Einführung in die
Künstliche Intelligenz
SS09 - Prof. Dr. J. Fürnkranz

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Übungsblatt (28.04.2009)

Aufgabe 1 Staubsauger-Agent

Wir betrachten folgende Welt:

Welt: Die Welt der Staubsauger-Agenten besteht aus Räumen, die rasterförmig angelegt sind. Jeder Raum ist entweder sauber oder dreckig und das Saugen von dreckigen Räumen reinigt sie zuverlässig. Versucht der Agent die Welt zu verlassen (d.h. er läuft gegen eine Wand), so bleibt er stehen und der Berührungssensor spricht an.

Wahrnehmungen: Ein Staubsauger-Agent erhält jederzeit einen dreielementigen booleschen Wahrnehmungsvektor `sensor`. Das erste Element ist ein Berührungssensor, der bei einer Kollision *wahr* liefert. Ein Fotosensor liefert *wahr*, falls der aktuelle Raum dreckig ist und ein Infrarotsensor liefert *wahr*, falls der Agent sich in seinem Ausgangsraum befindet.

Aktionen: Ein Staubsauger-Agent hat für jeden Schritt folgende Möglichkeiten:

- `vorwärts` : geradeaus in den nächsten Raum gehen bzw. stehenbleiben, falls er vor einer Wand steht (nachdem er die Wand berührt hat)
- `drehen(`g`)` : $g \in \{90, 180, 270\}$ Grad nach rechts drehen
- `saugen` und `abschalten`

Programm: Das Programm des Agenten besteht aus einer Liste von Anweisungen der Form

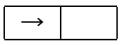
`Fakten/Wahrnehmungen --> Aktion/NeuerFakt.`

Die Anweisungen werden der Reihe nach bearbeitet. Die Regeln werden von oben nach unten durchgesehen. Wenn alle Fakten der linken Seite einer Anweisung wahr sind bzw. die erwähnten Wahrnehmungen gemacht werden, wird die rechte Seite ausgeführt, ansonsten wird mit der nächsten Regel weitergemacht. Wenn die rechte Seite eine Aktion ist, so wird diese ausgeführt und der neue Wahrnehmungsvektor berechnet (d.h. ein Schritt in der Welt gemacht). Ist die rechte Seite ein Fakt, so wird dieser Fakt als wahr gespeichert und die Regelliste von vorn durchsucht.

a) Das Beispiel aus der Vorlesung

"Wenn der aktuelle Raum schmutzig ist, dann sauge, ansonsten gehe in den anderen Raum"

für die 2-Raum-Welt



(der Staubsauger mit seiner Blickrichtung ist als Pfeil dargestellt) ist beispielsweise wie folgt:

```
sensor[2]==wahr --> sauge      % saugen, wenn der Raum dreckig ist
sensor[1]==wahr --> drehe(180) % umdrehen, falls wir vor einer Wand stehen
--> vorwärts                   % in den anderen Raum gehen
```

1. Analysieren Sie diesen reflexbasierten Agenten.
2. Was passiert, wenn man den ursprünglichen Agenten in eine 2x2-Welt setzt? Ändern Sie ihn so, daß er auch dort "vernünftig" funktioniert.
3. Was passiert in einer 3x3-Welt? Wie könnte hier ein simpler reflexbasierter Agent aussehen?

b) Die Welt des Staubsauger-Agenten bestehe nun aus einem 2×2 Schachbrett von Räumen, mit dem Raum links oben als Ausgangsposition.

1	2
3	4

Das Ziel eines Staubsauger-Agenten nach seiner Aktivierung ist die schnellste Reinigung seiner Welt, d.h. in der kleinsten Anzahl von Aktionen. Kollisionen sollen dabei möglichst vermieden werden. Um unnötige Störungen zu vermeiden, soll nur dort gesaugt werden, wo auch Dreck ist. Am Ende soll eine Selbstabschaltung an der Ausgangsposition erfolgen (hier liefere der Infrarotsensor erst nach einmaligen Verlassen *wahr*, wenn der Agent sich in seinem Ausgangsraum befindet). Die Anordnung der Räume und welche schmutzig sind sei a priori bekannt, d.h. bei geeignetem Agentenmodell kann dieses Wissen benutzt werden.

