

Grundlagen der Künstlichen Intelligenz

11. Klassische Suche: uniforme Kostensuche

Malte Helmert

Universität Basel

16. März 2015

Grundlagen der Künstlichen Intelligenz

16. März 2015 — 11. Klassische Suche: uniforme Kostensuche

11.1 Einführung

11.2 Algorithmus

11.3 Eigenschaften

11.4 Zusammenfassung

Klassische Suche: Überblick

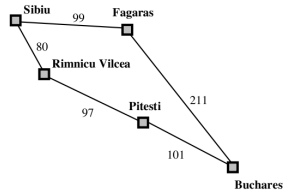
Kapitelüberblick klassische Suche:

- ▶ 5.–7. Grundlagen
- ▶ 8.–12. Basisalgorithmen
 - ▶ 8. Datenstrukturen für Suchalgorithmen
 - ▶ 9. Baumsuche und Graphensuche
 - ▶ 10. Breitensuche
 - ▶ 11. **uniforme Kostensuche**
 - ▶ 12. Tiefensuche und iterative Tiefensuche
- ▶ 13.–20. heuristische Algorithmen

11.1 Einführung

Uniforme Kostensuche

- ▶ Breitensuche optimal, wenn alle Aktionskosten gleich
- ▶ sonst Optimalität nicht garantiert $\rightsquigarrow$ Beispiel:



Abhilfe: uniforme Kostensuche

- ▶ expandiere immer Knoten mit **minimalen Pfadkosten** ($n.\text{path_cost}$ alias $g(n)$)
- ▶ Implementierung: **Prioritätswarteschlange** (Min-Heap) für Open-Liste

11.2 Algorithmus

Erinnerung: generischer Graphensuchalgorithmus

Erinnerung aus Kapitel 9:

Generische Graphensuche

```

open := new OpenList
open.insert(make_root_node())
closed := new ClosedList
while not open.is_empty():
    n := open.pop()
    if closed.lookup(n.state) = none:
        closed.insert(n)
        if is_goal(n.state):
            return extract_path(n)
        for each <a, s'> in succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable
  
```

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 not in closed:
        closed.insert(n)
        if is_goal(n.state):
            return extract_path(n)
        for each <a, s'> in succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable
  
```

Uniforme Kostensuche: Diskussion

Anpassung der generischen Graphensuche für uniforme Kostensuche:

- ▶ hier wären frühe Zieltests/frühe Updates der Closed-Liste **keine** gute Idee. (Warum nicht?)
- ▶ wie in BFS-Graph reicht eine **Menge** für die Closed-Liste aus
- ▶ eine Baumsuchvariante ist möglich, aber selten: dieselben Nachteile wie BFS-Tree und im allgemeinen **nicht einmal semi-vollständig** (Warum nicht?)

Anmerkung: identisch mit **Dijkstras Algorithmus** für kürzeste Pfade in gewichteten Graphen!
(bei beiden: Variante mit/ohne verzögerte Duplikateliminierung)

Uniforme Kostensuche: Verbesserungen

Mögliche Verbesserungen:

- ▶ wenn Aktionskosten kleine Ganzzahlen sind, sind **Bucket-Heaps** oft effizienter
- ▶ zusätzliche frühe Duplikatstests für erzeugten Knoten können Speicheraufwand reduzieren und Laufzeit verbessern oder verschlechtern

11.3 Eigenschaften

Vollständigkeit und Optimalität

Eigenschaften der uniformen Kostensuche:

- ▶ uniforme Kostensuche ist **vollständig** (Warum?)
- ▶ uniforme Kostensuche ist **optimal** (Warum?)

Zeit- und Platzaufwand

Eigenschaften der uniformen Kostensuche:

- ▶ **Zeitaufwand** hängt von Verteilung der Aktionskosten ab (keine einfachen und genauen Schranken).
 - ▶ Sei $\varepsilon := \min_{a \in A} \text{cost}(a)$ und gelte $\varepsilon > 0$.
 - ▶ Seien c^* die optimalen Lösungskosten.
 - ▶ Sei b der Verzweigungsgrad und gelte $b \geq 2$.
 - ▶ Dann beträgt der Zeitaufwand höchstens $O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$. (Warum?)
 - ▶ oft eine sehr schwache obere Schranke
- ▶ **Speicheraufwand** = Zeitaufwand

11.4 Zusammenfassung

Zusammenfassung

uniforme Kostensuche:
expandiere Knoten in Reihenfolge **aufsteigender Pfadkosten**

- ▶ üblicherweise als Graphensuche
- ▶ entspricht dann Dijkstra-Algorithmus
- ▶ **vollständig** und **optimal**