Wissensbasierte Suche

Jürgen Dorn

Inhalt

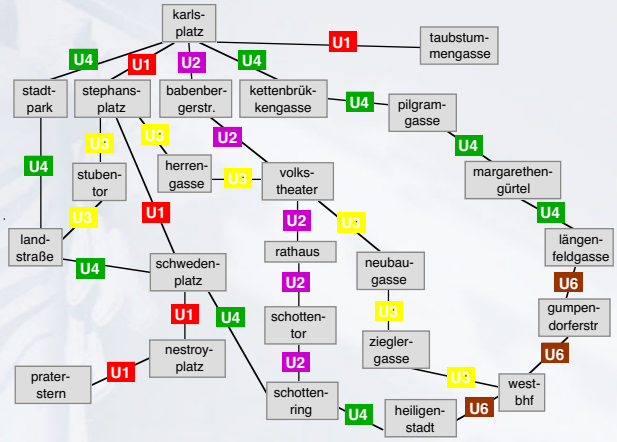
- uninformierte Suche
- wissensbasierte Suche
  - A* und IDA* Algorithmus
  - Suche in Und/Oder-Graphen

Technische Universität Wien

Suche

- Suche in (expliziten oder impliziten) Graphen
- Anwendung z.B. Wegsuche
  - Punkt bzw. Knoten (z.B. ein Ort)
  - Kante (direkte Verbindung zwischen zwei Orten)
  - Verzweigungsfaktor (wieviele Wege gehen von einem Punkt weg)
  - Kanten können Kosten zugeordnet sein (Dauer einer Fahrt, Länge der Strecke, Fahrpreis, ..).
- Probleme
  - finde Weg
  - finde kostengünstigsten Weg zwischen zwei Punkten

U-Bahn Wegegraph



Ein allgemeiner Suchalgorithmus

- Start- und Zielpunkt: p_s, p_z
- Suchgraph: Graph und Suchbaum: Tree
- bereits untersuchte Punkte: Closed
- als nächstes zu untersuchende Punkte: Open
- Sortieren von Open: heuristic_sort

```
function search(ps, pz, Path) : boolean;
begin
   Graph:=ps; Tree:= ps;
   Open := ps; Closed := ∅; pi := ps;
   while pz ∉ Open and Open ≠ ∅ do
     pi := first(Open);
     for all pj := suc(pi) do
       Graph := add(Graph, edge(pi, pj));
       if not pj in Open and not pj in Closed
       then
           Tree := add(Tree, edge(pi, pj));
           Open := Open + pj
       end if
     end for all;
     Open := Open - pi;
     Closed := Closed + pi;
     heuristic_sort(Open);
   end while;
```

```
/* Ausgabe des Weges */
IF pi = pz then
      Path := pz;
      pi := pz;
      repeat
        pi = pre(pi, Tree);
        Path := pi + Path
      until pi = ps;
      return search := true
    else search := false
   end if
end function search;
```


uninformierte Suchstrategien

- Tiefensuche und Backtracking
  - tiefere Punkte nach vorn in Open Liste
  - möglicherweise lange Suchzeit
- Breitensuche
  - höhere Punkte nach vorn in Open Liste
  - kürzester Weg (Anzahl der Kanten), aber großer Speicherbedarf
- Suche mit iterativer Vertiefung
  - erster Schritt, Suche mit Backtracking bis zur Tiefe 1
  - die Suchtiefe wird dann beim nächsten Versuch um 1 erhöht
  - wiederholen der Backtrackingsuche bis Weg gefunden
  - Kombination der Vorteile von Breitensuche und Backtracking
  - kürzester Weg wird gefunden und Speicheraufwand wird verringert

informierte heuristische Suchstrategien

- wissensbasiert, da zusätzliches Wissen die Effizienz des Suchverfahrens verbessert
- die Kosten werden optimistisch geschätzt
- Punkte, für die kleinere Kosten geschätzt werden, stehen weiter vorne in der Liste „Open"
- die Güte eines Weges vom Startpunkt p_s zu einem Zielpunkt p_z über einen Zwischenpunkt wird geschätzt
- Annahme:
  Funktion mit tatsächlichen Kosten für eine Kante $k(p_i, p_j)$

