Informatics 2D. Tutorial 2
Reasoning Agents: Agents and Propositional Inference

Kobby Nuamah, Michael Rovatsos

Week 3

1 Adversarial Search

This exercise was taken from R&N Chapter 5.

Consider the two-player game shown in Figure 1

1. Draw the complete game tree, using the following conventions:
   - Write each state as (S_A, S_B) where S_A and S_B denote token locations
   - Put each terminal state in square boxes and write its game value in a circle
   - Put *loop states* (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a "?" in a circle.
2. Now mark each node with its backed-up minimax value (also in a circle). You will have to think of a way to assign values to the loop states.
3. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to item (2) above. Does your modified algorithm give optimal decisions for all games with loops?

1.1 Tutor's Guide: Adversarial Search

1. The game tree, complete with annotations is shown in Figure 2
2. The "?" nodes can be assigned a value of 0, so choosing the loop state will not benefit either player. If there is a win for a player from a loop state then they will chose the moves that lead to the win, rather than end up in the loop state.

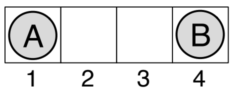


Figure 1: The starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move their token to an open adjacent space in either direction. If the opponent occupies an adjacent space, then the player may jump over the opponent to the next available space. For example, if A is on 3 and B is on 2, then A may move back to 1. The game ends when one player reaches the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is $+1$; if player B reaches space 1 first the value of the game to A is -1.

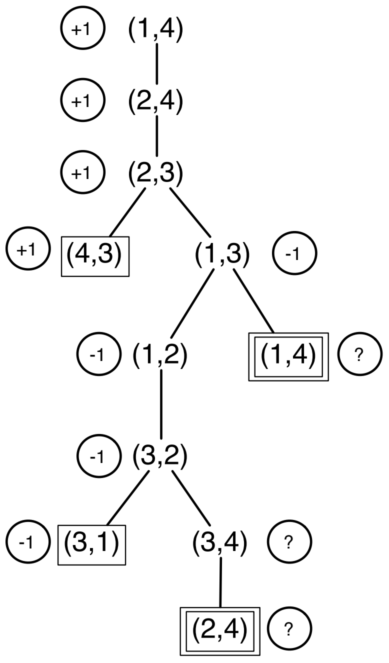


Figure 2: The game tree for the four-square game in Exercise 6.3. Terminal states are in single boxes, loop states in double boxes. Each state is annotated with its minimax value in a circle.

3. Standard minimax is depth-first and would go into an infinite loop. It can be fixed by comparing the current state against the stack; and if the state is repeated, then return a “?” value. Propagation of “?” is handled as above. Although it works in this case, it does not always work. For example, it is not clear how to compare “?” with a drawn position: a drawn position is not better or worse for either player, but might at least bring an end to the game, so assigning 0 to “?” states will not work there.

2 Informed Search

We saw in the lectures a graph representing the road map of part of Romania, see Figure 4. The cost of a path is the distance via the road, as given on the graph. We also have a table of straight-line distances from each town to Bucharest.

(a) Show that using greedy best-first search with the straight-line heuristic function, h_{SLD}, does not give an optimal solution when looking for a path from Arad to Bucharest.

(b) Suppose that you have the following straight-line distances from Fagaras to: Neamt 140km, Iasi 175km, Vaslui 175km, Urziceni 180km, Hirsova 230km, Giurgiu 220km, Pitesti 50km, Rimnicu Vilcea 50km, Craiova 180km, Sibiu 60km. What happens when you try to use greedy best-first search to find a path from Iasi to Fagaras?

(c) We can use A^* search in this problem; h_{SLD} is an admissible heuristic that can be combined with the actual distance of the path so far to get a new heuristic f. Show that f finds an optimal solution in part (a) and solves the problem in part (b).

Tutor's Guide

For both greedy best-first search and $A^* search$ you should go through (at least some of) the search tree, and show which nodes are expanded. Below are the search graphs for greedy best-first search and A^* search for the Arad to Bucharest problem (taken from R&N).

(a) Greedy best-first search should find the route Arad $\rightarrow$ Sibiu $\rightarrow$ Fagaras $\rightarrow$ Bucharest, which is 450km. But the route Arad $\rightarrow$ Sibiu $\rightarrow$ Rimnicu Vilcea $\rightarrow$ Pitesti $\rightarrow$ Bucharest is 418km.

