Informierte Suchverfahren

Bisher: Reihenfolge der Bearbeitung hängt ab von Position im Suchbaum

Wert eines Knotens *g(n)*: Knotentiefe bzw. Pfadkosten

Jetzt: Berücksichtige für Knoten *Schätzung*, „wie weit es noch zum Ziel ist“

Wert *h(n)*: Schätzung der Kosten von n bis zu einem Ziel

Die heuristische Funktion $h(n)$

Zulässigkeit: h *unterschätzt* die *tatsächlichen* Kosten h^* :

$h(n) \leq h^*(n)$

Monotonie: h-Wert von Knoten n zu Nachfolger n' nimmt mindestens um die Aktionskosten ab:

$h(n) \leq c(n,a,n')+h(n')$

Monotonie impliziert Zulässigkeit, aber nicht umgekehrt.

Dominanz: h_2 *dominiert* h_1 (ist „besser informiert“) :

$h_2(n) \geq h_1(n)$ für alle n, und beide sind zulässig

Knotenbewertung $f(n)=g(n)$: Werte nur *Pfadkosten*

Knotenbewertung $f(n)=h(n)$: Werte nur Rest-Kosten

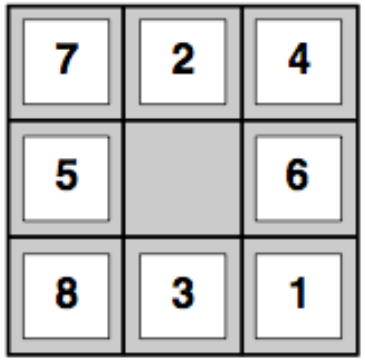
Knotenbewertung $f(n)=g(n) \oplus h(n)$: Werte beide Anteile

Beispiele für *h*-Funktionen im Verschiebespiel

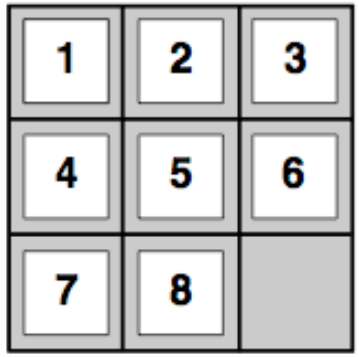
***h(n)* = 0**: monoton, daher zulässig
… und maximal uninformiert

h(n)* = Zahl falsch liegender Plättchen in *n:
zulässig (nicht monoton!)
Beispiel: *h(*Startzustand*)* = 6

h(n)* = „Manhattan-Distanz“ in *n:
zulässig (nicht monoton!)
Beispiel: *h(*Startzustand*)* = 14
(= 4+0+3+3+1+0+2+1)



STATE(*n*)

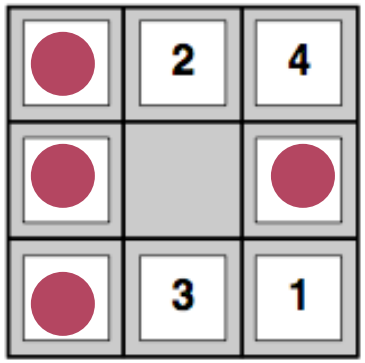


Zielzustand

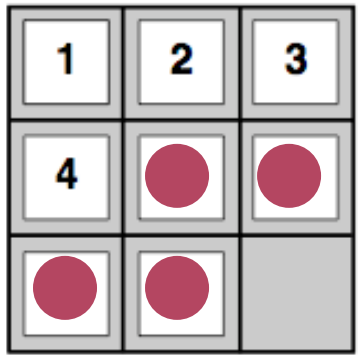
Wie findet man eine heuristische Funktion?

- Programmieren (d.h.: Intuition, Alltagswissen, Genialität, …)
- Systematische *Abstraktion* des Problems:
  Vereinfachte Problemversion, deren Lösung auf den Aufwand zur Lösung des vollen Problems rückschließen lässt.

Abstraktion im Verschiebspiel



STATE(*n*)



Zielzustand

Der Rechenaufwand für *h* darf nicht die Einsparung auffressen, die wir uns durch Einführung von *h* versprechen!