Gehen Sie für die folgenden zwei Teilaufgaben jeweils von einem *Simple reflex agent*, *Model-based reflex agent* und *Goal-based agent* aus:

1. Der Roboter schaue zunächst in Richtung Raum 2 und der einzige dreckige Raum sei Raum 3 (dieses Wissen soll ausgenutzt werden). Beschreiben Sie ein Programm, das das Ziel des Agenten am besten erfüllt.
2. Wie könnte ein Programm aussehen, das das Ziel des Agenten für beliebige 2×2 Welten am besten erfüllt.

c) Wir betrachten nun *beliebige* (aber endliche) Welten als Umgebung. Zu jedem Zeitpunkt gibt es einen Pluspunkt pro sauberen Raum, einen Minuspunkt pro schmutzigen Raum und (da dann neu gestrichen werden muß) 10 Minuspunkte pro Wand, gegen die gefahren wurde. Alle sauberen Räume werden mit gleicher Wahrscheinlichkeit nach einer gewissen Zeit wieder schmutzig.

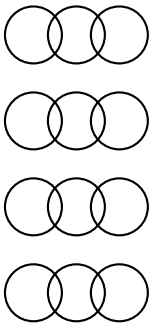
1. Konzipieren Sie einen Staubsauger-Agenten mit Gedächtnis, der sich rational in dieser Welt verhält.
2. Was ändert sich, wenn Sie zusätzlich noch einen Minuspunkt pro Aktion (*vorwärts, drehen, saugen*) erhalten?

Aufgabe 2 Modellierung

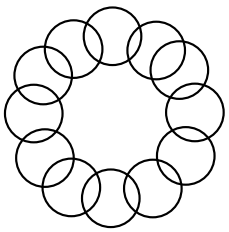
Im sogenannten **Kettenproblem** wird eine Anzahl von Ketten verschiedener Längen und Formen neu angeordnet. Das Ziel ist normalerweise die Neuanordnung mit einer minimalen Anzahl an Operationen zu erreichen.
Eine Kette kann entweder *kreisförmig* oder *linienförmig* angeordnet sein, d.h. jedes Kettenglied einer Kette ist maximal mit zwei Kettenglieder verbunden. Ein Kettenglied kann entweder *auf* oder *zu* sein. In einer Kette mit mehr als einem Kettenglied sind alle Kettenglieder zu. Die einzigen möglichen Aktionen sind das *Aufmachen* von Kettengliedern und das *Verbinden* zweier Ketten durch ein offenes Kettenglied (welches zugemacht wird). Zur Vereinfachung des Problems sei das Aufmachen einer linienförmigen Kette nur an den äußeren Kettenglieder möglich.

a) Formalisieren Sie das Kettenproblem als Suchproblem. Überlegen Sie sich zunächst eine geeignete Darstellung von Zuständen und Operatoren. Definieren Sie dann den Zustandsraum, indem Sie für jeden Operator angeben, in welchen Zuständen er anwendbar ist und wie er diese Zustände jeweils verändert.
Hinweise:
- Was passiert beim Aufmachen einer n-langen linienförmigen Kette? Und bei einer n-langen kreisförmigen Kette ?
- Es ist nicht nötig das Objekt Kettenglied zu modellieren.

b) Betrachtet wird folgendes Problem:



Ausgangszustand



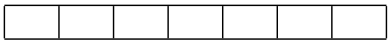
Zielzustand

Wie lassen sich der Ausgangszustand und der Zielzustand in Ihrer in a) vorgeschlagenen Repräsentation formulieren ?

c) Sei die Zahl der Operator-Anwendungen als Kosten-Funktion gegeben. Finden Sie die beste Lösung für das Problem aus b). Geben Sie den entsprechenden Pfad im Zustandsraum an.

Aufgabe 3 Suchalgorithmen

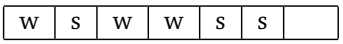
Gegeben ist ein Spielbrett mit 7 Feldern, auf dem 3 weiße und drei schwarze Steine verteilt sind (auf jedem Feld darf nur maximal ein Stein liegen):



Ein Stein darf nun wie folgt bewegt werden:

- Ein Stein darf in ein benachbartes leeres Feld gezogen werden. Kosten: 1.
- Ein Stein darf über einen oder zwei andere Steine in ein leeres Feld gezogen werden. Kosten: Anzahl der übersprungenen Steine.

Das Ziel ist nun, die Ausgangssituation



mit minimalen Kosten in die Zielkonfiguration

zu bringen, d.h. links alle weißen und rechts alle schwarzen Steine und das Loch genau in der Mitte zu haben.

a) Wenden sie die folgenden uninformierten Suchverfahren auf das Problem an, jeweils einmal mit und ohne *closed list*. Wenn Sie einen Knoten expandieren erzeugen Sie die Nachfolgeknoten jeweils so, daß zunächst der am weitesten links stehende Stein bewegt wird, dann der zweitlinkeste usw.

   1. Breadth-First
   2. Uniform-Cost
   3. Depth-First
   4. Iterative-Deepening