Informierte Suchverfahren

- Für größere Suchbäume sind Breiten- und Tiefensuche nicht effizient genug.
- Vielversprechender sind Ansätze, bei denen Problemwissen zur Steuerung des Suchprozesses eingesetzt wird.
- Dies können wir erreichen, indem wir die Zustände (Knoten) danach bewerten, wie erfolgversprechend sie sind.
- Wir schätzen beispielsweise für jeden Knoten, wie nahe er an einem Zielknoten liegt.
- Solch eine Bewertung heißt *heuristische Funktion*.

Heuristische Funktion

Definition 2.3. Eine Funktion, die jedem Zustand (Knoten) s eines Zustandsraums (Suchbaums) eine nichtnegative Zahl $h(s)$ zuordnet, heißt *heuristische Funktion*. Für einen Zielzustand s gilt dabei $h(s) = 0$.

Ein Suchverfahren, das eine heuristische Funktion zur Auswahl der zu expandierenden Zustände einsetzt, heißt *informiertes Suchverfahren* oder auch *heuristisches Suchverfahren*.

Beispiel 2.7. [Schiebepuzzle]

Startzustand

5	4	
6	1	8
7	3	2

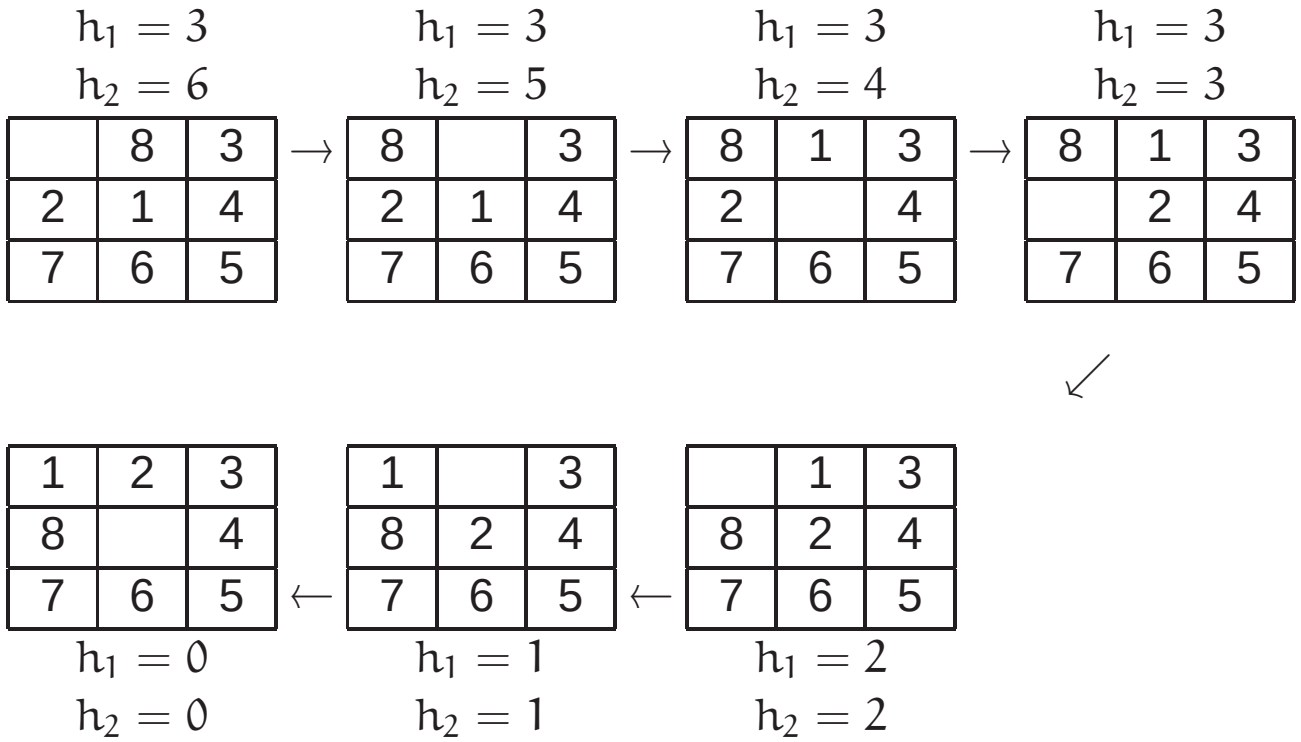