(b) Greedy best-first search should loop between Neamt and Iasi, since the heuristic value of Neamt is less than the heuristic value of Vaslui.

(c) A^* search should find the route Arad $\rightarrow$ Sibiu $\rightarrow$ Rimnicu Vilcea $\rightarrow$ Pitesti $\rightarrow$ Bucharest for part (a), and the route Iasi $\rightarrow$ Vaslui $\rightarrow$ Urziceni $\rightarrow$ Bucharest $\rightarrow$ Fagaras for part (b).

For (b) the A^* values are (for the initial transition):
f(Neamt) = 140 + 87 = 227
f(Iasi) = 175 = 175
f(Vaslui) = 175 + 92 = 267
f(Urziceni) = 180 + 234 = 414
f(Bucharest) = 176 + 319 = 495
f(Hirsova) = 230 + 332 = 562
f(Giurgui) = 220 + 319 = 539
f(Pitesti) = 50 + 420 = 470
f(Fagaras) = 0 + 530 = 530
f(Rimnicu Vilcea) = 50 + 517 = 567
f(Craiova) = 180 + 558 = 738

3 Heuristics

(Taken from R&N Chapter 3)

Sometimes there is no good evaluation function for a problem, but there is a good comparison method: a way to tell whether one node is better than another, without assigning numerical values to either. Show that this is enough to do a Best-First search. Is there an analog for A^*?

Tutor's Guide

If we assume the comparison function is transitive, then we can sort a list of nodes using it, and choose the node that is at the head of the list.

A^* relies on the division of the total cost estimate $f(n)$ into the cost-so-far and the cost- to-go. If we have comparison operators for each of these, then we can prefer to expand a node that is better than other nodes on both comparisons. Unfortunately, there may not be a node with these properties. In which case, the tradeoff between $g(n)$ and $h(n)$ cannot be realized without numerical values.

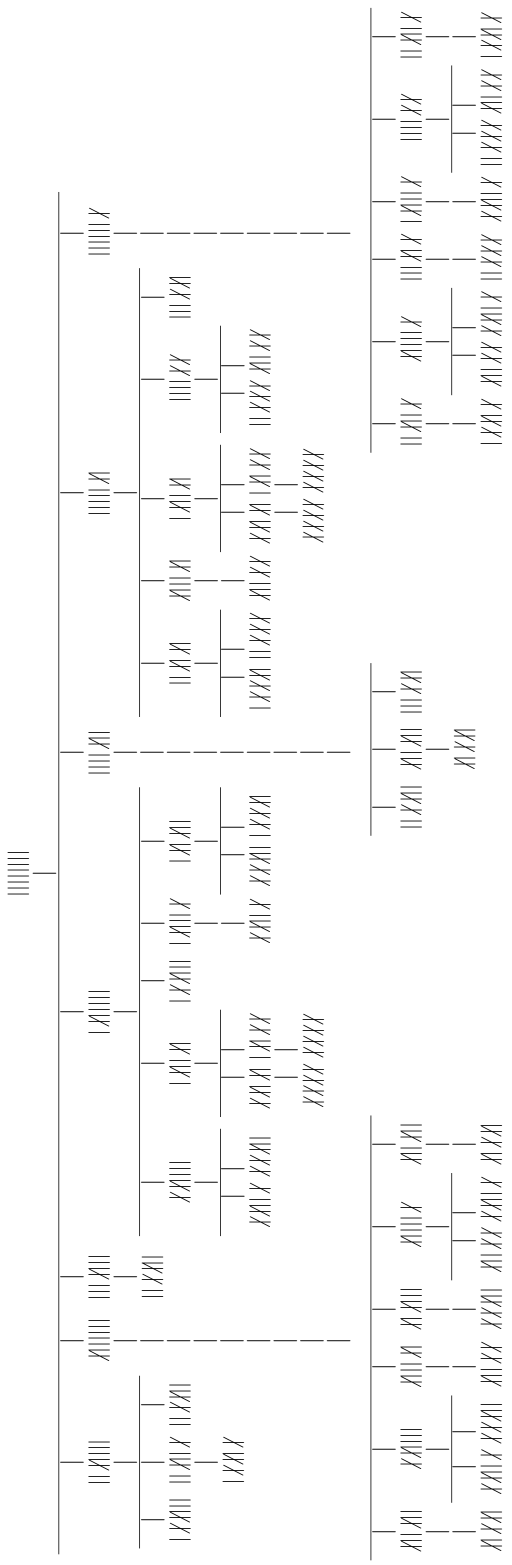
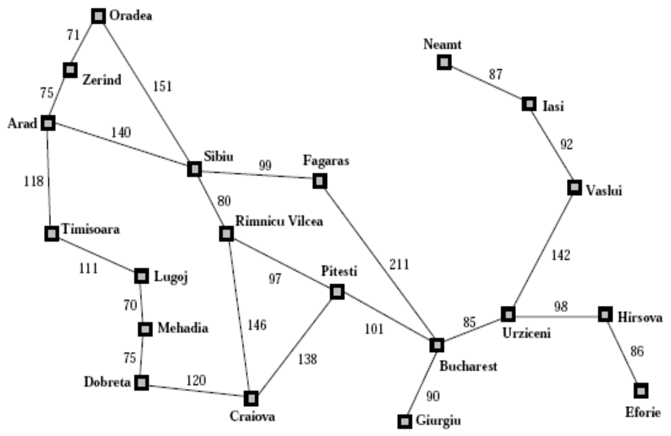


