Einführung in die
Künstliche Intelligenz
SS 17

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Dr. J. Fürnkranz

Beispiellösung für das 1. Übungsblatt (09.05.2017)

Aufgabe 1 Agenten-Umgebung

a) Eine beispielhafte PEAS-Beschreibung für eine Internet Suchmaschine:

- Performance:
  - Aus Sicht des Users: Relevanz der Suchergebnisse, Precision, Recall, Ranking Qualität, Beantwortungszeit, Vollständigkeit der Ergebnisse, Usability
  - Aus Sicht des Betreibers: Crawlinggeschwindigkeit, speichereffiziente Indizierung
- Environment: World-Wide-Web, Hyperlinks, verschiedene Medien (Text, Bilder, Videos), verschiedene Sprachen, verschiedene HTML Versionen, inkorrekte HTML Codes (Fehlertoleranz), mehrdeutige Eingaben/Wörter, Schreibfehler, Spam
- Actuators: Weiterleitung auf relevante Seiten, indirekte Beeinflussung: Pagerank, Suchmaschinenenoptimierung, Crawler (Entscheidung welchem Link gefolgt wird), ...
- Sensors: Text-Eingabe, Click-Messung, Parsen von verschiedenen Medien im WWW

b) Eine mögliche Charakterisierung der obigen Umgebung ist:

- Fully observable vs. partially observable: *partially observable*: Eine nicht gelinkte Seite wird nicht indiziert. Neben der Problematik die Abdeckung aller Websites zu maximieren, besteht das grundsätzliche Problem, dass der semantische Inhalt einer Website/Suchanfrage nicht erfasst wird.
- Deterministic vs. stochastic: *stochastic*: Web ändert sich ständig, Suchanfragen können in unterschiedlicher Zeit bearbeitet sein.
- Episodic vs. sequential: *sequential*: Suchergebnisse beeinflussen neue Suchanfragen
- Static vs. dynamic: *dynamic*: Seiten fallen weg/kommen neu hinzu, Links werden ergänzt/gelöscht
- Discrete vs. Continuous: *continuous*
- Single agent vs. multiagent: *multiagent*: Aktionen anderer Agenten beeinflussen eigene Ergebnisse

Aufgabe 2 Agenten-Struktur

Machen Sie sich klar, worin sich die Agenten-Typen *reflex, model-based, goal-based* und *utility-based* unterscheiden:

- Im Folgenden werden nur *reflex vs. model-based*, *model-based vs. goal-based* und *goal-based vs. utility-based* Agenten verglichen.
  - *reflex vs. model-based*: der *model-based* Agent unterhält im Gegensatz zum *reflex* Agent ein Modell von der Umgebung. Dies ist hilfreich, z.B. wenn die Umgebung nur teilweise erfassbar ist, da mehrfache Sensormessungen (zeitlich verschieden oder verschiedene Sensorinformationen) eine potentiell bessere Abbildung der Umgebung ermöglichen. Die Inputs werden in das eigene Weltmodell abgebildet, welches Effekte eigener Aktionen beinhalten kann.
  - *model-based vs. goal-based*: Es werden beim *goal-based* Agenten Ziele statt condition-action rules definiert. Es ist nun vielmehr Aufgabe des Agenten geeignete Regeln zu bestimmen oder allgemeiner Aktionen durchzuführen, die zum vorgegebenen Ziel führen. Die Zukunft in Form von Auswirkungen von Aktionen wird nun mit eingezogen, d.h. Effekte eigener Aktionen werden in die Aktionsfindung mit einbezogen. Ziele können nun leicht verändert werden.
  - *goal-based vs. utility-based*: Durch die Einführung einer Bewertungsfunktion wird es dem Agenten ermöglicht Zustände zu bewerten. Im Allgemeinen wird der Agent seine Aktionen so wählen, dass er seine Bewertungsfunktion maximiert. Stimmt die Bewertungsfunktion mit dem Performance Kriterium überein, agiert der Agent rational.

