Universität des Saarlandes
Fachrichtung 6.2 – Informatik
J. Siekmann, S. Autexier, K. Fischer, E. Melis, C. Zinn
M. Schiller, E. Schulz, R.Vollmann

3rd Theoretical Assignment in Artificial Intelligence (WS 2006/2007)

Solutions

Note: You need not hand these exercises in, and they are not graded. But bring along your solutions to the tutorial. Impress your tutors by presenting your favourite exercise to the class.

Exercise 3.1

Marvin Minsky claims to stem from Charles the Great. Which way is easier to check the claim: to show that Charles the Great is an ancestor of Minsky or to show that Minsky is a descendant of Charles the Great? Explain.

Solution:

Every person has only two parents, but can have arbitrarily many children. Therefore, backward search branches with exactly 2, but forward search may branch with much more than 2. On the other hand, forward search terminates for sure, the latest when we reach contemporary descendants of Charles the Great who don't have children yet. Backward search may not terminate at all (unless we all descend from Adam and Eve, if you see it from a religious perspective). Hence, the best search procedure would probably be backward with keeping track of the time when the person under scrutiny lived, such that the search in one branch is stopped as soon as the birthyear of Charles the Great has been passed. If it turned out Charles the Great had very few descendents, then forward search would be better.

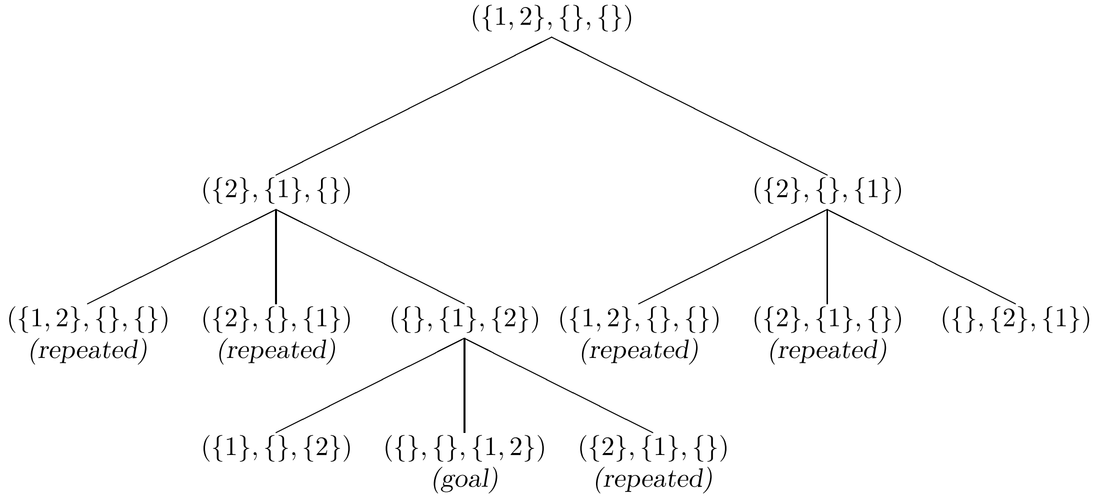
Exercise 3.2

Consider the problem that you have 3 pegs, A B and C, and a number N of discs, all with different diameters. A larger disc cannot be placed on top of a smaller disc. The discs are initially placed on peg A, and the goal is to move them all on peg C, in a minimal number of moves.

1. Formulate this as a search problem. Specify the representation of the states (parameterized over N), operators, goal, start state and cost function.
2. What is the maximal branching factor?
3. Let N=2. Use breadth-first-search to find a solution and draw the search tree.
4. Find a (non-trivial) admissible and consistent heuristic for the problem, and show that it is admissible and consistent.
5. Let N=2. Apply A* with that heuristic to the problem, and draw the search tree.

