

Einführung in die Künstliche Intelligenz

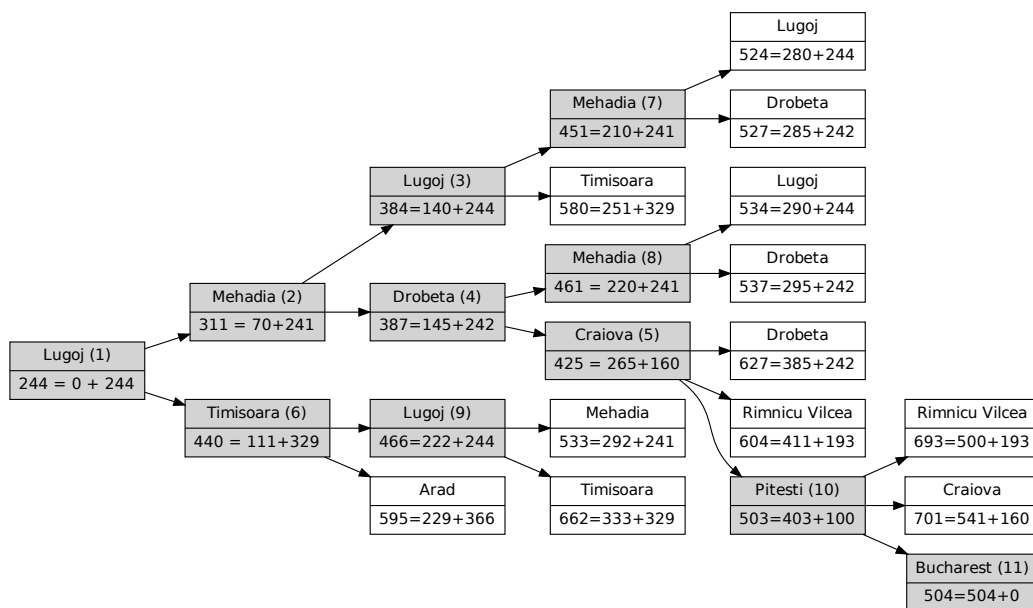
WS 10/11 - Prof. Dr. J. Fürnkranz



Beispiellösung für das 2. Übungsblatt (17.11.2010)

Aufgabe 1 Informierte Suchalgorithmen

- a) In dem folgenden Graph sind die expandierten Knoten grau markiert und die Reihenfolge, in der sie expandiert werden, ist durch die Nummerierung dargestellt.



- b) Man kann bei dieser Aufgabe den Algorithmus graphisch durchführen, da der Graph in etwa die tatsächliche geometrische Lage der Städte darstellt, so dass die angegebenen Distanzen und die Heuristiktable mit dem Graph bis auf einen Skalierungsfaktor konsistent sind.

Greedy Best-First-Search verfährt hier nach folgender Regel: Expandiere den Knoten aus der Kandidatenmenge mit der minimalen Entfernung (im Graph) zu Bucharest. So lassen sich recht schnell nahezu alle Städte überprüfen, ohne einen Blick auf die Heuristiktable zu werfen. Ein Fall, bei dem es aus dem Graph eventuell nicht ganz klar ist, tritt bei der Expansion von Mehadia auf, da Lugoj und Drobeta etwa gleich weit entfernt von Bucharest erscheinen.

Für alle Städte findet Greedy Best-First Search eine Verbindung nach Bucharest.

- c) Auch hier lässt sich das Problem schnell graphisch lösen. Für die Städte rechts von Bucharest und Giurgiu sind die gefundenen Verbindungen klar die kürzesten, da keine anderen Verbindungen (ohne mehrfache Benutzung von Kanten) existieren.

Die Verbindungen von den restlichen Städte lassen sich mithilfe kleiner Überlegungen ähnlich schnell auf Optimalität prüfen. Hilfreich ist z.B. die notwendige Bedingung, dass jede Teilverbindung einer optimalen Verbindung wieder eine optimale Verbindung sein muss. Zum Beispiel ist die Verbindung von Sibiu nach Bucharest, die Greedy Best-First Search findet (Sibiu, Fagaras, Bucharest), nicht optimal. Somit sind auch die ermittelten Verbindungen von Oradea, Zerind und Arad, die durch Sibiu verlaufen, auch nicht optimal.

Für Timisoara, Arad, Zerind, Oradea und Sibiu findet Greedy Best-First Search nicht die optimale Verbindung.

Aufgabe 2 Heuristiken

- a) Zunächst ist festzustellen, dass die Kosten für eine Aktion (Zug) hier auf 1 gesetzt sind, d.h. für alle legalen Züge ist $c(n, a, n') = 1$. Für beide Heuristiken muss also gezeigt werden: $h(n) \leq 1 + h(n')$.

