

Grundlagen der Künstlichen Intelligenz

6. Suchalgorithmen: Bestensuche

Malte Helmert

Universität Basel

22. März 2013

Grundlagen der Künstlichen Intelligenz

22. März 2013 — 6. Suchalgorithmen: Bestensuche

6.1 Einleitung

6.2 Bestensuche

6.3 Gierige Bestensuche

6.4 A*

6.5 Weighted A*

6.6 IDA*

6.7 Zusammenfassung

6.1 Einleitung

Heuristische Suchverfahren

Heuristische Suchverfahren:

Suchverfahren, die Heuristiken verwenden

- ▶ **systematische** Suche (**Bestensuche**):
 - ▶ gierige Bestensuche
 - ▶ A*
 - ▶ Weighted A*
 - ▶ IDA*
- ▶ **lokale** Suche:
 - ▶ Hill-Climbing
 - ▶ Simulierte Abkühlung
 - ▶ genetische Algorithmen

Suchprobleme: Überblick

Kapitelüberblick klassische Suchprobleme:

- ▶ Formalisierung von Suchproblemen (↔ Kapitel 3)
- ▶ blinde Suchverfahren (↔ Kapitel 4)
- ▶ Heuristiken (↔ Kapitel 5)
- ▶ **Bestensuche** (↔ dieses Kapitel)
- ▶ Eigenschaften von A^* (↔ Kapitel 7)
- ▶ Lokale Suche (↔ Kapitel 8)

6.2 Bestensuche

Bestensuche

Bestensuche (**best-first search**) ist eine Klasse von Suchverfahren, bei der in jedem Schritt der „beste“ Knoten expandiert wird.

- ▶ bei Entscheidung, welcher Knoten am besten ist, wird Heuristikfunktion verwendet
 - ▶ aber nicht unbedingt ausschliesslich
- ▶ als Graphen- oder Baumsuche möglich
- ▶ Normalfall: **Graphensuche** (d. h. mit Duplikatelimination)

Bestensuche: Varianten

Bestensuche

Eine **Bestensuche** ist ein heuristisches Suchverfahren, das Suchknoten mit einer **Bewertungsfunktion** f bewertet und immer einen Knoten n mit minimalem $f(n)$ expandiert.

- ▶ unterschiedliche Definitionen für f
 - ↔ unterschiedliche Suchverfahren
- ▶ Implementation wie **uniforme Kostensuche**
 - ▶ Bestensuche mit $f(n) := g(n)$ ist uniforme Kostensuche
 - ▶ Allerdings würde man ein Verfahren, das h nicht verwendet, normalerweise nicht „Bestensuche“ nennen

Bestensuche: Pseudo-Code (ohne Reopening)

Bestensuche (mit verzögerter Duplikatelim., ohne Reopening)

```

open := new min-heap ordered by f
open.insert(make-root-node(init()))
closed := {}
while not open.empty():
    n = open.pop-min()
    if n.state ∉ closed:
        closed := closed ∪ {n.state}
        if is-goal(n.state):
            return extract-solution(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            if h(s') < ∞:
                n' := make-node(n, a, s')
                open.insert(n')
return unsolvable

```

Reopening

- ▶ **Erinnerung:** uniforme Kostensuche besucht Zustände in Reihenfolge aufsteigender g -Werte
- ↪ garantiert, dass bei Expansion eine Knotens **billigster Pfad** zu dessen Zustand gefunden wurde
- ▶ bei Bestensuche gilt dies im Allgemeinen **nicht**
- ↪ manche Bestensuchverfahren expandieren bereits betrachtete Zustände erneut, wenn billigerer Pfad gefunden (**Reopening**)

Bestensuche: Pseudo-Code (mit Reopening)