$\rightarrow$

1	2	3
8		4
7	6	5

Endzustand

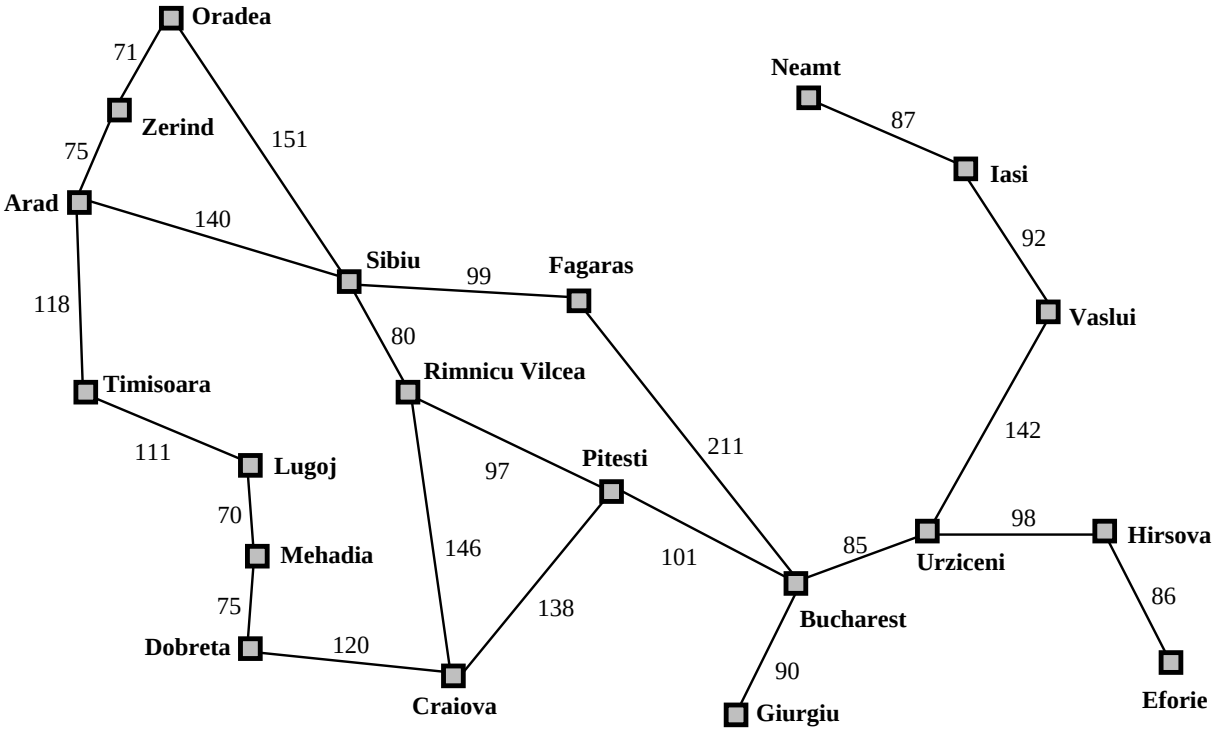
Mögliche heuristische Funktionen:

- $h_1(s) :=$ Anzahl der Plättchen, die nicht an der richtigen Stelle liegen.
  Hier: $h_1(s) = 7$.
- $h_2(s) :=$ Summe der Entfernungen aller Plättchen von der Zielposition.
  Hier: $h_2(s) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$.


$h_1 = 3$ $h_2 = 6$
$h_1 = 3$ $h_2 = 5$
$h_1 = 3$ $h_2 = 4$
$h_1 = 3$ $h_2 = 3$
$h_1 = 2$ $h_2 = 2$
$h_1 = 1$ $h_2 = 1$
$h_1 = 0$ $h_2 = 0$

- Die heuristische Funktion h_2 differenziert stärker als h_1, d.h.
- h_2 kann Zustände unterscheiden, die von h_1 gleich bewertet werden.
- Eine heuristische Funktion ist um so brauchbarer, je mehr Zustände sie unterschiedlich bewertet.
- Eine heuristische Funktion, die alle Zustände gleich bewertet, ist unbrauchbar.

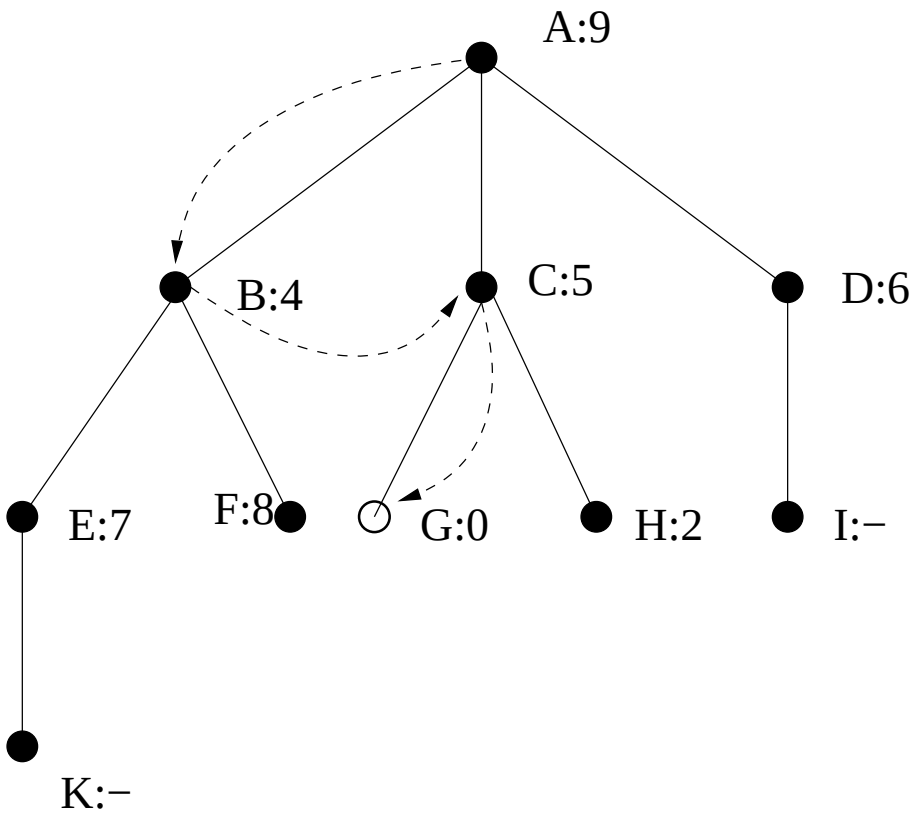
Beispiel 2.8. [Routenplanung] Zur Abschätzung der Straßenentfernung wird die Luftlinienentfernung benutzt.



