EinfKI Zusammenfassung

Florian Forster

30. Juli 2006

1 Einleitung

- Schwache KI-Hypothese

  Denken ist *auch* Rechenprozesse

- Starke KI-Hypothese

  Denken ist *nur* Rechenprozesse

2 Wissen

2.1 Wissenstypen

- analytisch vs. synthetisch
- empirisch vs. theoretisch
- deklarativ vs. prozedural
- explizit vs. implizit
- qualitativ vs. quantitativ

2.2 Wissensbasierte Systeme

Ein wissensbasiertes System ist ein intelligentes Informationssystem, in dem Wissen mit Methoden der *Wissensrepräsentation* und *Wissensmodellierung* abgebildet und nutzbar gemacht wird.

- Regelbasierte Systeme
- Expertensysteme
- Software-Agenten

2.3 Faktenwissen

ggf. mit sekundär- und tertiär-Wissen, a. k. a. „Meta Daten“

- Darstellung: Tabellen, Bäume, Vektoren; ggf. via Meta-/Sekundärinformation
- Invertierte Datei (wie bei Volltextsuche)
- Klassifikation anhand von Merkmalen, z. B. „Thesaurus“

2.4 Unsicheres Wissen

An die Stelle von Tatsachen treten Möglichkeiten mit Wahrscheinlichkeiten.

$$\text{Entscheidung} = \text{Wahrscheinlichkeit} + {}^{1}\text{Nutzen} \tag{1}$$

Besonders wichtig sind bedingte Wahrscheinlichkeiten und in diesem Zusammenhang die Formel von Bayes und **Bayes-Netze**.

Bei Bayes-Netzen ist die Ausgangswahrscheinlichkeit $P(out)$ gegeben durch:

$$P(out) = P(this|in_1 \wedge in_2 \wedge \ldots \wedge in_n) \tag{2}$$

3 Agenten

- hat Sensoren und Aktoren mit der die Umwelt wahrgenommen und veränder werden kann
- führt die Aktion aus, die (erwartungsgemäß) seine Performanz maximiert
- ist *autonom*, d. h. die Aktionsauswahl beruht auf eigenen Erfahrungen
- Grundlegende Modelle
  - Einfacher Agent
  - Planender Agent
  - Beobachtender Agent
  - Bewertender Agent

4 Suchen

	Breitensuche	Tiefensuche	„Iterative Deepening“	Bidirektional
Aufwand (Zeit)	b^d	b^m	b^d	$b^{\frac{d}{2}}$
Aufwand (Speicher)	b^d	$b \cdot m$	$b \cdot d$	$b^{\frac{d}{2}}$
Optimal	Ja	Nein	Ja	Ja
Vollständig	Ja	Nein	Ja	Ja

- Breitensuche
  - Vollständig

[1] Sollte das nicht eher eine Multiplikation sein? -octo

  - Optimal
  - Verbraucht viel Speicher ($O(b^d)$)

- Tiefensuche
  - **nicht** Vollständig
    Wenn der Algorithmuss in einen Zyklus läuft wird er nie wieder „frei kommen“ und keine Lösung finden. Auch bei unendlichen Graphen gibt es Probleme.
  - **nicht** Optimal
    Ich glaube ein Algorithmus, der nicht Vollständig ist, kann gar nicht optimal sein. Sicher bin ich mir aber nicht. Jedenfalls ist die **erste** Gefundene Lösung nicht zwangsläufig die Beste.
  - Verbraucht wenig Speicher

- „Iterative Deepening“
  - Vollständig
  - Optimal
  - Wenig(er) Speicherverbrauch ($O(b \cdot d)$)
  - Vereint die Vorteile von Breiten- und Tiefensuche

4.1 Heuristische Suchverfahren

- Greedy Suche
  - a. k. a. „Best-First-Search“
  - Auswahl des nächsten Knotens, der expandiert wird, anhand einer Bewertungsfunktion: Es wird der Knoten expandiert, der am „billigsten“ erreichbar ist.
  - **nicht** vollständig und auch nicht optimal

- Dijkstra Algorithmus
  - Greedy Suche mit dynamischer Programmierung
  - Bewertungsfunktion ist die Summe der Kantengewichte

- A* Algorithmus
  - Erweitert die Greedy Suche um eine Heuristik
  - Wenn die Heuristik „zuverlässig“ ist (also die tatsächlichen Kosten nie überschätzt), dann kann Vollständigkeit und Optimalität gezeigt werden
  - In der Regel ist der euklidische Abstand (also zum Beispiel die Luftlinie) eine gute Heuristik
  - Varianten: „Iterative Deepening A* Search“, „Simplified Memory Bounded A*“

5 Planen

Planen wird abgebildet auf (konstruktives) Beweisen, also dem Suchen nach eine Lösung („Planen als Inferenz“).

Ganz wichtig: „**Reifikation**“ von Prädikaten.

5.1 STRIPS

Vorwärts- und Rückwärtsphase: In der **Vorwärtsphase** werden zu Vorbedingungen Aktionen hinzugefügt, die wiederum Nachbedingungen definieren, etc. Dabei werden „exklusive“ (also sich gegenseitig ausschließende) Aktionen und Bedingungen markiert.[2] „Alte“ Vorbedingungen werden via „NOOP“-Statements in die Liste der Nachbedingungen übernommen, so dass sowohl die Aktionen als auch die Bedingungen monoton steigen.

In der **Rückwärtsphase** wird der Plangraph rückwärts nach einem (gültigen) Plan durchsucht, der die erwünschte Schlusssituation herstellt. Für diese (Graph-)Suche kann zum Beispiel der A* Algorithmus verwendet werden.

Probleme:

- Frame-Problem
- Qualifikations-Problem
- Schneeball-Problem

5.2 Situationskalkül

Prädikate werden durch „**Reifikation**“ zu Objekten, es ist also möglich darüber zu quantifizieren. So wird die „Logik 2. Stufe“ umgangen.

6 Schließen

7 Robotik

Nix besonderes. Im Endeffekt wird nochmal alles wiederholt.

[2] Das kann recht kompliziert sein.