Bestensuche (*greedy best-first*)

… ein Beispiel für einen *greedy* („gierigen“) Algorithmus

- Geh vor wie bei TREESEARCH,
- bewerte Knoten durch ***f(n)=h(n)***
- sortiere bei INSERTALL nach Knotenwerten (billigste vor)

$\Rightarrow$ endet damit bei zuerst gefundenem Zielknoten

☹ Zeitbedarf: $O(b^m)$, wenn m Maximaltiefe des Baums

☹ Speicherbedarf: $O(b^m)$ (da alle Knoten im Speicher)

☹ Unvollständig (da anfällig für „Sackgassen“)

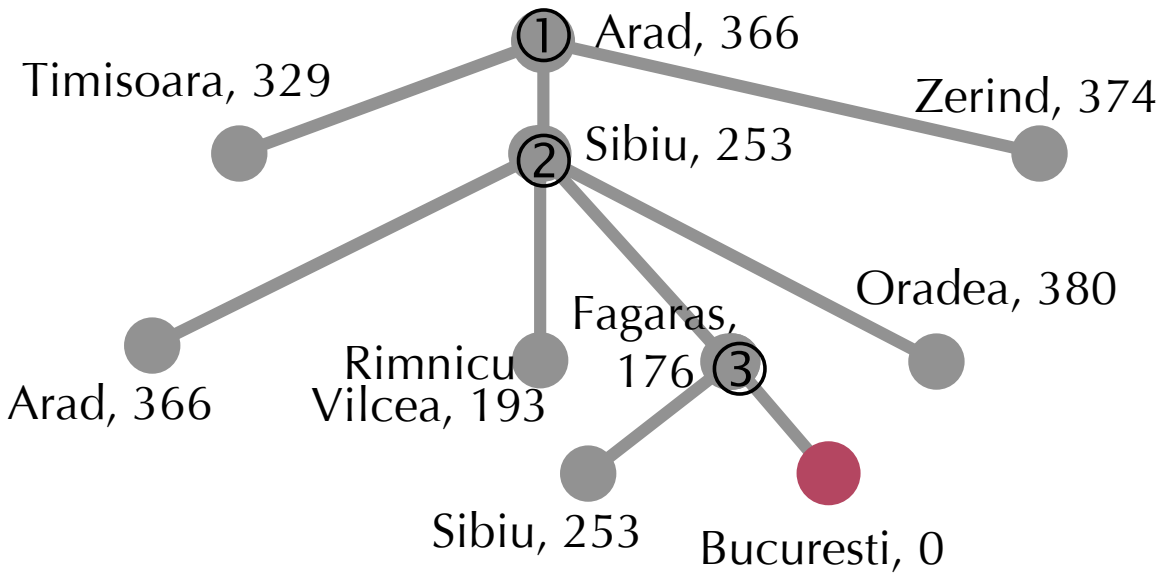
☹ Nicht optimal (siehe folgendes Beispiel)

Zeit und Speicher *praktisch* oft viel besser bei guter h-Funktion!

Beispiel: Bestensuche im Reiseproblem

Luftlinienentfernungen nach Bucuresti

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



Pfadkosten: 450 (siehe Straßenentfernungen)
optimal: 418

Der Algorithmus A*

- Geh vor wie bei *uniform-cost*-Suche,
- bewerte Knoten durch $f(n)=g(n)+h(n)$, wobei **$h(n)$ zulässig**
- sortiere bei INSERTALL nach Knotenwerten (billigste vor)
- ende erst, wenn ein Zielknoten expandiert werden müsste

☹ Zeitbedarf: $O(b^{(1+\lfloor C^*/\varepsilon \rfloor)})$ (*uniform-cost* ≈ A* für $h(n)=0$)

☹ Speicherbedarf: $O(b^{(1+\lfloor C^*/\varepsilon \rfloor)})$ (alle Knoten im Speicher)

☺ Vollständig (Details folgen)

☺ Optimal (Details folgen)

