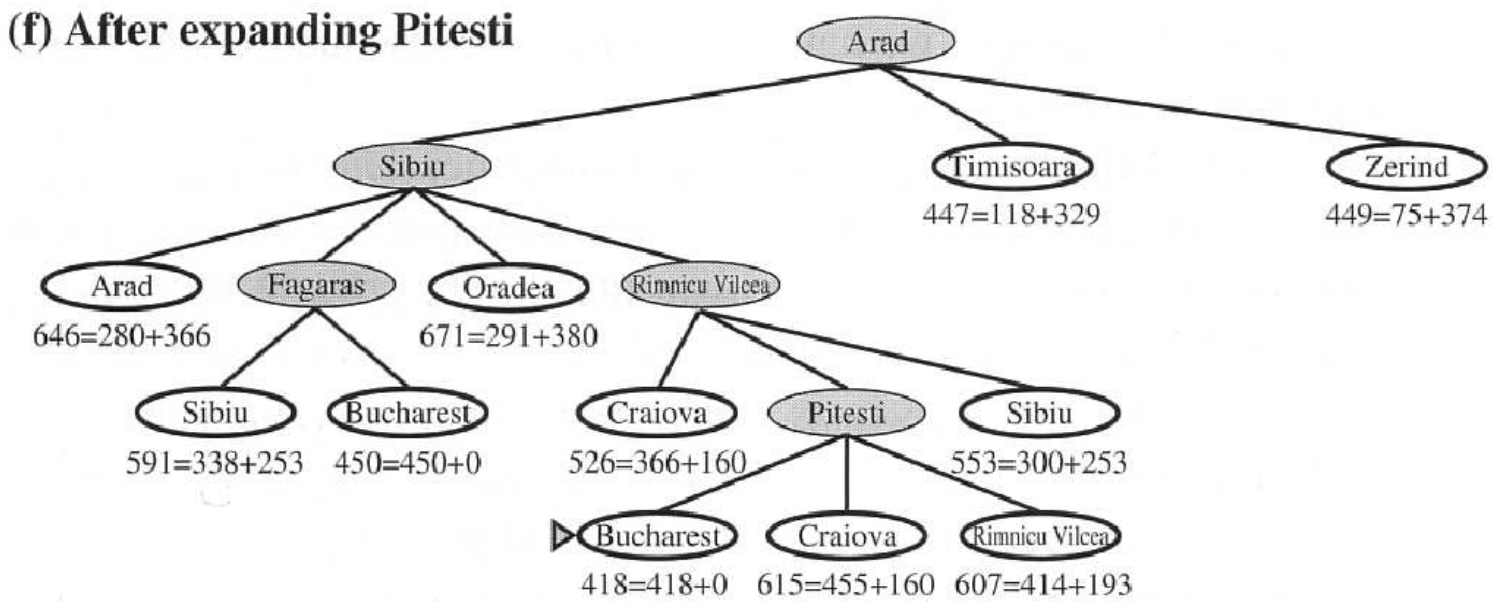


A*-Algorithmus





A*-Algorithmus



A*-Algorithmus

In realen Anwendungen wird der A*-Algorithmus oft bei Planungs-Agenten eingesetzt (z. B. beim Robocup). Eine verwendete Architektur sieht wie folgt aus: