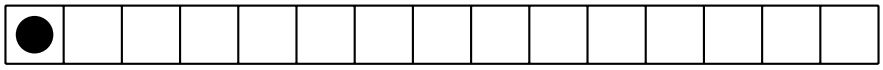
02180: Introduction to Artificial Intelligence
Exercises at week 2

Ex. 1 Consider the following two-player game **Race Game**. Initially there is a token on cell 1. The two players take turn to move the token 1, 2 or 3 cells to the right. The one who moves the token into cell 15 has won.



Play the game a few times in pairs. Reflect on the following questions:

1. What is the best strategy for the first player?
2. How did you come up with it?
3. How can we make AI to play such a game?

Ex. 2 The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. Whenever the boat goes from one side to the other, everybody has to go out before the boat can go back again. The boat can not sail without anybody in it.

1. A *state* is a description of all relevant parameters of a given problem (in this case it should include the positioning of all missionaries and cannibals with respect to the side of the river, and the position of the boat). How can the possible states of the problem be represented? What is then the initial state? Draw a diagram of the complete state space. What is the size of the state space (measured in number of states it contains)?
2. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?
3. Assume instead that there are n missionaries and n cannibals. What is then the size of the state space? You can report the size of the full state space, including the states that are not reachable without violating the constraint of cannibals not outnumbering missionaries. You can measure the size of the state space in number of distinct states. Does the size of the state space imply that the problem is difficult for a computer to solve for big values of n?
4. Assume that in addition to the n missionaries, n cannibals there are now also n boats. Does this imply that the problem is difficult for a computer to solve for big values of n?
5. Assume that in addition to the n missionaries, n cannibals and n boats there are now also n banks instead of only 2. This means that any of the persons and boats can be at any of n banks. Does this imply that the problem is difficult for a computer to solve for big values of n?

Ex. 3 Go to: https://sokoban-game.com/packs/sokogen-990602-levels/ and complete as many levels as possible, starting with Level 1.

02180: Introduction to Artificial Intelligence
Solutions to exercises at week 2
Missionaries and cannibals

The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. Whenever the boat goes from one side to the other, everybody has to go out before the boat can go back again. The boat can not sail without anybody in it.

1. A *state* is a description of all relevant parameters of a given problem (in this case it should include the positioning of all missionaries and cannibals with respect to the side of the river, and the position of the boat). How can the possible states of the problem be represented? What is then the initial state? Draw a diagram of the complete state space. What is the size of the state space (measured in number of states it contains)?

2. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

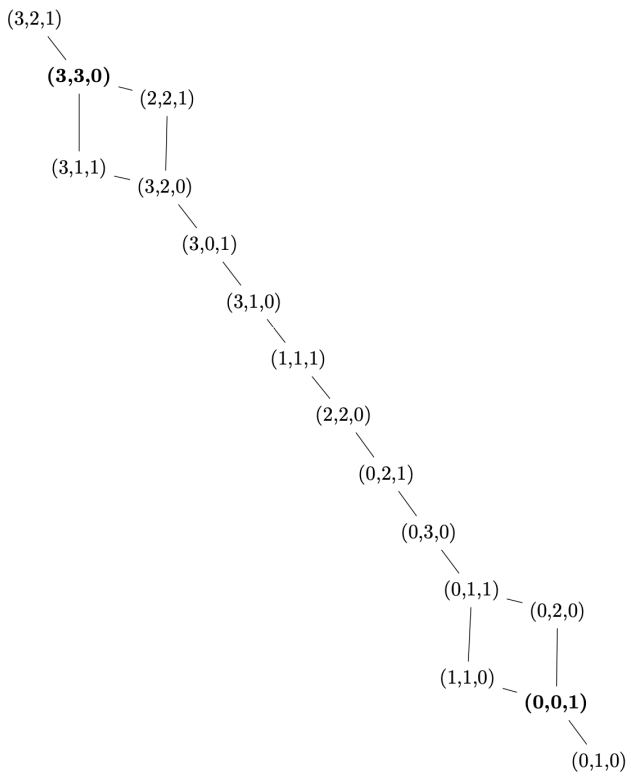
3. Assume instead that there are n missionaries and n cannibals. What is then the size of the state space? You can report the size of the full state space, including the states that are not reachable without violating the constraint of cannibals not outnumbering missionaries. You can measure the size of the state space in number of distinct states. Does the size of the state space imply that the problem is difficult for a computer to solve for big values of n?

4. Assume that in addition to the n missionaries, n cannibals there are now also n boats. Does this imply that the problem is difficult for a computer to solve for big values of n?

5. Assume that in addition to the n missionaries, n cannibals and n boats there are now also n banks instead of only 2. This means that any of the persons and boats can be at any of n banks. Does this imply that the problem is difficult for a computer to solve for big values of n?

Solution

1. In each state we only have to keep track of how many missionaries and cannibals are at each bank. Since the number of missionaries and cannibals at the right bank can be deduced from the number of missionaries and cannibals at the left, we only need to keep track of how many of each are at the left bank. The only additional thing we have to keep track of is whether the boat is at the left or right bank. Hence each state can be represented as a triple (m, c, b), where $m \in \{0, \ldots, 3\}$ is the number of missionaries at the left bank, $n \in \{0 \ldots 3\}$ is the number of cannibals at the left bank, and $b = 0$ if the boat is at the left bank, otherwise $b = 1$. Hence an upper bound on the size of the reachable part of the state space is $4 \times 4 \times 2 = 32$. The initial state can be assumed to be $(3, 3, 0)$ (everything initially at the left bank). The goal state is then $(0, 0, 1)$. The available actions are for the boat to take 1?2 people from the current bank to the opposite bank. However, we also have to make sure that the number of cannibals at any bank never outnumber the missionaries. This leads to the following state space (edge labels left out):



2. The are several reasons. One of them is that it is a bit tricky to keep track of which moves are legal. Another reason is that the solution is relatively long, 11 steps. Long solutions are not always a problem to find for humans, but it is only easy for us when we have a good heuristics to guide our search. For instance, finding a walking route from A to B on a map is usually not difficult for us, even if there are more than 11 intersections where we have to make a choice. Usually, it is sufficient in any intersection to choose the next road segment to be the one that ends up in a new intersection with minimal straight line distance to B (technically speaking, we would be doing a greedy best-first search where the heuristics is the straight line distance to B). However, for the missionary and cannibals problem, there is no simple heuristics to guide the search. Indeed, the obvious heuristics which counts the number of people not yet at the right bank is not particularly efficient, as in every second move, we have to move further away from the solution according to this heuristics (we need to move people back from the right to the left bank). Finally, for humans it is not so easy to keep track of which states have already been visited, so we easily end up in loops (a greedy best-first search with the mentioned heuristics would end up in an infinite loop unless it keeps track of the already visited states).

3. With n cannibals and n missionaries, the state can now be represented as an element of $[0, n] \times [0, n] \times \{0, 1\}$. There are $2(n + 1)^2$ such triples, which is hence the size of the (full) state space. As there is hence still only a polynomial number of states in n, it should not be too hard for a computer to solve it even for large values of n. Actually, it has been proven that there is no solution to the problem with $n > 3$, but a computer searching for a solution might of course not realise that until having explored the full reachable state space.

4. Now states can be represented as elements $(m, c, b) \in [0, n] \times [0, n] \times [0, n]$, where m is the number of missionaries at the left bank, c is the number of cannibals at the left bank, and b is the number of boats at the left bank. The size of the state space becomes $(n + 1)^3$, but this is still polynomial in n and should not be too hard for a computer.

Take the case $n = 5$. We can represent the missionaries as a list of dots:

$$\cdot\cdot\cdot\cdot\cdot$$

We can then place $n-1$ vertical bars to represent how they are distributed into the n banks. For instance:

$$\cdot|\cdot\cdot||\cdot\cdot|$$

means that there is 1 missionary on the first bank, 2 on the second, none on the third, 2 on the fourth, and none on the fifth. The number of distributions of missionaries is equal to the number of ways we can position the $n-1$ vertical bars. Each bar can be positioned in $n+1$ different ways. This gives in total $(n+1)^{(n-1)}$ different possibilities. But this is assuming that the bars are distinguishable, which they are not, so we have to divide by $n!$. So the number of possible distributions of missionaries is

$$\frac{(n+1)^{(n-1)}}{n!}.$$

We have the same number of possible distribution of cannibals and boats. This expression grows exponentially in n which can be seen by the following calculations:

$$\begin{aligned}\frac{(n+1)^{(n-1)}}{n!} &\geq \frac{1}{n}\Pi_{i=1}^{n-1}\frac{n+1}{i} \geq \frac{1}{n}\Pi_{i=1}^{n-1}\frac{n}{i} \geq \Pi_{i=2}^{n-1}\frac{n}{i} = \Pi_{i=2}^{\lfloor n/2\rfloor}\frac{n}{i} + \Pi_{i=\lfloor n/2\rfloor+1}^{n-1}\frac{n}{i} \\ &\geq \Pi_{i=2}^{\lfloor n/2\rfloor}\frac{n}{i} \geq \Pi_{i=2}^{\lfloor n/2\rfloor}\frac{n}{(n/2)} \geq \Pi_{i=2}^{\lfloor n/2\rfloor}2 \geq 2^{\lfloor n/2\rfloor-1} \geq 2^{(n/2)-2} = \frac{1}{4}(\sqrt{2})^n.\end{aligned}$$