Solution:

1. *Identify each disc with a number from 1 to N in the order of their size. Each state can be represented as a triple of sets, where each set represents the discs on the pegs A, B and C in that order. The operators move the smallest disc from one peg to another, provided there is no smaller disc on the target peg. The start state is* $(\{1, 2, 3, ..., N\}, \{\}, \{\})$*, the goal state is ({},{},{1,2,3,...,N}). The cost for each operator application is one (or can be any fixed constant).*

2. *We maximally have three pegs on which there can be a disc. We may chose one disc and then chose among the two other pegs onto which to move the disc (in principle, we may also include the original peg, which would make it three choices). So we get a maximal branching factor of six (or nine, if we consider the useless option of moving the disc onto its original peg).*
3. 

   The algorithm unfolds the nodes level by level, until it finds a goal state (here it is in the 3rd level).
4. *We chose as heuristic the number of discs that are not on the target disc, e.g. at the start-state there are N discs on the first peg, so the heuristic gives us the value N for this state.*

   We have to prove that this heuristic is admissible and consistent. It suffices to prove that it is consistent, because consistency implies admissibility.

   Given a state n and n' its successor-state which results from the action a. The cost of each action a is 1. We have to prove that $h(n) \leq c(n, a, n') + h(n')$ which is in our case $h(n) \leq 1 + h(n')$.

   Let's consider all the possible actions:

   we move a disc from peg 1 to peg 2: *The value of the heuristic doesn't change,* $h(n') = h(n)$, *and we get* $h(n') + 1 \geq h(n') = h(n)$

   we move a disc from peg 1 to peg 3: *The value of the heuristic decreases,* $h(n') = h(n) - 1$, *and we get* $h(n') + 1 = h(n) - 1 + 1 = h(n) \geq h(n)$

   we move a disc from peg 2 to peg 1: *The value of the heuristic doesn't change,* $h(n') = h(n)$, *and we get* $h(n') + 1 \geq h(n') = h(n)$

   we move a disc from peg 2 to peg 3: *The value of the heuristic decreases,* $h(n') = h(n) - 1$, *and we get* $h(n') + 1 = h(n) - 1 + 1 = h(n) \geq h(n)$

   we move a disc from peg 3 to peg 1: *The value of the heuristic increases,* $h(n') = h(n) + 1$, *and we get* $h(n') + 1 = h(n) + 1 + 1 \geq h(n)$

   we move a disc from peg 3 to peg 2: *The value of the heuristic increases,* $h(n') = h(n) + 1$, *and we get* $h(n') + 1 = h(n) + 1 + 1 \geq h(n)$
5. *The search tree is displayed in Figure 1. The order in which the nodes are expanded is the following: node 1, node 2, node 4, node 3, node 7, node 13. Since node 13 is the goal, the algorithm stops.*

   When considering the tree for the BFS search, the A* tree looks just as big. Why is this a bad example to demonstrate the superiority of A* over BFS?

   First of all, the problem space is very small (otherwise, the exercise would be unmanagable as a homework assignment). We did not try to expand repeated states again, which made it easy for BFS to find the solution (since there remained only few options). For example, on the left branch of the BFS search tree, on the second level there remained only one useful option (which by chance lead to the goal). In order to demonstrate the usefulness *of A*, we should have rather taken a problem with lots of different options. Another problem concerns the effectiveness of the heuristic we have chosen. If we place small discs on peg C, we will be rewarded by the heuristic, in the same way in which we can put big discs on peg C. But in reality, putting the biggest disc on peg C is more valuable, because we will never need to move it away again (in contrast to the small discs). Our heuristic ignored that.*

Exercise 3.3

Prove each of the following statements:

1. Breadth-first search is a special case of uniform-cost search.
2. Breadth-first search, depth-first search, and uniform-cost search are special cases of greedy search.
3. Uniform-cost search is a special case of A* search.