Bestensuche (mit verzögerter Duplikatelimination und Reopening)

```

open := new min-heap ordered by f
open.insert(make-root-node(init()))
distances := new hash-table
while not open.empty():
    n = open.pop-min()
    if n.state ∉ distances or g(n) < distances[n.state]:
        distances[n.state] := g(n)
        if is-goal(n.state):
            return extract-solution(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            if h(s') < ∞:
                n' := make-node(n, a, s')
                open.insert(n')
return unsolvable

```

distances steuert Reopening und übernimmt Rolle von *closed*

Die wichtigsten Bestensuchverfahren

Die wichtigsten Bestensuchverfahren:

- ▶ $f(n) = h(n)$:
↪ **gierige Bestensuche** (greedy best-first search)
↪ nur die Heuristik zählt
 - ▶ $f(n) = g(n) + h(n)$:
↪ **A***
↪ Kombination aus Pfadkosten und Heuristik
 - ▶ $f(n) = g(n) + w \cdot h(n)$:
↪ **Weighted A***
↪ ähnlich A*; $w \in \mathbb{R}_0^+$ ist ein Parameter
- ↪ im Folgenden: detailliertere Besprechung dieser Algorithmen

6.3 Gierige Bestensuche

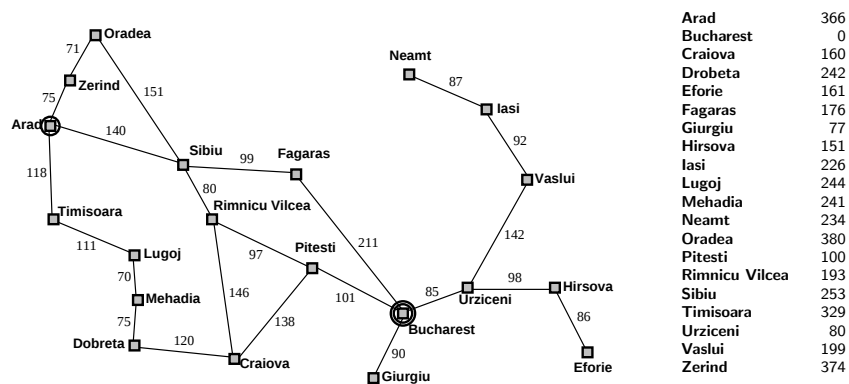
Gierige Bestensuche

Gierige Bestensuche

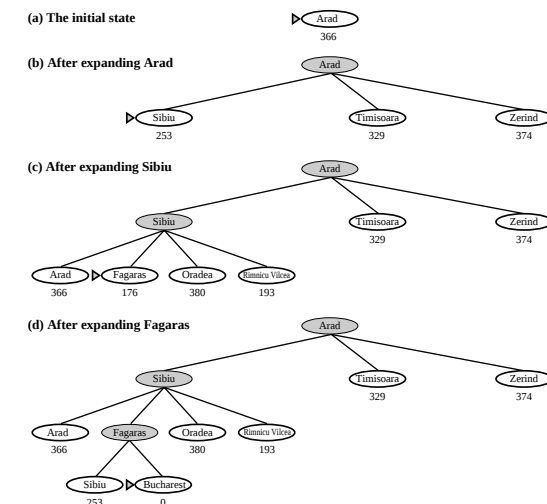
Berücksichtigt nur die Heuristik: $f(n) = h(n)$

- ▶ normalerweise aus Effizienzgründen ohne Reopening implementiert
- ▶ wir betrachten nur diese Variante

Beispiel: gierige Bestensuche für Routenplanung



Beispiel: gierige Bestensuche für Routenplanung



Gierige Bestensuche: Eigenschaften

- ▶ **vollständig** für **sichere** Heuristiken (wegen Duplikatelimination)
- ▶ **suboptimal** (Lösung kann **beliebig schlecht** sein)
- ▶ oft eines der besten Suchverfahren in der Praxis
- ▶ monotone Transformationen von h (z. B. Skalierung, Addition einer Konstante) ändern nicht das Verhalten