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Bestensuche

- Bei der *Bestensuche* erfolgt die Expansion eines Knotens auf Basis der heuristischen Funktion.
- Hierzu werden in der Agenda die Knoten zusammen mit ihrer Bewertung abgelegt.
- Es wird nun jeweils der Knoten der Agenda expandiert, der die geringste Bewertung aufweist.
- Die Agenda hat also die Form einer Prioritätswarteschlange (priority queue).
- Ansonsten ist die Bestensuche analog zur Tiefen- und Breitensuche.



Schritt	Agenda	s_{akt}
1	A:9	A
2	B:4, C:5, D:6	B
3	C:5, D:6, E:7, F:8	C
4	G:0, H:2, D:6, E:7, F:8	G

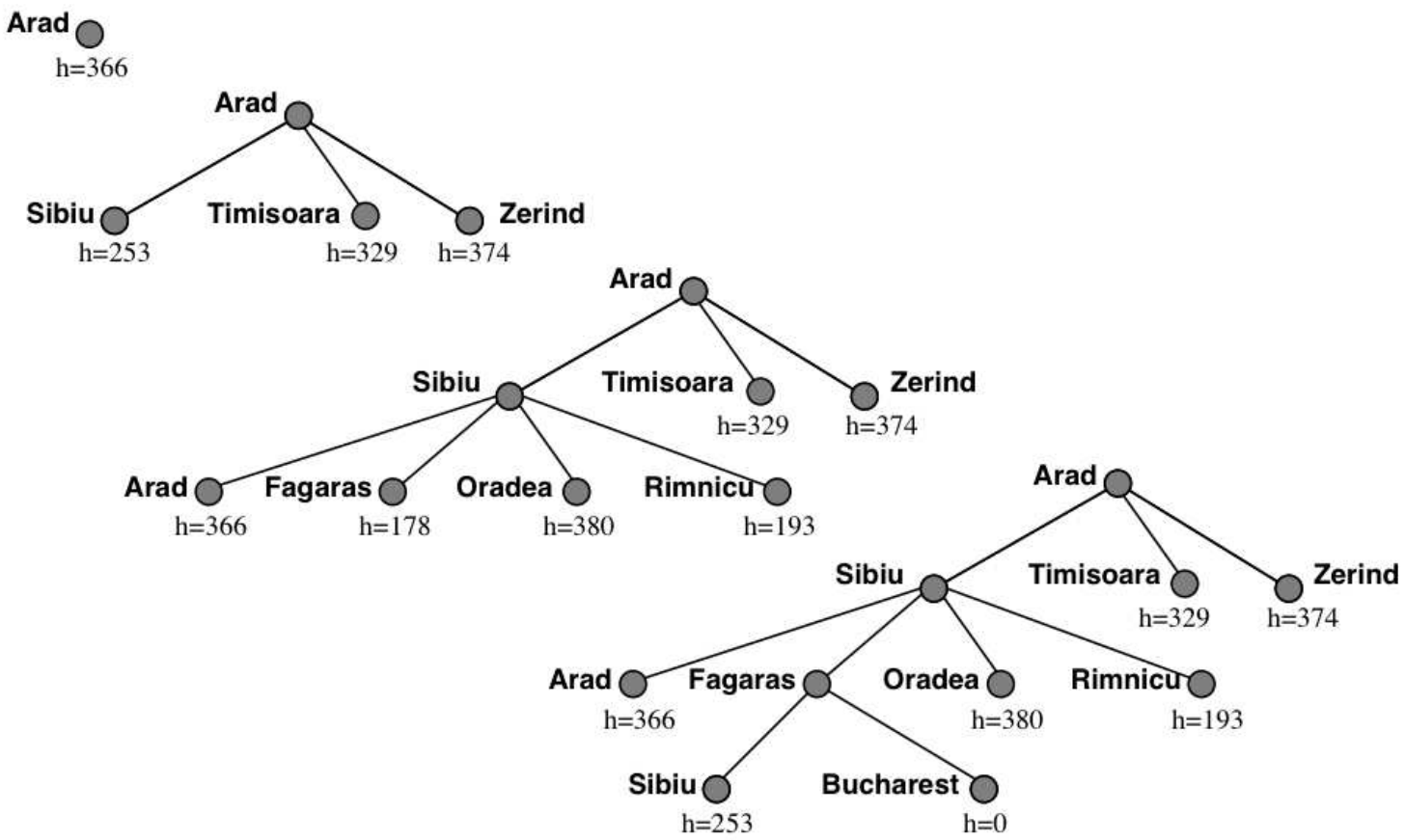
Algorithmus zur Bestensuche

Algorithmus 2.3. [Bestensuche]