Solution:

1. *BFS is simulated by uniform-cost search when all costs are set to the same constant.*
2. - *BFS is simulated by best-first search, where the heuristic function h(n) = distance from n to the root node.*
   - *DFS is simulated by best-first search where the heuristic function h() = - distance from n to the root node.*
   - *Uniform cost search is simulated by best-first search where h(n) = sum of the costs on the path from n to the root node.*
3. *Set the heuristic function to h(n)=0.*

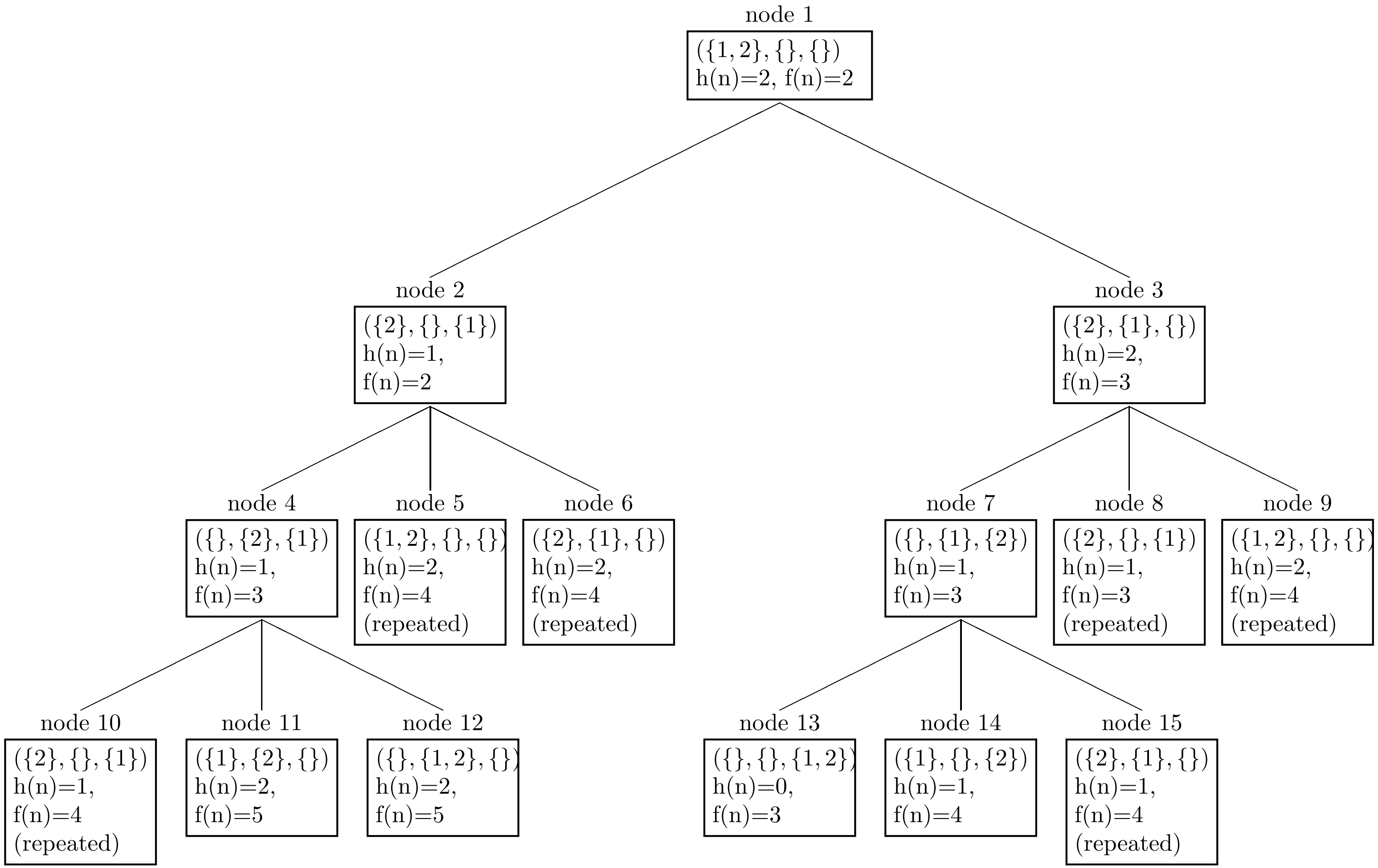
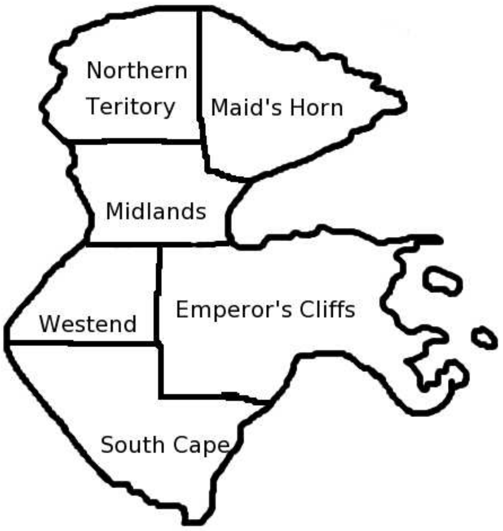