6.4 A*

A*

A*

Kombiniere gierige Suche mit uniformer Kostensuche:

$$f(n) = g(n) + h(n)$$

- ▶ **Abwägen** zwischen Pfadkosten und Zielnähe
- ▶ $f(n)$ schätzt Kosten der günstigsten Lösung vom Anfangszustand über n ins Ziel

A*: Zitierungen

Google scholar hart nilsson raphael Search

Scholar Articles and patents anytime include citations Create email alert

[A formal basis for the heuristic determination of minimum cost paths](#)
 ... N.J. Nilsson, B. Raphael - Systems Science and ... 1968 - [ieeexplore.ieee.org](#)
 Abstract Although the problem of determining the minimum cost path through a graph arises naturally in a number of interesting applications, there has been no underlying theory to guide the development of efficient search procedures. Moreover, there is no adequate ...
 Cited by 195 - Related articles - All 13 versions

[Correction to a formal basis for the heuristic determination of minimum cost paths](#)
 ... N.J. Nilsson, B. Raphael - ACM SIGART Bulletin, 1972 - [dl.acm.org](#)
 Abstract Our paper on the use of heuristic information in graph searching defined a path-finding algorithm, A*, and proved that it had two important properties. In the notation of the paper, we proved that if the heuristic function $h(n)$ is a lower bound on the true minimal ...
 Cited by 195 - Related articles - All 6 versions

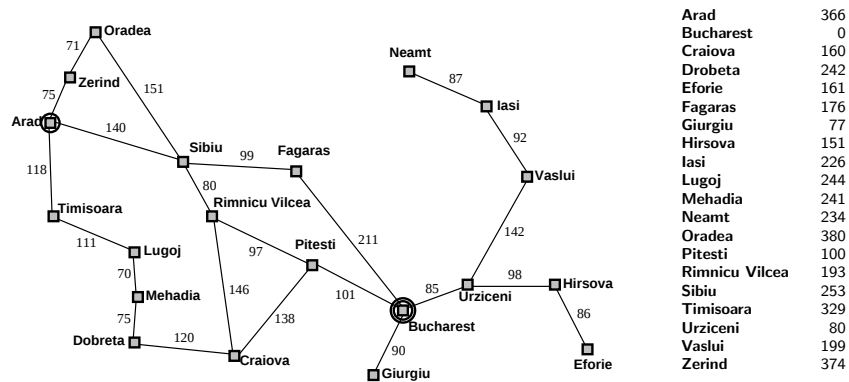
[CITATION] Bi-directional search
 I. Pohl - 1970 - IBM TJ Watson Research Center
 Cited by 337 - Related articles - Find in IDS Basel/Bern - All 2 versions

[CITATION] and B. Raphael. A formal basis for heuristic determination of minimum path cost
 ... N.J. Nilsson - IEEE Transaction on SSC, 1968
 Cited by 28 - Related articles

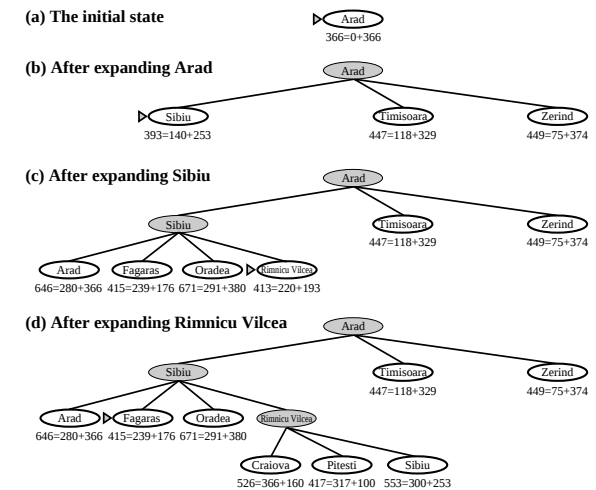
[CITATION] Research and applications—artificial intelligence
 B. Raphael, R. Duda, R. E. Fikes, P. E. Hart, N. Nilsson... - Final Report, SRI Project, 1971
 Cited by 13 - Related articles