$Agenda := (Startknoten)$;
`while` $Agenda \neq ()$ `do`
    $s_{akt} := first(Agenda)$;
    Entferne s_{akt} aus der Agenda;
    `if` s_{akt} ist Zielknoten `then` s_{akt} ist Lösung; STOP;
    $Agenda := einfuegen(Agenda, Nachfolger(s_{akt}))$;
`end`
Problem hat keine Lösung; STOP;

Beispiel 2.9. Suchbaum für Beispiel 2.7 mit Bestensuche. Tafel ✎

Beispiel 2.10. Bestensuche für die Suche einer Route von Arad nach Bucharest.


Arad
h=366
Arad
Sibiu
Timisoara
Zerind
h=253
h=329
h=374
Arad
Sibiu
Timisoara
Zerind
h=329
h=374
Arad
Fagaras
Oradea
Rimnicu
h=366
h=178
h=380
h=193
Arad
Sibiu
Timisoara
Zerind
h=329
h=374
Arad
Fagaras
Oradea
Rimnicu
h=366
h=380
h=193
Sibiu
Bucharest
h=253
h=0

Eigenschaften der Bestensuche

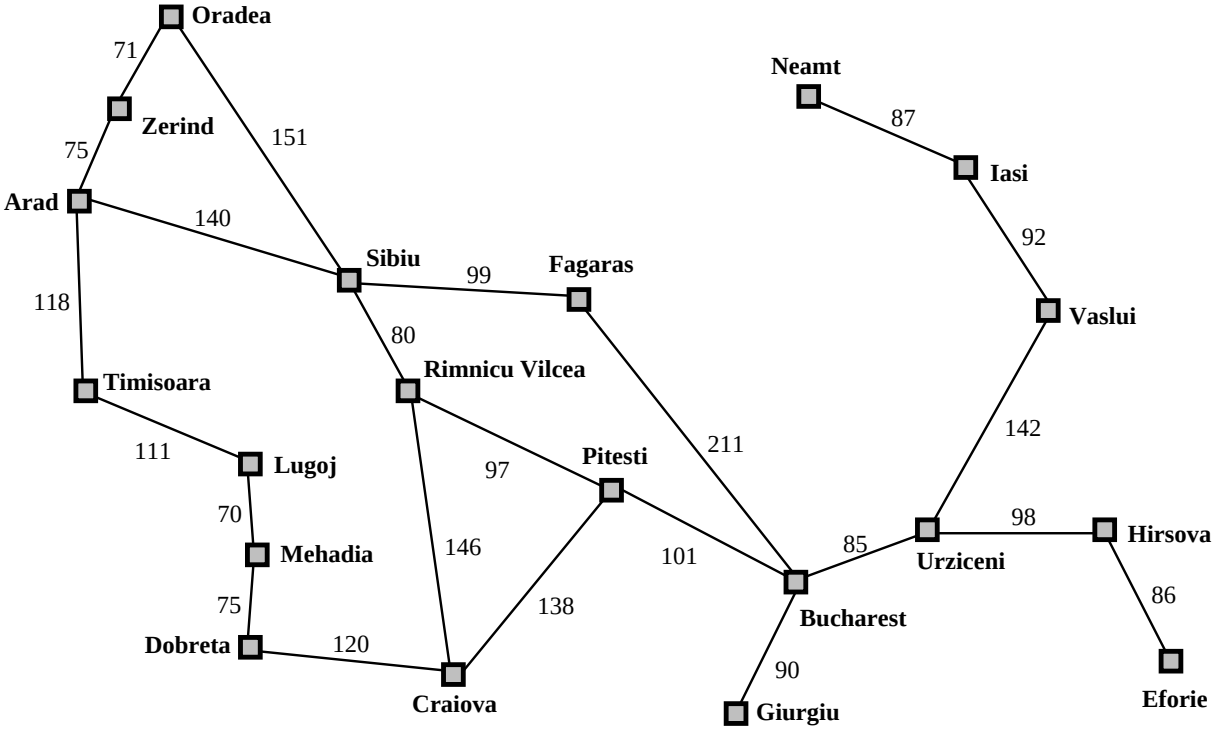
Definition 2.4. Eine heuristische Funktion h heißt *fair* gdw. es zu jedem $n \geq 0$ nur endlich viele Knoten s gibt mit $h(s) \leq n$.

- Fairness entspricht der Vollständigkeit bei uninformierten Suchverfahren.
- Ist eine heuristische Funktion fair, so wird ein Zielknoten gefunden, falls ein solcher existiert.

Bestensuche und Optimalität

- Die Bestensuche vernachlässigt die “Kosten” bei der Anwendnung der Operatoren.
- Wird die Güte einer Lösung charakterisiert durch diese Operatorkosten, so findet die Bestensuche allgemein keine optimale Lösung.

