

Grundlagen der Künstlichen Intelligenz

9. Klassische Suche: Baumsuche und Graphensuche

Malte Helmert

Universität Basel

13. März 2015

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

Einführung

Suchalgorithmen

Suchalgorithmen allgemein

- Ausgehend vom **Anfangszustand**,
- **expandiere** wiederholt einen Zustand durch **Erzeugen** seiner **Nachfolger**.
- Höre auf, wenn ein **Zielzustand** expandiert wird
- oder **alle erreichbaren Zustände** betrachtet wurden.

Suchalgorithmen

Suchalgorithmen allgemein

- Ausgehend vom **Anfangszustand**,
- **expandiere** wiederholt einen Zustand durch **Erzeugen** seiner **Nachfolger**.
- Höre auf, wenn ein **Zielzustand** expandiert wird
- oder **alle erreichbaren Zustände** betrachtet wurden.

In diesem Kapitel betrachten wir zwei wesentliche Klassen von Suchalgorithmen:

- **Baumsuche** und
- **Graphensuche**

(Zu jeder Klasse gehört eine Vielzahl konkreter Algorithmen.)

Baumsuche

Baumsuche

Baumsuche

- mögliche Pfade, die exploriert werden können, sind in einem Baum (**Suchbaum**) organisiert
- **Suchknoten** entsprechen 1:1 den **Pfaden** vom Anfangszustand
- **Duplikate** (auch: **Transpositionen**) sind möglich, d. h. mehrere Knoten mit demselben Zustand
- Suchbaum kann unbegrenzt in die Tiefe wachsen

Generischer Baumsuchalgorithmus

Generische Baumsuche

```
open := new OpenList
open.insert(make_root_node())
while not open.is_empty():
    n := open.pop()
    if is_goal(n.state):
        return extract_path(n)
    for each  $\langle a, s' \rangle \in \text{succ}(n.\text{state})$ :
        n' := make_node(n, a, s')
        open.insert(n')
return unsolvable
```

Generischer Baumsuchalgorithmus: Diskussion

Diskussion:

- **generisches Muster** für Baumsuchalgorithmen
- ↪ für konkreten Algorithmus müssen wir (mindestens) entscheiden, wie Open-Liste implementiert wird
- konkrete Algorithmen oft leicht anders implementiert (aus Effizienzgründen), folgen aber **konzeptuell** dem Muster (= bauen denselben Suchbaum auf)

Graphensuche

Erinnerung: Baumsuche

Erinnerung:

Baumsuche

- mögliche Pfade, die exploriert werden können, sind in einem Baum (**Suchbaum**) organisiert
- **Suchknoten** entsprechen 1:1 den **Pfaden** vom Anfangszustand
- **Duplikate** (auch: **Transpositionen**) sind möglich, d. h. mehrere Knoten mit demselben Zustand
- Suchbaum kann unbegrenzt in die Tiefe wachsen

Graphensuche

Graphensuche

Unterschiede zur Baumsuche:

- erkenne **Duplikate**: wenn ein Zustand auf mehreren Pfaden erreichbar ist, erzeuge nur einen Suchknoten
- **Suchknoten** entsprechen **1:1** den **erreichbaren Zuständen**
- Suchbaum beschränkt, da es nur endlich viele Zustände gibt

Anmerkungen:

- einige Graphensuchalgorithmen ($\rightsquigarrow$ später) eliminieren nicht alle Duplikate sofort
- ein möglicher Grund: optimale Lösungen finden, wenn ein später gefundener Pfad zum Zustand s billiger ist als ein früher gefundener

Generischer Graphensuchalgorithmus

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  $\langle a, s' \rangle \in \text{succ}(n.state)$ :
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable
```

Generischer Graphensuchalgorithmus: Diskussion

Diskussion:

- Kommentare zur Baumsuche gelten analog
- im „reinen“ Algorithmus muss die Closed-Liste nicht notwendigerweise die Suchknoten speichern
 - es reicht, *closed* als Menge von Zuständen zu implementieren
 - fortgeschrittene Algorithmen benötigen die Knoten, daher ist hier gleich der allgemeine Fall dargestellt
- einige Varianten führen Ziel- und Duplikatstests an anderen Stellen (früher) aus
 - ↪ Vorteile? Nachteile?

Bewertung von Suchalgorithmen

Kriterien: Vollständigkeit

Vier Kriterien für Bewertung von Suchalgorithmen:

Vollständigkeit

Findet der Algorithmus garantiert eine Lösung, wenn eine existiert?
Terminiert er, wenn keine Lösung existiert?

erste Eigenschaft: **semi-vollständig**

beide Eigenschaften: **vollständig**

Kriterien: Optimalität

Vier Kriterien für Bewertung von Suchalgorithmen:

Optimalität

Sind die vom Algorithmus berechneten Lösungen immer optimal?

Kriterien: Zeitaufwand

Vier Kriterien für Bewertung von Suchalgorithmen:

Zeitaufwand

Wieviel **Zeit** benötigt der Algorithmus, um eine Lösung zu finden?

- üblicherweise **Worst-Case**-Betrachtung
- üblicherweise gemessen in **erzeugten Knoten**

oft als Funktion der folgenden Größen:

- **b : Verzweigungsgrad (branching factor)** des Zustandsraums (max. Anzahl Nachfolger eines Zustands)
- **d : Suchtiefe** (Länge des längsten Pfads im erzeugten Suchbaum)

Kriterien: Speicheraufwand

Vier Kriterien für Bewertung von Suchalgorithmen:

Speicheraufwand

Wieviel **Speicher** benötigt der Algorithmus, um eine Lösung zu finden?

- üblicherweise **Worst-Case**-Betrachtung
- üblicherweise gemessen in **gespeicherten Knoten**

oft als Funktion der folgenden Größen:

- **b : Verzweigungsgrad (branching factor)** des Zustandsraums (max. Anzahl Nachfolger eines Zustands)
- **d : Suchtiefe** (Länge des längsten Pfads im erzeugten Suchbaum)

Analyse der generischen Suchalgorithmen

Generischer Baumsuchalgorithmus

- Ist er vollständig? Ist er semi-vollständig?
- Ist er optimal?
- Was ist der Zeitaufwand im worst case?
- Was ist der Speicheraufwand im worst case?

Generischer Graphensuchalgorithmus

- Ist er vollständig? Ist er semi-vollständig?
- Ist er optimal?
- Was ist der Zeitaufwand im worst case?
- Was ist der Speicheraufwand im worst case?

Zusammenfassung

Zusammenfassung (1)

Baumsuche:

- Suchknoten entsprechen 1:1 Pfaden vom Anfangszustand

Graphensuche:

- Suchknoten entsprechen 1:1 erreichbaren Zuständen
- $\rightsquigarrow$ **Duplikateliminierung**

generische Verfahren mit vielen möglichen Varianten

Zusammenfassung (2)

Bewertung von Algorithmen:

- Vollständigkeit und Semi-Vollständigkeit
- Optimalität
- Zeit- und Speicheraufwand