This means that in principle it might be very difficult for a computer to solve the problem for big values of n, unless it employs an efficient heuristics to avoid having to explore the entire state space. Having more banks available than in the version considered in question (e) ought to make the problem easier to solve, since it is easier to satisfy the constraint of never having the cannibals outnumber the missionaries at any bank. In fact, there is a trivial solution in this case: use half of the boats to bring all the cannibals to the goal bank, then use the rest to bring all the missionaries over there. But even though the problem should be easier in principle (and easier in practice for a human), it still might create more problems for a (naively implemented) computer program, since there are many more available actions to consider. For instance, a breadth-first search will certainly run out of time and memory for large values of n: it will explore an exponentially large state space (in n) before finding a solution.

02180 Intro to AI
Exercises for week **3**

Exercise 1

The following exercise is adapted from Exercise 9 in Chapter 3 of the textbook (3rd ed, Global Edition). The *missionaries and cannibals problem* is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.[1] This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

Solve the problem using the GRAPH-SEARCH algorithm. You can use the pseudocode from the book or the slides. Make sure to keep track of of your *frontier* and your *explored set*. Which *search strategy* are you using (in which order do you choose frontier nodes for expansion)? Does you search strategy guarantee that you find an optimal solution (a solution with fewest possible actions)?

[1] Whenever the boat goes from one side to the other, everybody has to go out before the boat can go back again. The boat can not sail without anybody in it.

Exercise 2

Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall. When thinking about the size of the state space etc. it can be an advantage to look at concrete mazes. Consider for instance the maze and video shown here:

http://ing.dk/video/video-robotmus-overvinder-labyrint-pa-3921-sekunder-124487

a. How can the states of this problem be represented? How large is then the state space, measured as a function of the number of locations in the maze?

b. Define ACTIONS and RESULTS.

c. In navigating a maze, the only places we need to turn is at dead ends or intersections of two or more corridors (why?). Hence, from each point of the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using this observation, that is, reformulate ACTIONS and RESULTS. How large is the state space now?

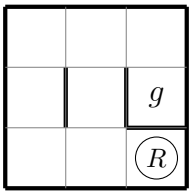
d. In our initial description of the problem we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

e. Can you say something about the search strategy that the mouse in the video appears to be using? You are not expected to be able to give a very precise answer to this question.

f. Is finding your way through a maze like this a difficult problem in AI?

Exercise 3

In the previous exercise, we considered a robot navigating a maze. Consider the following slight variation of the problem. The robot, R, is characterised by its position only, it does not have a direction. In each state, the robot can move one cell north (N), west (W), south (S) or east (E) (unless, of course, blocked by a wall in that direction). The goal of the robot is to reach the cell marked with a g. Consider the following instance of the problem:



You should assume that the robot—and the algorithms considered below—always explore the possible moves in the following order: N, W, S, E.

a. Illustrate how BFS graph search would solve this instance of the problem. You should: 1) draw the graph generated by BFS; 2) put numbers on the nodes of the graph according to the order in which they are generated, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the FIFO queue used for the frontier.

b. Illustrate how DFS graph search would solve this instance of the problem. You should: 1) draw the graph generated by DFS; 2) put numbers on the nodes of the graph according to the order in which they are visited, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the LIFO queue (stack) used for the frontier.

c. Illustrate how DFS tree search would solve this instance of the problem. You should: 1) draw the tree generated by DFS; 2) put numbers on the nodes of the tree according to the order in which they are visited, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the LIFO queue (stack) used for the frontier.

d. *OPTIONAL.* Do the same as in (c) for iterative deepening DFS. This time you don't have to show each single node explored and each single configuration of the LIFO queue, but you should try to at least informally go through each step of the algorithm.

e. Could a different order of exploring the moves change the solution found by DFS? And the solution found by BFS?

f. Provide a table with an overview of the performance of the algorithms considered above in terms of: 1) number of nodes generated; 2) cost of the solution found (number of moves to get to the goal).