Beispiel 2.11. Suche einer Route von Arad nach Bucharest:



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Bestensuche wählt Route über Sibiu und Fagaras, obwohl die Route über Rimnicu Vilcea und Pitesti kürzer ist.

Bewertung von Lösungen

Definition 2.5. Es sei $p = (s_0, s_1, \ldots, s_r)$ eine Folge von Zuständen und s_{i+1} sei durch Anwendung eines Zustandsübergangsoperators auf s_i erreichbar.

Beim Übergang von s_i nach s_{i+1} fallen Kosten in Höhe von $k(s_i, s_{i+1})$ an.

Die *Kosten $k(p)$ der Zustandsfolge* seien definiert durch:

$$k(p) := \sum_{i=0}^{r-1} k(s_i, s_{i+1})$$

Für einen Zustand s sei:

$g^*(s)$:= minimale Kosten für einen Weg vom Startzustand s_0 nach s

$h^*(s)$:= minimale Kosten für einen Weg von s zu einem Zielzustand

Problem: Finde (falls möglich) eine Zustandsfolge p^* vom Startzustand s_0 in einen Zielzustand z, die minimale Kosten aufweist, d.h.

$k(p^*) = h^*(s_0)$ bzw.

$k(p^*) = \min\{g^*(z) | z \text{ ist Zielzustand}\}$.

Zulässiger Schätzer

Definition 2.6. Eine heuristische Funktion h heißt *zulässiger Schätzer* bzw. *zulässig* gdw. $\mathrm{h}(s) \leq \mathrm{h}^*(s)$ für alle Zustände s des Zustandsraums.

Bemerkung: Eine zulässiger Schätzer überschätzt nie die anfallenden Restkosten!

Beispiel 2.12. Zulässige Schätzer sind:

- die heuristischen Funktionen aus Beispiel 2.3 für das Schiebepuzzle und
- die Luftlinienentfernung beim Routenproblem.
- Bei kombinatorischen Optimierungsproblemen werden als zulässige Schätzer häufig effizient lösbare Relaxationen des Problems verwendet. Beispiel: minimaler Spannbaum als Relaxation für die Berechnung eines minimalen Hamiltonschen Weges.
- Allgemein: Man vernachlässigt Nebenbedingungen, so daß man zu einem einfach lösbaren Problem kommt.

Der A*-Algorithmus

Der A*-Algorithmus basiert auf:

1. einer Bewertung $g(s)$ für die Zustände, wobei $g(s)$ die bisher geringsten Kosten zur Erreichung des Zustands s angibt,
2. einer (üblicherweise zulässigen) heuristischen Funktion $h(s)$ zur Schätzung der Restkosten und
3. einer Bewertungsfunktion $\Phi(s) = g(s) + h(s)$, die zur Auswahl des zu expandierenden Zustandes dient.

Steuerung der Suche bei A*:

☞ Es wird der Knoten der Agenda expandiert, der die geringste Bewertung $\Phi(s)$ aufweist.

Folgende Punkte sind beim A*-Algorithmus zu berücksichtigen:

- Durch eine Verringerung von $g(s)$ für einen Zustand s kann auch eine Verringerung von $\Phi(s)$ auftreten.
- Dies kann im allgemeinen auch für schon expandierte Knoten der Fall sein!
- Deshalb werden schon expandierte Knoten in einer speziellen Liste *Closed* verwaltet.
- Bewertungen sind dementsprechend anzupassen.

Algorithmus 2.4. [A*]

$Agenda := (Startknoten)$;
$g(Startknoten) := 0$;
$p(Startknoten) = \texttt{nil}$;
`while` $Agenda \neq ()$ `do`
    $s_{akt} := first(Agenda)$;
    Entferne s_{akt} aus der Agenda;
    Füge s_{akt} in *Closed* ein;
    `if` s_{akt} ist Zielknoten `then` s_{akt} ist Lösung; STOP;
    `forall` $s \in Nachfolger(s_{akt})$ `do`
        `if` $s \notin Agenda \wedge s \notin Closed$ `then`
            $g(s) := g(s_{akt}) + k(s_{akt}, s)$;
            $p(s) := s_{akt}$;
            Füge s in die Agenda mit Bewertung $\Phi(s)$ ein;
        `else`
            `if` $g(s_{akt}) + k(s_{akt}, s) < g(s)$ `then`