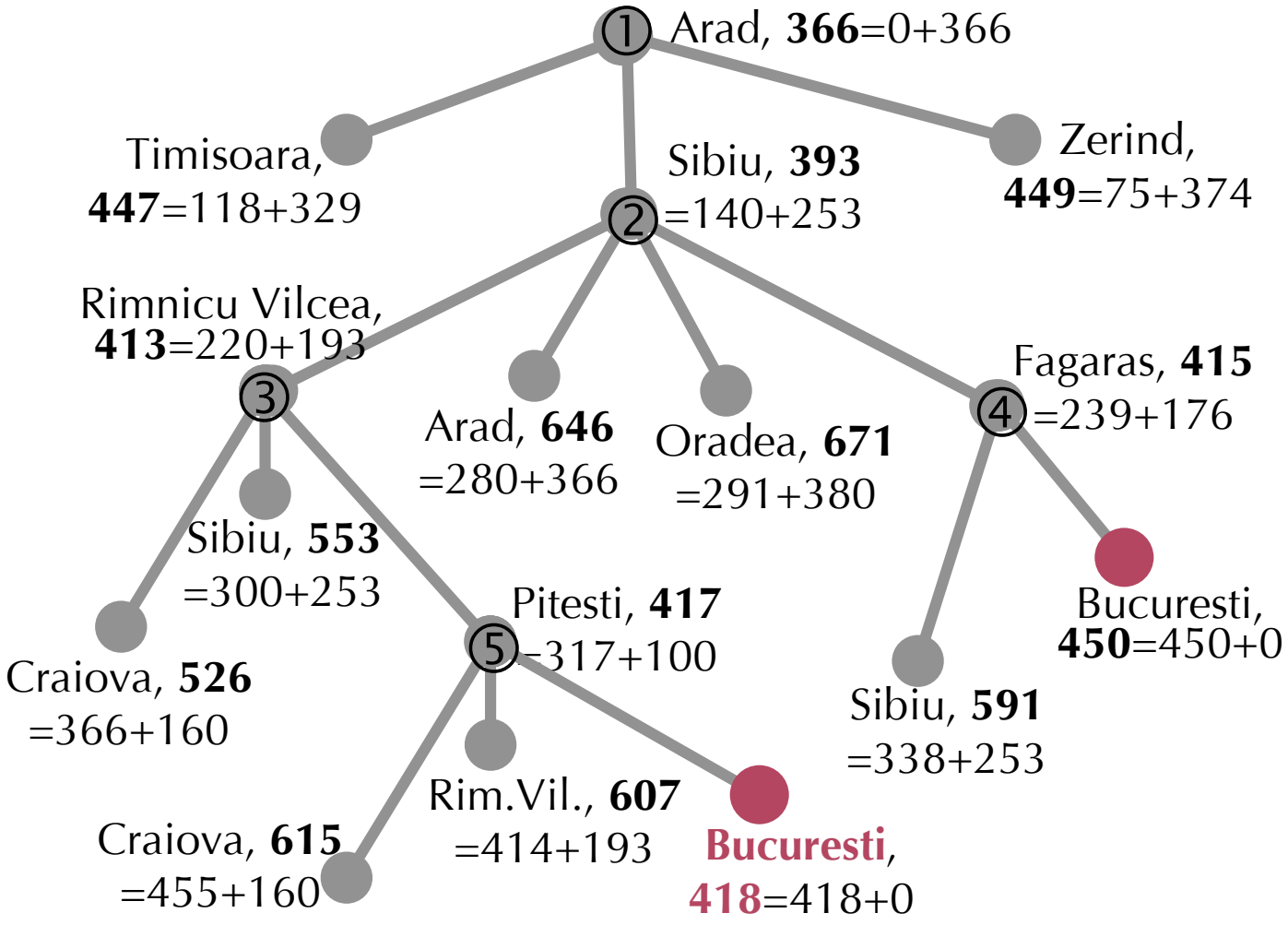
- Vollständigkeit & Optimalität schon früh bewiesen (1968)

Nils Nilsson, *1933

Beispiel: A* beim Reiseproblem

Luftlinienentfernungen nach Bucuresti

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



Optimalität und Vollständigkeit von A*

Ist für alle Knoten n die Bewertung $f(n)=g(n)+h(n)$ für zulässiges h, existiert eine Lösung, sind alle Aktionskosten >0 und ist $b<\infty$, so findet A* einen optimalen Lösungsknoten.