[A mobile automaton: An application of artificial intelligence techniques](#)
 N.J. Nilsson - 1969 - DTIC Document
 A MOBILE AUTOMATON: AN APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES

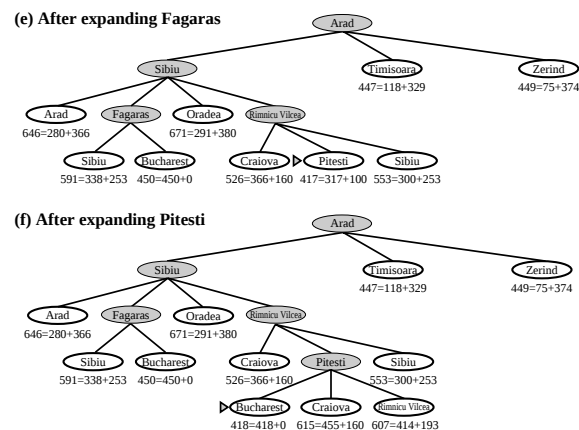
Beispiel: A* für Routenplanung



Beispiel: A* für Routenplanung



Beispiel: A* für Routenplanung



A*: Eigenschaften

- ▶ **vollständig** für **sichere** Heuristiken
- ▶ **mit Reopening**:
 - ▶ **optimal**, wenn Heuristik **zulässig**
- ▶ **ohne Reopening**:
 - ▶ **optimal**, wenn Heuristik **konsistent** und **zielerkennend**

⇝ Beweise: nächstes Kapitel

A*: Implementierungsaspekte

Einige praktische Hinweise für die Implementation von A*:

- ▶ Bei Knoten mit **identischen** f -Werten wird der h -Wert zur Heap-Ordnung herangezogen. Dabei werden Knoten mit niedrigen h -Werten bevorzugt.
- ▶ **häufiger Bug**: Reopening wird vergessen, obwohl Konsistenz nicht garantiert ist
- ▶ **häufiger Bug**: Duplikatelimination wird schon bei der **Erzeugung** von Knoten vorgenommen
- ▶ **häufiger Bug**: Zieltest findet an der falschen Stelle statt
- ▶ all diese Bugs können lange unerkannt bleiben und führen im Allgemeinen zum Verlust der Optimalität

6.5 Weighted A*

Weighted A*

Weighted A*

A* mit stärker gewichteter Heuristik: $f(n) = g(n) + w \cdot h(n)$, wobei **Gewicht** $w \in \mathbb{R}_0^+$ mit $w \geq 1$ frei wählbarer Parameter

Weighted A*: Eigenschaften

Das frei wählbare Gewicht steuert, wie gierig der Algorithmus ist:

- ▶ $w = 0$: wie uniforme Kostensuche
- ▶ $w = 1$: wie A*
- ▶ $w \rightarrow \infty$: wie gierige Bestensuche

Mit $w \geq 1$ analoge Eigenschaften zu A*:

- ▶ **h konsistent und zielerkennend**:
gefundene Lösung höchstens Faktor w teurer als Optimum;
kein Reopening nötig
- ▶ **h zulässig**:
gefundene Lösung höchstens Faktor w teurer als Optimum,
wenn Reopening verwendet wird

(ohne Beweis)

6.6 IDA*

IDA*

Der Hauptnachteil der vorgestellten Bestensuchverfahren ist ihr Platzaufwand.