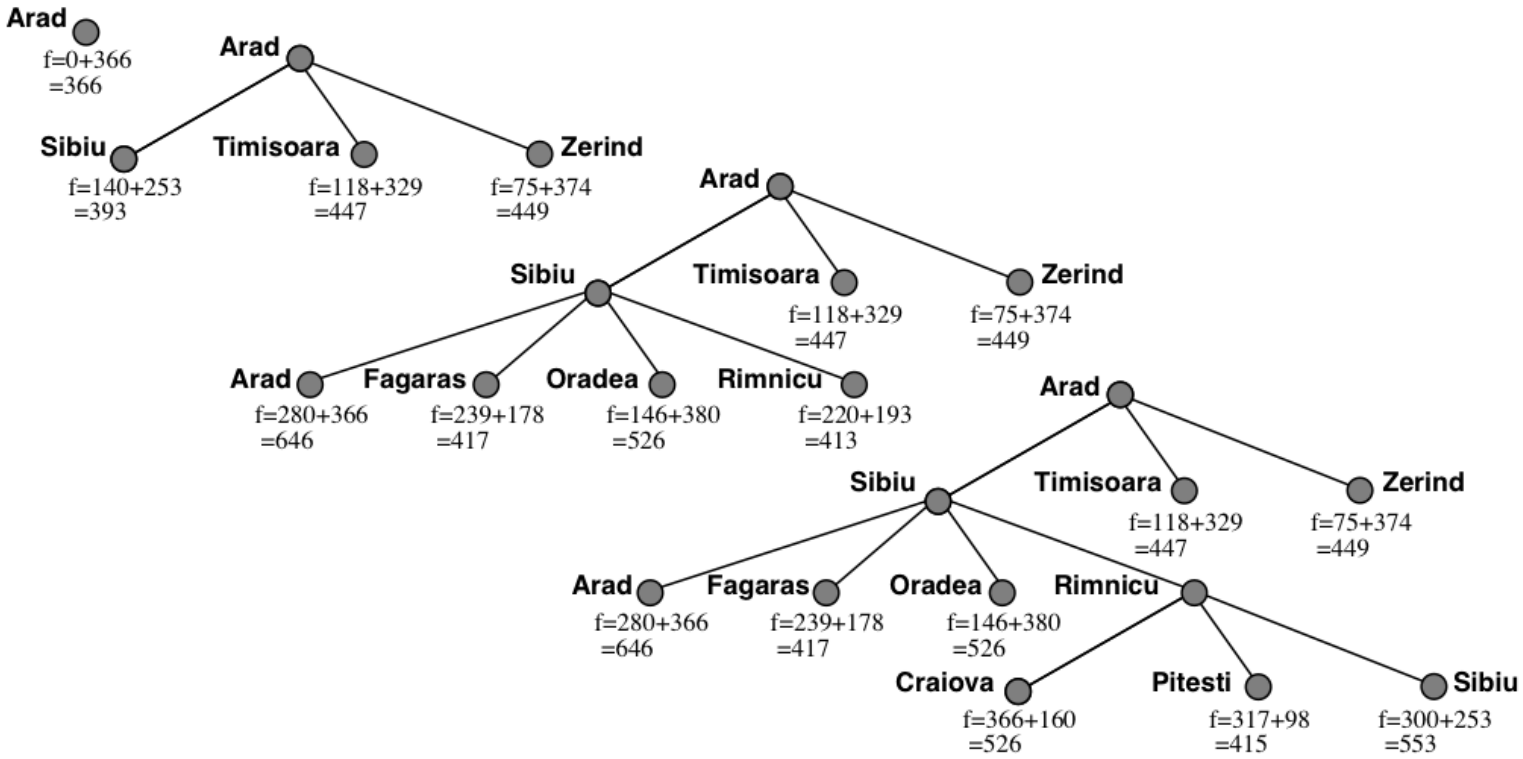
```
                g(s) := g(s_akt) + k(s_akt, s);
                p(s) := s_akt;
                if s ∈ Closed then
                    Entferne s aus Closed;
                    Füge s in die Agenda ein;
                endif
            endif
        endif
    end
end
```

