

Grundlagen der Künstlichen Intelligenz

15. Klassische Suche: Bestensuche als Graphensuche

Malte Helmert

Universität Basel

23. März 2015

Grundlagen der Künstlichen Intelligenz

23. März 2015 — 15. Klassische Suche: Bestensuche als Graphensuche

15.1 Einführung

15.2 Bestensuche

15.3 Algorithmen-Details

15.4 Reopening

15.5 Zusammenfassung

Klassische Suche: Überblick

Kapitelüberblick klassische Suche:

- ▶ 5.–7. Grundlagen
- ▶ 8.–12. Basialgorithmen
- ▶ 13.–20. heuristische Algorithmen
 - ▶ 13. Heuristiken
 - ▶ 14. Analyse von Heuristiken
 - ▶ 15. Bestensuche als Graphensuche
 - ▶ 16. Gierige Bestensuche, A*, Weighted A*
 - ▶ 17. IDA*
 - ▶ 18. A*: Optimalität, Teil I
 - ▶ 19. A*: Optimalität, Teil II
 - ▶ 20. A*: Vollständigkeit und Komplexität

15.1 Einführung

Heuristische Suchalgorithmen

heuristische Suchalgorithmen

Heuristische Suchalgorithmen verwenden **Heuristikfunktionen**, um die Reihenfolge der Knotenexpansion teilweise oder vollständig zu bestimmen.

- ▶ **dieses Kapitel:** kurze Einführung
- ▶ **Folgekapitel:** gründlichere Analyse

15.2 Bestensuche

Bestensuche

Bestensuche ist eine Klasse von Suchalgorithmen, die in jedem Schritt den „am besten aussehenden“ Knoten expandieren.

- ▶ Entscheidung, welcher Knoten am besten ist, verwendet **Heuristik**...
- ▶ ...aber **nicht notwendigerweise ausschliesslich**.

Bestensuche

Eine **Bestensuche** ist ein heuristischer Suchalgorithmus, der Suchknoten anhand einer **Bewertungsfunktion** f evaluiert und immer einen Knoten n mit minimalem $f(n)$ expandiert.

- ▶ Implementierung im Wesentlichen wie **uniforme Kostensuche**
- ▶ unterschiedliche Wahl von f
 $\rightsquigarrow$ unterschiedliche Suchalgorithmen

Die wichtigsten Bestensuchalgorithmen

Die wichtigsten Bestensuchalgorithmen:

- ▶ $f(n) = h(n.state)$: **gierige Bestensuche** (greedy best-first search)
 $\rightsquigarrow$ nur die Heuristik zählt
- ▶ $f(n) = g(n) + h(n.state)$: **A^***
 $\rightsquigarrow$ Kombination von Pfadkosten und Heuristik
- ▶ $f(n) = g(n) + w \cdot h(n.state)$: **Weighted A^***
 $w \in \mathbb{R}_0^+$ ist ein Parameter
 $\rightsquigarrow$ interpoliert zwischen gieriger Bestensuche und A^*

$\rightsquigarrow$ **Eigenschaften:** nächste Kapitel

Was erhalten wir mit $f(n) := g(n)$?

Bestensuche: Graphen- oder Baumsuche?

Bestensuche kann eine **Graphensuche** oder eine **Baumsuche** sein.

- ▶ hier: **Graphensuche** (d. h., mit Duplikateliminierung), was der häufigere Fall ist
- ▶ **Kapitel 17**: eine Baumsuch-Variante

15.3 Algorithmen-Details

Erinnerung: uniforme Kostensuche

Erinnerung: uniforme Kostensuche

Uniforme Kostensuche

```

open := new MinHeap ordered by g
open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
    n := open.pop_min()
    if n.state ∉ closed:
        closed.insert(n)
        if is_goal(n.state):
            return extract_path(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable

```

Bestensuche ohne Reopening (1. Versuch)

Bestensuche ohne Reopening (1. Versuch)

Bestensuche ohne Reopening (1. Versuch)

```

open := new MinHeap ordered by f
open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
    n := open.pop_min()
    if n.state ∉ closed:
        closed.insert(n)
        if is_goal(n.state):
            return extract_path(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable

```

Bestensuche ohne Reopening (1. Versuch): Diskussion

Diskussion:

Das ist schon fast alles.