Schätzfunktion

- $h^*(p)$ sind die optimalen (theoretischen) Kosten von p nach p_z
- $g^*(p)$ sind die optimalen (theoretischen) Kosten von p_s nach p
- $f^*(p) = g^*(p) + h^*(p)$
- die Kosten $g^*(p)$ sind aber unbekannt
- deswegen wird dafür die Schätzfunktionen eingeführt
- $h(p)$ ist die Schätzfunktion für die Kosten von p nach p_z
- $g(p)$ ist die Schätzfunktion für die Kosten von p_s nach p
- $f(p) = g(p) + h(p)$

Algorithmus A*

- Bedingung $0 < h(p) \leq h^*(p)$, optimistische Schätzung
- $h(p)$ ist monoton, wenn $h(p_i) - h(p_j) \leq \text{kosten}(p_i, p_j)$
- ein Suchalgorithmus ist zulässig, wenn zugesichert ist, dass er einen Weg findet, soweit dieser existiert
- A* ist zulässig, wenn der Graph endlich und $h(p)$ monoton ist
- die Auswahl von $h(p)$ ist entscheidend für die Effektivität der heuristischen Suche
- $h(p) = 0$ und einheitliche Kosten der Kanten -> Breitensuche
- Komplexität im Extremfall wie Breitensuche

Iterative Deepening A* (IDA*)

- Kombination von Algorithmus A* und Backtracking
- Backtracking benutzt die geschätzten Kosten als Tiefenschranke
- die geschätzten Kosten zwischen p_s und p_z sind Tiefenschranke
- mit Backtracking nach Lösung suchen
- wird keine Lösung gefunden, erhöhen der Tiefenschranke auf den minimalen Wert, der beim Backtracking zum Abbruch führte

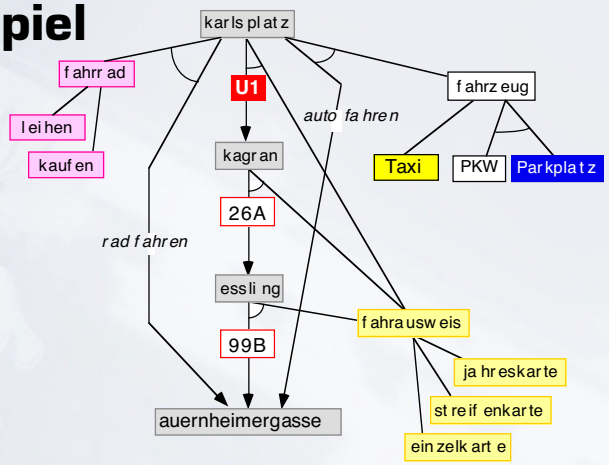
Iterative Deepening A* (IDA*)

- Vorteil: linearer Speicheraufwand im Gegensatz zum exponentiellem Aufwand bei A*
- der Zeitaufwand ist theoretisch größer als bei A*
- praktisch wurde jedoch nachgewiesen, dass der Zeitaufwand nicht viel größer ist

Und/Oder Graphen

- bisher vorgestellte Graphen sind Oder-Graphen
  - wenn verschiedene Bedingungen gleichzeitig erfüllt sein sollen, damit eine Regel angewendet werden kann, dann müssen zur Repräsentation Und/Oder-Graphen benutzt werden
- Und/Oder-Graphen sind Hypergraphen
  - es existieren Kanten zwischen Mengen von Punkten
  - Lösungen sind nicht mehr Lösungswege sondern Lösungsgraphen
  - das Ziel kann eine Menge von Punkten sein
  - der Algorithmus benutzt einen vielversprechendsten Lösungsgraphen GP
  - AO* Algorithmus

Beispiel



Entscheidungen bei Zielkonflikten

- Suche in Nullsummenspielen
- der Gewinn des einen Spielers ist gleich hoch wie der Verlust des anderen Spielers (z.B. Schach)
- wenn die Gegner abwechselnd eine Aktion ausführen, muss Spieler A immer damit rechnen, dass der Gegenspieler B, die für Spieler A ungünstigste Alternative wählt
- der so genannte Spielbaum wird wie ein Und/Oder-Graph repräsentiert

Entscheidungen bei Zielkonflikten

- die Suche wird jedoch anders durchgeführt
- Expandierung bis zum Suchhorizont
- Bewertung der Blätter
- Heraufpropagierung mit MinMax Strategie
- möglicherweise α-β Beschneidung

Spielbaum