Die h_{MIS} Heuristik zählt die Anzahl falsch platzierter Steine. Mit einem Zug kann sich die Anzahl der falsch platzierten Steine um maximal eins verringern, d.h. $h(n') \geq h(n) - 1$.

Die h_{MAN} Heuristik zählt die Anzahl der Felder, die überquert werden müssen, um zur Zielposition zu gelangen (summiert über alle Steine). Mit einem Zug ändert sich die entsprechende Anzahl nur für ein Stein, die nur maximal um eins verringert sein kann.

- b) Seien h, h' zwei beliebige admissible Heuristiken bezüglich der wahren Kostenfunktion h^* . Es gilt also: $h(n) \leq h^*(n)$ und $h'(n) \leq h^*(n)$ für alle n .

Sei g nun die mittels dem Maximum-Operator kombinierte Heuristik von h und h' , d.h. $g(n) := \max(h(n), h'(n))$. Es gilt offensichtlich $g(n) \leq h^*(n)$, da für beliebige n , die Heuristik g entweder $h(n)$ oder $h'(n)$ zurück liefert, welche beide per Konstruktion nicht größer als $h^*(n)$ sind.

Aufgabe 3 Lokale Suchalgorithmen

- a) Hill Climbing
b) (ähnlich zu) Hill Climbing
c) (ähnlich zu) Random Walk, eine zufällige Traversierung des Zustandsraumes
-

Aufgabe 4 Constraint Satisfaction Problem

Es sollte klar sein, dass folgende Formalisierung eine Möglichkeit und nicht die einzig Richtige darstellt.

- a) Wertebereiche der Variablen:

$$\begin{aligned} \text{domain}(A) &= \text{domain}(C) = \{0, \dots, 9\} \\ \text{domain}(B) &= \{1, \dots, 9\} \\ \text{domain}(U) &= \{0, 1\} \end{aligned}$$

Constraints:

$$\begin{aligned} A + B &= C + 10 \cdot U \\ B &= U \\ A \neq B, A \neq C, B \neq C \end{aligned}$$

- b) C selected
C = 0 consistent
B selected
B = 1 consistent
A selected
A = 0 not consistent (A != C constraint violated)
A = 1 not consistent (A != B constraint violated)
A = 2 consistent
U selected
U = 0 not consistent (B = U constraint violated)
U = 1 not consistent (A+B=C+10*U constraint violated)
U removed
A = 3 consistent
... (identischer Verlauf wie in A=2, mit den gleichen Constraint Verletzungen)
A = 9 consistent
U selected
U = 0 not consistent
U = 1 consistent
result <- C=0,B=1,A=9,U=1
-

- c) In der folgenden Darstellung des Algorithmus-Ablaufs werden nun zusätzlich die durch Forward-Checking veränderten Wertebereiche dargestellt.

```
C selected
C = 0 consistent
FC: domain(A) = {1,...,9}
B selected
B = 1 consistent
FC: domain(A) = {2,...,9}, domain(U) = {1}
A selected
A = 2 consistent (Beachte: die Zuweisung A = 0,1 kommt hier nicht mehr vor)
  FC: domain(U)={}, backtrack
A = 3 consistent
... (identischer Verlauf wie in A=2)
A = 9 consistent
  FC: nothing to change
U selected
  U = 1 consistent (U=0 kommt nicht mehr vor)
  result <- C=0,B=1,A=9,U=1
```

Es war unklar, wie Forward-Checking bei höherwertigen (higher-order) Constraints verfährt. Hier wurde ein höherwertiges Constraint erst dann für Forward-Checking herangezogen, falls sie nur noch eine nicht gesetzte Variable enthält.

Constraint Propagation bezeichnet den allgemeinen Begriff für die Übertragung von Implikationen eines Constraints einer Variablen auf andere Variablen. Zusammen mit Variablensetzungen lassen sich damit oft viele inkonsistente Variablensetzungen ausschließen. In diesem Beispiel ist der Suchraum der noch nicht belegten Variablen nach der Festlegung $C = 0$ und $B = 1$: $A \in \{2, \dots, 9\}$ und $U \in \{1\}$. Die Festlegung von $B = 1$ hat Konsequenzen auf das erste Constraint, das nun folgendermaßen aussieht: $A + 1 = 10 \cdot U$. Mit Betrachtung der möglichen Werte beider Variablen (ähnlich zu Arc Consistency) kann für A alle Werte außer 9 ausgeschlossen werden.

Auf den ersten Blick erscheint diese Überprüfung identisch mit der normalen Abarbeitung aller möglichen Werte, wie oben dargestellt. Der Nutzen wird klarer, wenn man sich vorstellt, dass das Problem aus mehr als den vier Variablen und (unabhängigen) Constraints besteht, die im Anschluss an A getestet werden.