Beweisskizze (Widerspruchsbeweis)

Sei C^* Kosten einer optimalen Lösung im Knoten L, d.h. $f(L) = g(L)+h(L) = C^*+0$
Annahme: A* terminiert mit Lösung in $L^+ \neq L$, wobei $f(L^+) = C^+ > C^*$.

Nach Vorschrift müsste A* alle *fringe*-Knoten mit Kosten $< C^+$ expandiert haben; wenn A* einen Lösungsknoten expandieren müsste, terminiert er.
Wegen Zulässigkeit von h: Alle Vorgänger k von L haben Kosten $f(k) \leq C^* < C^+$
L und L^+ haben mindestens einen gemeinsamen, voll expandierten Vorgänger.
Folglich sind alle Vorgänger von L expandiert, folglich ist L in *fringe* gewesen.

Da $f(L) < f(L^+)$, terminiert A* mit der Lösung in L.

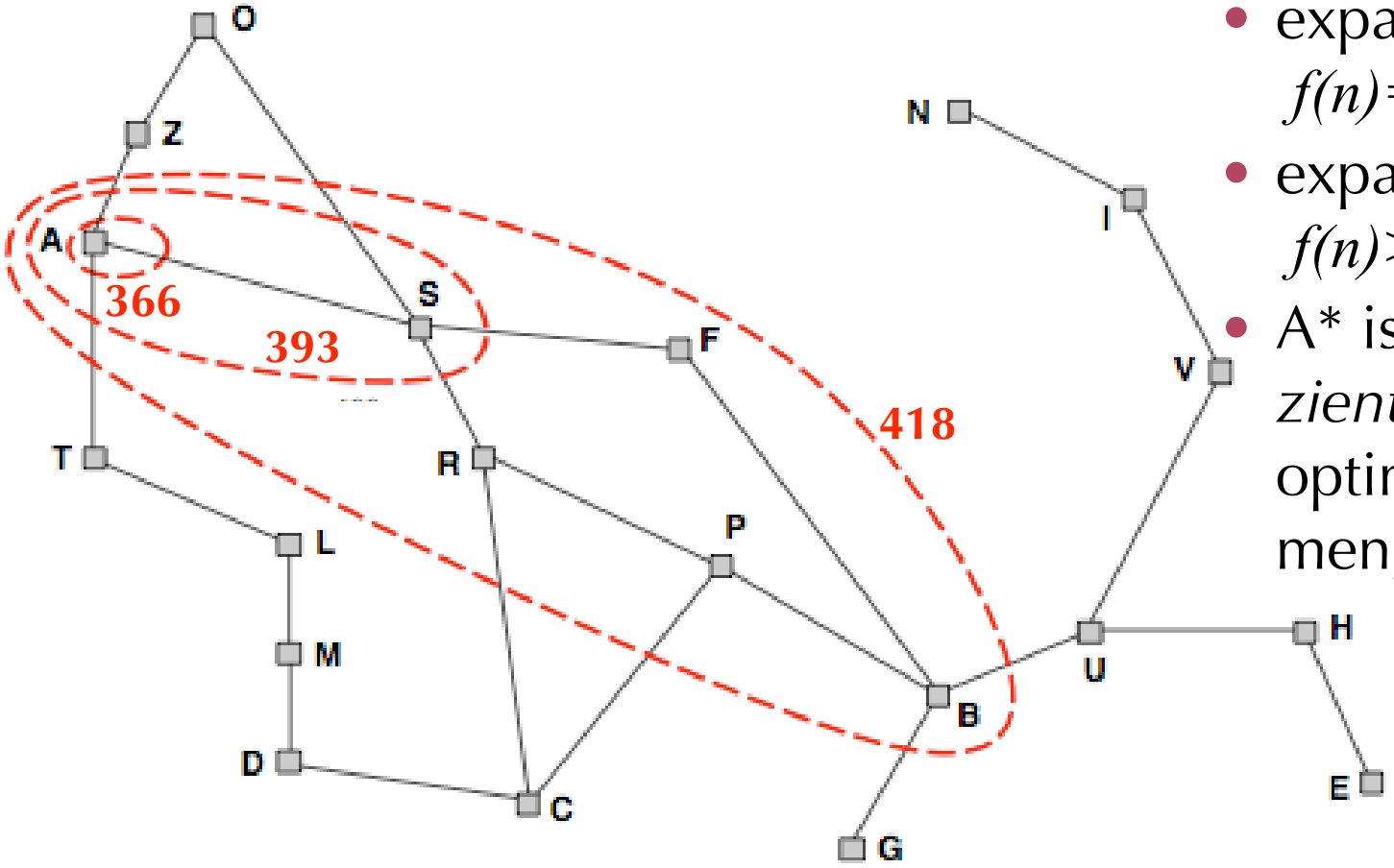
A* in GRAPHSEARCH-Variante

… geht analog!

Für Optimalität und Vollständigkeit braucht man monotones h!

Qualitatives Verhalten von A*

A* propagiert wachsende „f-Konturen“ ab Start, wobei $f_i < f_{i+1}$



- expandiert alle n: $f(n) < C^*$
- expandiert einige n: $f(n) = C^*$
- expandiert kein n: $f(n) > C^*$
- A* ist *optimal effizient* unter den optimalen Algorithmen, gegeben h

Einfluss der Informiertheit

Ist h_1 informierter als h_2,
so expandiert A*/h_2 mindestens alle die Knoten im Suchraum, welche A*/h_1 expandiert

(Hier ohne Beweis)

Andererseits (schlecht bei *vielen* Zielknoten im Suchraum):

A* macht „Breitensuche durch alle plausiblen Pfade“!

A* mit Iterierter Tiefensuche: IDA*

- Gleiches Vorgehen wie bei Einheitskosten:
  Beschränkte Tiefensuche mit Schranke erhöht in ε-Schritten
- Statt Tiefe $d(n)$ verwende $f(n)=g(n)+h(n)$ als Schranke