02180 Intro to AI
Exercises for week 3
SOLUTIONS

Exercise 1

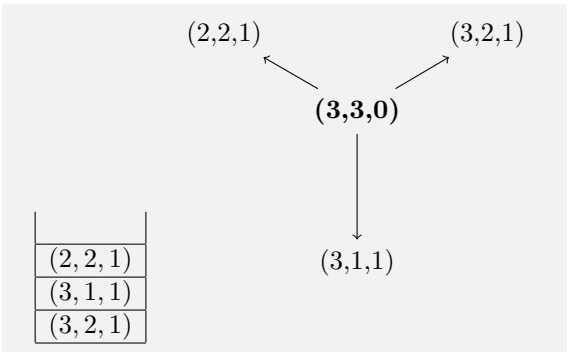
The following exercise is adapted from Exercise 9 in Chapter 3 of the textbook (3rd ed, Global Edition). The *missionaries and cannibals problem* is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.[1] This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

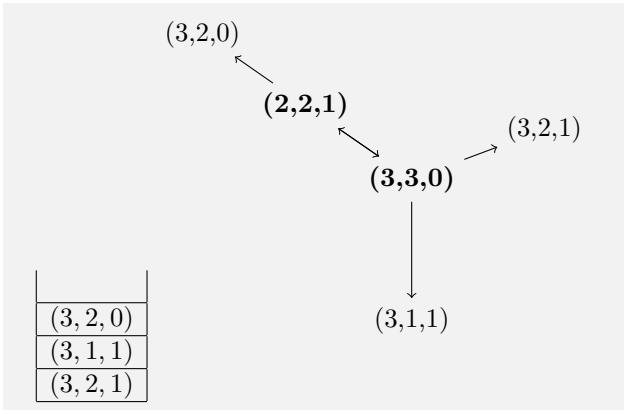
Solve the problem using the GRAPH-SEARCH algorithm. You can se the pseudocode from the book or the slides. Make sure to keep track of of your *frontier* and your *explored set*. Which *search strategy* are you using (in which order do you choose frontier nodes for expansion)? Does you search strategy guarantee that you find an optimal solution (a solution with fewest possible actions)?

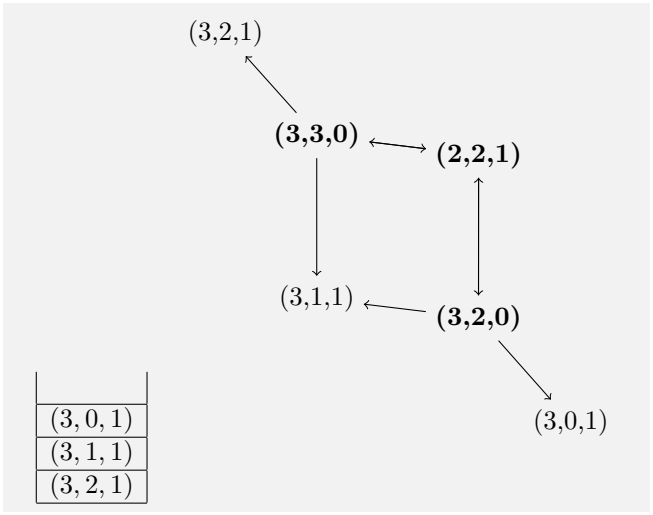
SOLUTION. GRAPH-SEARCH automatically checks for repeated states, which is a good idea in this case, since a DFS tree search might get stuck in the oop $(3, 3, 0) \rightarrow (2, 2, 1) \rightarrow (3, 2, 0) \rightarrow (3, 1, 1) \rightarrow (3, 3, 0)$. BFS tree search does not risk entering an infinite loop, but the search tree will grow exponentially with the depth, because the initial state will keep being re-expanded at every alternate level of the tree, and it has an outdegree of two (two outgoing edges).