Aufgabe 3 Modellierung

a) **Ketten und Zustände**

Kette: (f,l,s) — f Form ($linie/kreis$), l Länge, s Status (auf/zu)

Zustand: $\{(n,f,l,s),(n',f',l',s'),\ldots\}$ — $n \in \mathbb{N}_0^+$ Anzahl vorhandener (f,l,s)-Ketten. Ketten $n=0$ werden nicht explizit gespeichert

Operatoren

AUFMACHEN(f,l): verwandelt eine Kette (f,l,zu) in zwei Ketten $(linie,l-1,zu)$ und $(linie,1,auf)$
$(n,f,l,zu) \rightarrow (n-1,f,l,zu),(n'+1,linie,l-1,zu),(n''+1,linie,1,auf)$

VERBINDEN(l,l',f): verwandelt drei Ketten $(linie,l,zu)$, $(linie,l',zu)$ und $(linie,1,auf)$ in eine Kette $(f,l+l'+1,zu)$
$(n,linie,l,zu),(n',linie,l',zu),(n'',linie,1,auf) \rightarrow (n-1,linie,l,zu),(n'-1,linie,l',zu),(n''-1,linie,1,auf),(n'''+1,f,l+l'+1,zu)$

Merke: (f,l,s) mit $l=0$ ist eine leere Kette!

b)

Ausgangszustand: $\{(4,linie,3,zu)\}$

Zielzustand: $\{(1,kreis,12,zu)\}$

c) **Lösungspfad** (die von der Aktion betroffenen Ketten sind grün markiert)

$\{(4,linie,3,zu)\}$

$\downarrow$ AUFMACHEN$(linie,3)$

$\{(3,linie,3,zu),(1,linie,2,zu),(1,linie,1,auf)\}$

$\downarrow$ AUFMACHEN$(linie,2)$

$\{(3,linie,3,zu),(1,linie,1,zu),(2,linie,1,auf)\}$

$\downarrow$ AUFMACHEN$(linie,1)$

$\{(3,linie,3,zu),(3,linie,1,auf)\}$

$\downarrow$ VERBINDEN$(3,3,linie)$

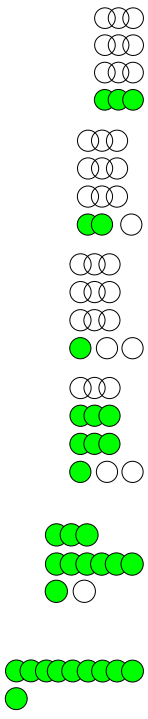
$\{(1,linie,7,zu),(1,linie,3,zu),(2,linie,1,auf)\}$

$\downarrow$ VERBINDEN$(7,3,linie)$

$\{(1,linie,11,zu),(1,linie,1,auf)\}$

$\downarrow$ VERBINDEN$(11,0,kreis)$

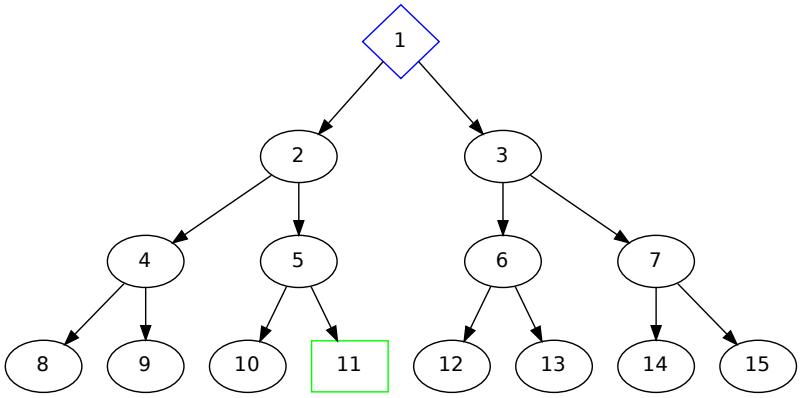
$\{(1,kreis,12,zu)\}$



Aufgabe 4 Uninformierte Suchalgorithmen

Betrachten Sie einen Zustandsraum, wo der Ausgangszustand die Nummer 1 ist und die Nachfolgerfunktion für den Zustand mit der Nummer n zwei Zustände mit den Nummern $2n$ und $2n+1$ zurückgibt.