☹ Zeitbedarf: $O(b^{(1+\lfloor C^*/\varepsilon \rfloor)})$ (analog A* und *uniform-cost*)

☺ Speicherbedarf: $O(b(1+\lfloor C^*/\varepsilon \rfloor))$

☺ Vollständig (wie A*)

☺ Optimal (wie A*)

- Oft wird strikte Optimalität aufgegeben für größere Schrittweite $\delta>\varepsilon$: Lösung dann optimal bis auf $\delta-\varepsilon$

A* mit Speicherbegrenzung: SMA*

Gegeben feste Speicherkapazität für M Knoten, sodass

$b(1+\lfloor C^*/\varepsilon \rfloor) \ll M \ll b^{(1+\lfloor C^*/\varepsilon \rfloor)}$

- Wenn M Knoten im Speicher, Lösung nicht gefunden, (M+1)-ter Knoten wäre zu speichern:
  - wähle den schlechtest bewerteten *fringe*-Knoten H aus mit Bewertung $f(H)$
    (falls mehrere Knoten gleichwertig, nimm zuerst generierten)
  - für den Vorgängerknoten G von H setze $f(G):=f(H)$
  - verdränge H aus dem Speicher

☺ Vollständig und optimal, falls optimaler Pfad $<M$ Schritte

Zustandsraumsuche vs. Lösungsraumsuche

- Vielfach gibt es nicht nur eine einzige Lösung unter sehr vielen „unfertigen“ Zuständen,
- sondern sehr viele Lösungen sehr unterschiedlicher Qualität,
- und einige (typischerweise „schlechte“) Lösungen sind effizient konstruierbar.
- „Dasselbe“ Problem kann auf beide Arten repräsentierbar sein.

Beispiel: n-Damen-Problem

- Zustandsraumsuche startet mit leerem $n \times n$-Brett; findet Folge von n Aktionen „Setze_Dame(i,y)“, bis n Damen unbedroht auf dem Brett
- Lösungsraumsuche startet mit beliebiger Verteilung von n Damen in n Spalten; verbessert aktuelle Verteilung, bis frei von Bedrohung