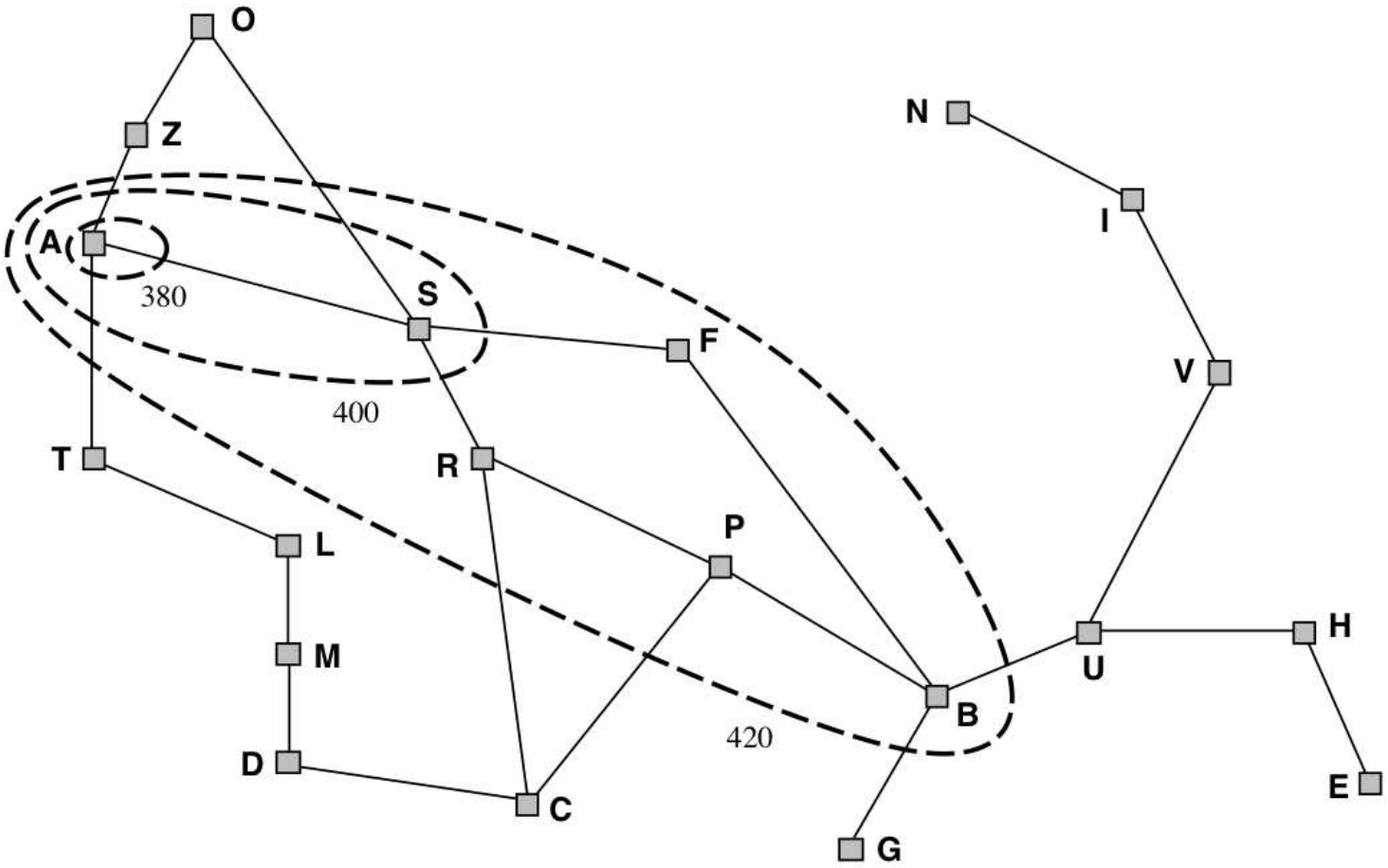
Problem hat keine Lösung; STOP;

- Für einen Knoten s gibt $\mathrm{p}(s)$ den Vorgängerknoten auf dem bisher besten Weg an.
- Den bisher besten Weg zu einem Knoten s erhält man also, in dem man von s sukzessive den Verweisen $\mathrm{p}(.)$ folgt.
- Alternativ kann man an jedem Knoten den kompletten bisher optimalen Pfad speichern.
- Der notwendige Speicherplatzverbrauch für die Pfade ist dann aber quadratisch in der Länge des Suchpfades.

A*-Anwendungsbeispiele

Beispiel 2.13. Beim Routenproblem berechnet der A*-Algorithmus die kürzeste Route.





☞ Durch die Schätzfunktion findet die Suche “zielgerichtet” statt.

Beispiel 2.14. Man benutze den A*-Algorithmus, um eine möglichst kurze Folge von Verschiebeoperationen zu finden, die den Zustand

1	4	2
	8	3
7	6	5

in den Endzustand überführen. Tafel ✎.

Beispiel 2.15. [Rucksackproblem] Das Rucksackproblem lautet:

- Gegeben ist eine Menge $G = \{g_1, \ldots g_n\}$ von Gegenständen.
- Jeder Gegenstand $g \in G$ hat ein Gewicht $w(g)$ und liefert einen Profit $p(g)$.
- Weiterhin ist ein maximales Gesamtgewicht C gegeben.

Es soll nun eine Teilmenge $F \subseteq G$ der Gegenstände bestimmt werden, so daß:

- der Gesamtprofit $\sum_{g \in F} p(g)$ von F maximal ist
- unter der Nebenbedingung, daß das Gesamtgewicht der ausgewählten Gegenstände $\leq C$ ist, d.h. $\sum_{g \in F} w(g) \leq C$.

Zentrale Fragen:

- Wie kann man das Rucksackproblem als Suchproblem formulieren.
- Was muß ein A*-Algorithmus für das Rucksackproblem berücksichtigen?
- Wie erhält man eine Abschätzung des noch erzielbaren Nutzens?

A* und andere Suchverfahren

Bemerkung 2.1. Der A*-Algorithmus enthält die folgenden Algorithmen als Spezialfälle:

- Für $k \geq 0$ und $h \equiv 0$ erhält man den Dijkstra-Algorithmus.
- Für $k \equiv 0$ erhält man die Bestensuche.
- Für $k \equiv 1$ und $h \equiv 0$ erhält man die Breitensuche.
- Für $k \equiv -1$ und $h \equiv 0$ erhält man die Tiefensuche, wenn man Wiederbelebungen verbietet (Übergang von Closed in die Agenda).