Zwei nützliche Verbesserungen:

- ▶ **verwirf Zustände, die die Heuristik als unlösbar betrachtet**
 $\leadsto$ benötigen dann keinen Platz in *open*
- ▶ wenn mehrere Suchknoten denselben *f*-Wert aufweisen,
verwende *h* zum Tie-Breaking (bevorzuge niedriges *h*)
 - ▶ nicht immer eine gute Idee, aber oft
 - ▶ offensichtlich unnötig, wenn $f = h$ (gierige Bestensuche)

Bestensuche ohne Reopening (endgültige Version)

Bestensuche ohne Reopening

```

open := new MinHeap ordered by  $\langle f, h \rangle$ 
if  $h(\text{init}()) < \infty$ :
    open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
     $n := \text{open.pop\_min}()$ 
    if  $n.\text{state} \notin \text{closed}$ :
        closed.insert( $n$ )
        if is_goal( $n.\text{state}$ ):
            return extract_path( $n$ )
        for each  $\langle a, s' \rangle \in \text{succ}(n.\text{state})$ :
            if  $h(s') < \infty$ :
                 $n' := \text{make\_node}(n, a, s')$ 
                open.insert( $n'$ )
return unsolvable
  
```

Bestensuche: Eigenschaften

Eigenschaften:

- ▶ **vollständig**, wenn *h* sicher ist (Warum?)
- ▶ **Optimalität** hängt von *f* ab
 $\leadsto$ folgende Kapitel

15.4 Reopening

Reopening

- ▶ **Erinnerung:** uniforme Kostensuche besucht Knoten in Reihenfolge ansteigender g -Werte
- ↪ garantiert, dass **billigster Pfad** zum Zustand eines Knoten gefunden wurde, wenn der Knoten expandiert wird
- ▶ mit beliebigen f -Funktionen in der Bestensuche gilt dies im Allgemeinen **nicht**
- ↪ um billigere Lösungen zu finden, kann es sinnvoll sein, **Duplikatknoten zu expandieren**, wenn billigere Pfade zu deren Zuständen gefunden werden (**Reopening**)

Bestensuche mit Reopening

Bestensuche mit Reopening

```

open := new MinHeap ordered by  $\langle f, h \rangle$ 
if  $h(\text{init}()) < \infty$ :
    open.insert(make_root_node())
distances := new HashTable
while not open.is_empty():
    n := open.pop_min()
    if distances.lookup(n.state) = none or  $g(n) < \text{distances}[n.state]$ :
        distances[n.state] :=  $g(n)$ 
        if is_goal(n.state):
            return extract_path(n)
        for each  $\langle a, s' \rangle \in \text{succ}(n.state)$ :
            if  $h(s') < \infty$ :
                n' := make_node(n, a, s')
                open.insert(n')
return unsolvable

```

↪ *distances* steuert Reopening und ersetzt *closed*

15.5 Zusammenfassung

- ▶ **Bestensuche** expandiert immer Knoten mit minimalem Wert der **Bewertungsfunktion f**
 - ▶ $f = h$: **gierige Bestensuche**
 - ▶ $f = g + h$: **A***
 - ▶ $f = g + w \cdot h$ für Parameter $w \in \mathbb{R}_0^+$: **Weighted A***
- ▶ **hier:** Bestensuche als Graphensuche
- ▶ **Reopening:** expandiere Duplikate mit niedrigeren Pfadkosten, um billigere Lösungen zu finden

Zusammenfassung