Idee: ähnlicher Ansatz wie bei der iterativen Tiefensuche:

- ▶ beschränkte Tiefensuche mit ansteigenden Schranken
- ▶ statt der **Tiefe** beschränken wir **f**
- ↪ IDA* (iterative-deepening A*)
- ▶ anders als die anderen Bestensuchverfahren eine **Baumsuche**

IDA*: Pseudo-Code (Hauptprozedur)

IDA*: Hauptprozedur

```

 $n_0 := \text{make-root-node}(\text{init}())$ 
 $f\text{-limit} := 0$ 
while  $f\text{-limit} \neq \infty$ :
     $\langle f\text{-limit}, \text{solution} \rangle := \text{recursive-search}(n_0, f\text{-limit})$ 
    if  $\text{solution} \neq \text{none}$ :
        return  $\text{solution}$ 
return unsolvable
  
```

IDA*: Pseudo-Code (Tiefensuche)

function recursive-search($n, f\text{-limit}$):

```

if  $f(n) > f\text{-limit}$ :
    return  $\langle f(n), \text{none} \rangle$ 
if is-goal( $n.\text{state}$ ):
    return  $\langle \text{none}, \text{extract-solution}(n) \rangle$ 
 $\text{next-limit} := \infty$ 
for each  $\langle a, s' \rangle \in \text{succ}(n.\text{state})$ :
    if  $h(s') < \infty$ :
         $n' := \text{make-node}(n, a, s')$ 
         $\langle \text{rec-limit}, \text{solution} \rangle := \text{recursive-search}(n', f\text{-limit})$ 
        if  $\text{solution} \neq \text{none}$ :
            return  $\langle \text{none}, \text{solution} \rangle$ 
         $\text{next-limit} := \min(\text{next-limit}, \text{rec-limit})$ 
return  $\langle \text{next-limit}, \text{none} \rangle$ 
  
```

(Praktische Implementierung verwendet in der Regel keine explizite Datenstruktur für Knoten.)

IDA*: Eigenschaften

Erbt wichtige Eigenschaften von A* und iterativer Tiefensuche:

- ▶ **semi-vollständig**, wenn h sicher und $cost(a) > 0$ für alle Aktionen a
- ▶ **optimal**, wenn h zulässig
- ▶ expandiert für zulässiges h nur Zustände mit $f_h^*(s) \leq c^*$, wobei c^* optimale Lösungskosten
- ▶ **Platzaufwand** $O(lb)$, wobei l Länge des längsten erzeugten Pfades (bei Einheitskosten: $\leq c^*$), b Verzweigungsgrad

IDA*: weitere Eigenschaften

- ▶ gegenüber A* potenziell erheblicher Mehraufwand durch Nichterkennen von Duplikaten
 - ↪ bei vielen Problemen exponentiell mehr Zeitaufwand
 - ↪ oft mit teilweiser Duplikatelimination kombiniert (Zykelerkennung, Transpositionstabellen)
- ▶ Zusatzkosten durch iteratives Erhöhen der f -Schranke oft, aber nicht immer vernachlässigbar
 - ▶ problematisch vor allem, wenn viele stark unterschiedliche Aktionskosten existieren, weil dann möglich ist, dass bei jeder Schrankenerhöhung nur wenige neue Zustände erreicht werden

6.7 Zusammenfassung

Zusammenfassung

- ▶ **Bestensuche** expandiert immer den anhand der f -Funktion besten Zustand

verschiedene Verfahren durch verschiedene f -Funktionen:

- ▶ $f = h$: **gierige Bestensuche**
suboptimal, oft sehr effizient
- ▶ $f = g + h$: **A***
optimal, falls h zielerkennend und konsistent
oder falls h zulässig und **Reopening** durchgeführt wird
- ▶ $f = g + w \cdot h$: **Weighted A***
für $w \geq 1$ Suboptimalitätsfaktor höchstens w ,
wenn Bedingungen für die Optimalität von A* gelten

Speichereffiziente Varianten:

- ▶ **IDA*** ist eine Baumsuchvariante von A*,
die auf der Idee der iterativen Tiefensuche aufbaut