Eigenschaften von A*

Satz 2.1. [Terminierung, Fairness] *Es gelte:*

- *Jeder Zustand besitzt nur endlich viele Nachfolgerzustände,*
- *es existiere ϵ, so daß für die Kosten $\mathrm{k}(s, s')$ bei einem Zustandsübergang stets $\mathrm{k}(s, s') \geq \epsilon > 0$ gilt und*
- *es gibt einen erreichbaren Zielzustand.*

$\Rightarrow$ *Dann terminiert A* nach endlich vielen Schritten mit dem Erreichen eines Zielzustandes.*

Bemerkung 2.2. Unter den gegebenen Voraussetzungen endet die Suche u.U. in einem nicht optimalen Zielzustand.

Eigenschaften von A* (2)

Satz 2.2. [Optimalität] *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.1 und*
- h *ist zulässig.*

$\Rightarrow$ *Dann ist der Zielknoten* z*, mit dem A* terminiert, ein optimaler Zielknoten,*

$\Rightarrow$ *die minimalen Kosten ergeben sich durch* $\mathrm{g}(z)$ *und*

$\Rightarrow$ *ausgehend von* $\mathrm{p}(z)$ *kann eine optimale Zustandsfolge ermittelt werden.*

Korollar 2.3. *Gegeben seien die Voraussetzungen von Satz 2.2. Der gefundene optimale Zielknoten sei* z*. Dann wurden während des Laufs von A* nur Zustände* s *mit* $\Phi(s) \leq \mathrm{g}(z)$ *expandiert.*

Wahl guter Schätzer

- Die Eigenschaften der heuristischen Funktion haben einen wesentlichen Einfluß auf die Performanz der Suche mit A*.
- Eine zulässige heuristische Funktion ist um so besser, je näher sie dem Optimalwert zur Erreichung eines Zielzustandes kommt.

Definition 2.7. Für zwei zulässige Schätzer h und h' heißt:

- h' *besser informiert* als h gdw. $h(s) < h'(s)$ für alle Zustände s gilt.
- h' *nicht schlechter informiert* als h gdw. $h(s) \leq h'(s)$ für alle Zustände s gilt.

Satz 2.4. *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.2,*
- A *bzw.* A' *seien A*-Algorithmen, die zulässige Schätzer* h *bzw.* h' *verwenden und*
- h' *sei besser informiert als* h*.*

$\Rightarrow$ *Dann wird jeder Zustand* s*, der von* A' *expandiert wird, auch von* A *expandiert.*

Satz 2.5. *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.1 und*
- $\mathrm{h}(s) \leq (1+\epsilon)\mathrm{h}^*(s)$ *für alle Zustände* s*.*

$\Rightarrow$ *Dann gilt für den vom A*-Algorithms ermittelten Zielzustand* z*:*

$$\mathrm{g}(z) \leq (1+\epsilon)\mathrm{g}(z^*)$$

mit z^* *ist Zielzustand einer optimalen Lösung.*

Monotone Schätzer

Definition 2.8. Gegeben sei eine nichtnegative Kostenfunktion k. Eine heuristische Funktion h heißt *monotoner Schätzer* gdw. gilt:

- $\mathrm{h}(z) = 0$ für alle Zielzustände z.
- Für alle Zustände s und alle Nachfolger s' von s gilt:

$$\mathrm{h}(s) \leq \mathrm{k}(s, s') + \mathrm{h}(s')$$

Beispiel 2.16. Alle Schätzer aus Beispiel 2.12 sind auch monotone Schätzer.

Satz 2.6. *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.1 und*
- h *sei ein monotoner Schätzer.*

$\Rightarrow$ *Dann ist* h *auch ein zulässiger Schätzer.*

$\Rightarrow$ *Ist der Knoten* s' *durch Expansion des Knotens* s *entstanden, so gilt* $\Phi(s) \leq \Phi(s')$.

$\Rightarrow$ *Es gibt keine Wiederbelebung von Zuständen, d.h. ein Knoten, der expandiert wurde, wird nie mehr selektiert.*

Satz 2.7. h *sei ein zulässiger Schätzer. Dann existiert ein monotoner Schätzer* h'*, der nicht schlechter informiert ist als* h.

Begründung: Betrachte Zustand s mit Vorgänger s_{pre}:

$$\mathrm{h}'(s) := \begin{cases} \mathrm{h}(s) & \text{falls } \mathrm{h}(s_{\mathrm{pre}}) \leq \mathrm{k}(s_{\mathrm{pre}}, s) + \mathrm{h}(s) \\ \mathrm{h}(s_{\mathrm{pre}}) - \mathrm{k}(s_{\mathrm{pre}}, s) & \text{sonst} \end{cases}$$

Dann gilt:

- h' ist monoton.
- h' ist zulässig, weil h zulässig ist.
- $\mathrm{h}(s) \leq \mathrm{h}'(s)$, also ist h' besser informiert als h.

Zusammenfassung des Kapitels

- Zustandsraum: Zustände, Zustandsübergänge, Startzustand, Zielzustände
- Systematische Suche im Zustandsraum: Breitensuche, Tiefensuche
- Heuristische Funktionen: Schätzung der Entfernung zum Ziel
- Bestensuche garantiert keine Optimalität
- A*: Operatorkosten plus heuristischer Funktion
- A* liefert optimale Lösungen bzgl. Operatorkosten