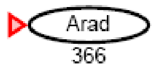
Figure 3: Search tree for the Sticks Problem



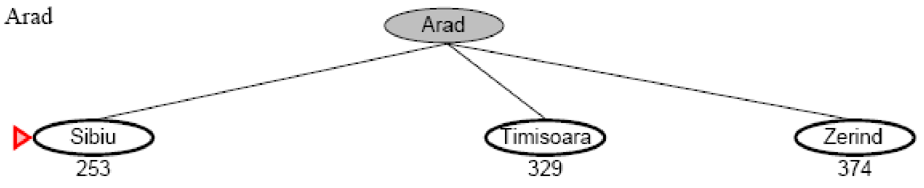
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 4: The Romania map from R&N with a table of straight-line distances to Bucharest

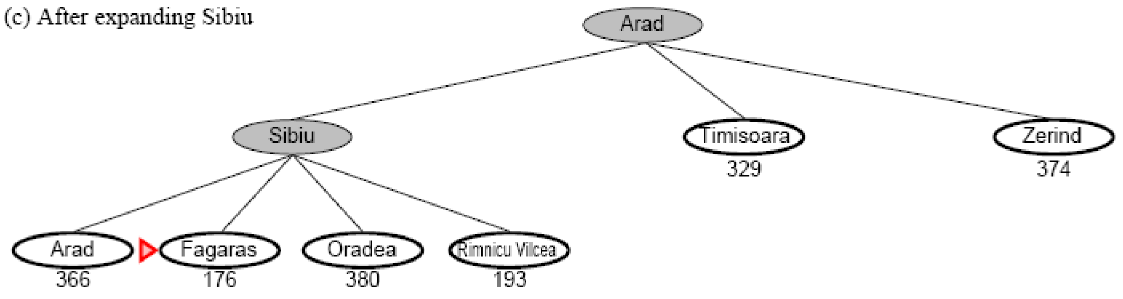
(a) The initial state



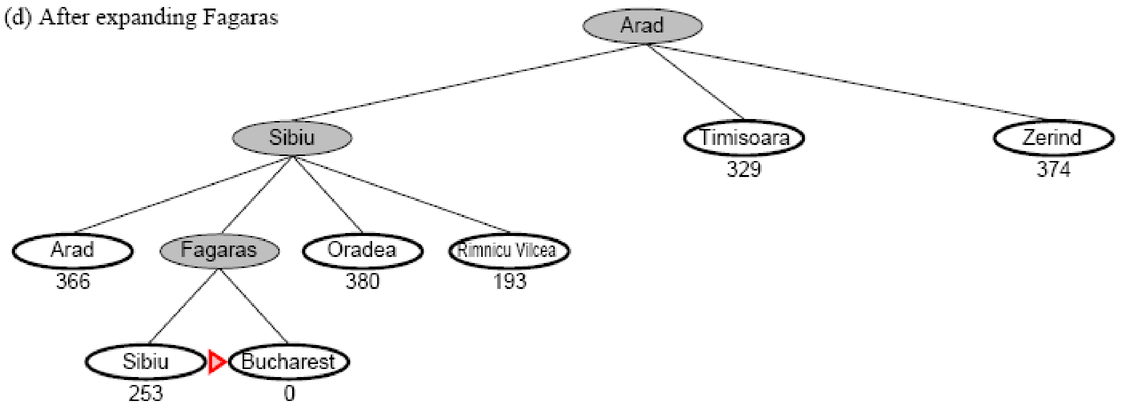
(b) After expanding Arad

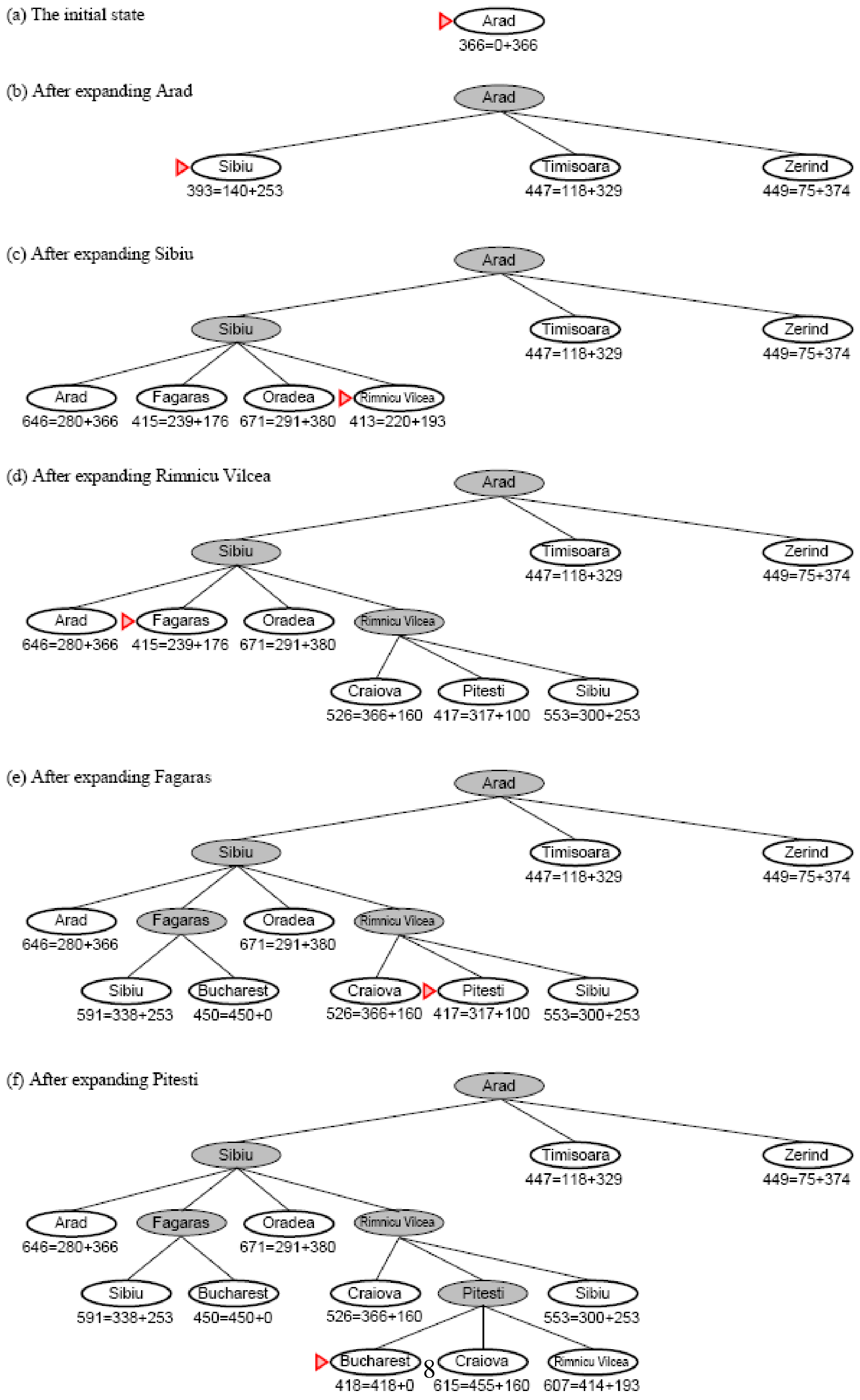


(c) After expanding Sibiu



(d) After expanding Fagaras




(a) The initial state
Arad
366=0+366
(b) After expanding Arad
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
413=220+193
(d) After expanding Rimnicu Vilcea
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
(e) After expanding Fagaras
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
(f) After expanding Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193