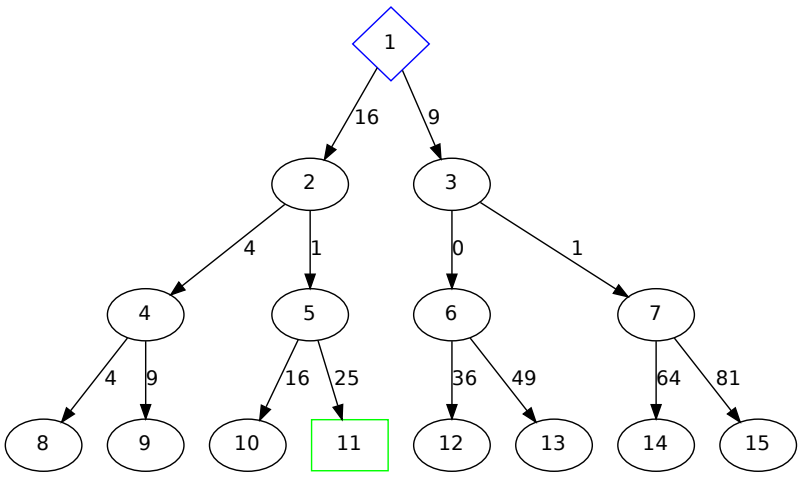
a) Zeichnen Sie den Teil des Zustandsraumes für die Zustände 1 bis 15.



b) Der Zielzustand sei 11. Geben Sie die Reihenfolge an, in der die Knoten mit den folgenden Suchalgorithmen besucht werden:

1. Breadth-First: 1,2,3,4,5,6,7,8,9,10,11
2. Depth-First: 1,2,4,8,9,5,10,11
3. Iterative-Deepening:
   - Limit 0: 1
   - Limit 1: 1,2,3
   - Limit 2: 1,2,4,5,3,6,7
   - Limit 3: 1,2,4,8,9,5,10,11

c) Die Kosten von einem unmittelbaren Vorgänger-Knoten zum Knoten mit der Nummer n zu gelangen sei $(n-6)^2$. Geben Sie die Reihenfolge an, in der die Knoten mit der *Uniform-Cost*-Search besucht werden.



Hier wird eine sog. fringe aufgebaut, indem immer der Knoten mit den geringsten Kosten expandiert wird und seine Nachfolger in die fringe einsortiert werden (die Werte in den Klammern beschreiben die Pfadkosten.):

expandiere **1**: {3(9), 2(16)} $9 < 16$, beachte: Fringe ist sortiert, daher immer ersten Knoten expandieren
expandiere **3**: {6(9), 7(10), 2(16)}
expandiere **6**: {7(10), 2(16), 12(45), 13(58)}
expandiere **7**: {2(16), 12(45), 13(58), 14(74), 15(91)}
expandiere **2**: {5(17), 4(20), 12(45), 13(58), 14(74), 15(91)}
expandiere **5**: {4(20), 10(33), 11(42), 12(45), 13(58), 14(74), 15(91)}
expandiere **4**: {8(24), 9(29), 10(33), 11(42), 12(45), 13(58), 14(74), 15(91)}
expandiere **8**: {9(29), 10(33), 11(42), 12(45), 13(58), 14(74), 15(91)}
expandiere **9**: {10(33), 11(42), 12(45), 13(58), 14(74), 15(91)}
expandiere **10**: {11(42), 12(45), 13(58), 14(74), 15(91)}
expandiere **11**: {12(45), 13(58), 14(74), 15(91)}

Die Reihenfolge ist also:

1(0),3(9),6(9), 7(10), 2(16), 5(17), 4(20), 8(24), 9(29), 10(33), 11(42)