Figure 1: Search Tree with A* Search

Exercise 3.4



On the picture above, you see a map showing the six regions of an imaginary country. Now consider the problem that you want to colour the map with three colors, `red`, `blue` and `green`, such that no adjacent regions have the same color.

1. Explain why this problem is suitable for an *iterative improvement* algorithm, rather than the other search algorithms from the lecture.
2. Give a suitable representation for the states of the colouring problem.
3. Give the number of possible states of the problem (including those states that do not solve the problem). What can you say about the size of the search tree?
4. Assume the start configuration where all regions are coloured `red`. Use the hillclimbing algorithm to find a solution. Indicate each step. Hint: As an evaluation function for states, use the number of adjacent regions that have the same colour, and minimise this value during hillclimbing.
5. Assume the start configuration where Northern Teritory, Maid's Horn, Westend and Emperor's Cliffs are coloured `green`, Midlands is coloured `blue` and South Cape is coloured `red`. Again, use the algorithm and explain what is happening.

Solution:

1. *Here, only the goal state is important, namely the colouring of the map, whereas the particular search path is not of importance.*
2. *We can represent each state as a six-tuple,* $(C_{NT}, C_{MH}, C_{ML}, C_{WE}, C_{EC}, C_{SC})$, *where* $C_i \in \{R, G, B\}$ *for each of the six components. Each* C_i *represents one of the regions of the imaginary country, and* R, G *and* B *represent the colours red, green and blue. For example, the starting state can be represented as* (R, R, R, R, R, R).
3. *There are* $3^6 = 729$ *distinct states. However, if we construct a search tree without checking for repeated states, the problem allows an infinite search tree.*
4. *We begin with state (R,R,R,R,R,R), where we have 8 pairs of adjacent regions with the same colour. For a given state* $(C_{NT}, C_{MH}, C_{ML}, C_{WE}, C_{EC}, C_{SC})$, *we obtain a neighbour state* $(C'_{NT}, C'_{MH}, C'_{ML}, C'_{WE}, C'_{EC}, C'_{SC})$ *by changing the colour of one region r, i.e.* $C_r \neq C'_r$, *and leaving all the other colours the same, i.e.* $C_i = C'_i$ *for all other regions i.*
   We have written a program in Lisp that uses hillclimbing to solve our problem (in the hope that it will produce less errors than a human). It produces the following run:

```
 USER(15):  USER(15):  (hillclimbing '(R R R R R R))
State (R R R R R R) Conflicts 8
State (R R G R R R) Conflicts 4
State (R R G B R R) Conflicts 2
State (B R G B R R) Conflicts 1
State (B R G B R G) Conflicts 0
```

The assignment did not specify how – in the case that a state has several best neighbours – the algorithm would chose one of them.

5.

```
USER(16):  (hillclimbing '(G G B G G R))
State (G G B G G R) Conflicts 2
State (R G B G G R) Conflicts 1
State (R G B B G R) Conflicts 1
State (R G R B G R) Conflicts 1
State (B G R B G R) Conflicts 0
```

We observe a "shoulder", where the value of the state does not improve for a long time. So we have to allow the algorithm to move "sidewards". In the case of our toy problem, we are lucky, and after a long walk over the "shoulder", we can improve our state again. However, depending on the problem we might be less lucky, and the hillclimbing algorithm possibly gets stuck in a local optimum and does not find a solution, even if one exists.