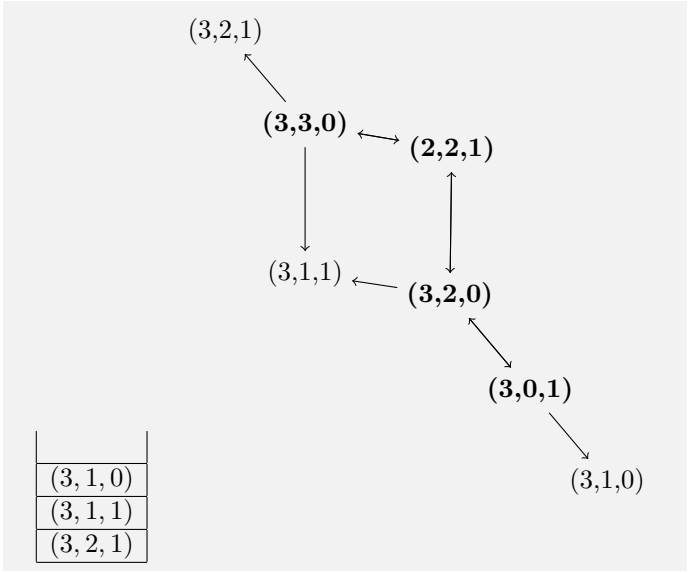
GRAPH-SEARCH can be used with different search strategies. We will here choose DFS. This means that the frontier is a stack. Below we show the first 4 steps of the algorithm, where the boldface nodes are the explored ones, and the frontier is visualised as a stack to the left:

[1] Whenever the boat goes from one side to the other, everybody has to go out before the boat can go back again. The boat can not sail without anybody in it.


(2,2,1)
(3,2,1)
(3,3,0)
(3,1,1)
(2, 2, 1)
(3, 1, 1)
(3, 2, 1)


(3,2,0)
(2,2,1)
(3,2,1)
(3,3,0)
(3,1,1)
(3, 2, 0)
(3, 1, 1)
(3, 2, 1)


(3,2,1)
(3,3,0)
(2,2,1)
(3,1,1)
(3,2,0)
(3,0,1)
(3, 0, 1)
(3, 1, 1)
(3, 2, 1)


(3,2,1)
(3,3,0)
(2,2,1)
(3,1,1)
(3,2,0)
(3,0,1)
(3,1,0)
(3, 1, 0)
(3, 1, 1)
(3, 2, 1)

Exercise 2

Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall. When thinking about the size of the state space etc. it can be an advantage to look at concrete mazes. Consider for instance the maze and video shown here:

http://ing.dk/video/video-robotmus-overvinder-labyrint-pa-3921-sekunder-124487

a. How can the states of this problem be represented? How large is then the state space, measured as a function of the number of locations in the maze?

SOLUTION. The states of the problem can be represented as triples (x, y, d) consisting of the x- and y-coordinates of the agent as well as its direction d. The size of the state space is the number of possible triples (x, y, d), which is 4 times the number of locations in the maze.

b. Define ACTIONS and RESULTS.

SOLUTION. The actions can be taken to be $\mathit{Turn}(d)$, where $d = W, E, N, S$ and $\mathit{Move}(l)$, where l is a natural number (the length/distance). All actions are applicable in any state, so

$$\textsc{Actions}(s) = \{\mathit{Turn}(d) \mid d \in W, E, N, S\} \cup \{\mathit{Move}(l) \mid l > 0\}$$

The transition model is:

$$\textsc{Result}((x, y, d), \mathit{Turn}(d')) = (x, y, d')$$
$$\textsc{Result}((x, y, d), \mathit{Move}(l)) = (x + x', y + y', d),$$

where if $d = E$ then x' is the maximal number $\leq l$ such that there is no wall in any of the locations $(x + 1, y), (x + 2, y), \ldots, (x + x', y)$. Similarly for $d = W, N, E$.

c. In navigating a maze, the only places we need to turn is at dead ends or intersections of two or more corridors (why?). Hence, from each point of the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using this observation, that is, reformulate ACTIONS and RESULTS. How large is the state space now?

SOLUTION. The idea is to remove the *Turn* action, and only have actions $\mathit{Move}(d)$ where $d = W, E, N, S$. The transition model is:

$$\textsc{Result}((x, y, d), \mathit{Move}(d')) = (x + x', y + y', d'),$$

where if $d' = E$ then x' is the maximal number such that $(x+1, y), (x+2, y), \ldots, (x+x'-1, y)$ are not goal locations, not walls and not intersections, and $(x+x', y)$ is not a wall. Note that the parameter d in the argument of RESULT is not used in defining the value of RESULT. This means that we can omit d from the state description and simply describe states as pairs (x, y). The transition model then becomes:

$$\textsc{Result}((x, y), \mathit{Move}(d')) = (x + x', y + y'),$$

where x' and y' are defined in terms of d' as above. The size of the state space is clearly now only the number of locations. We are abstracting away from the direction the robot is facing. In fact, the state space is only the number of intersections and dead ends plus possible the initial state and the goal (that might neither be intersections nor dead ends).

d. In our initial description of the problem we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

SOLUTION. (1) The robot can only be in a discrete location, not between locations (compare with the robot mouse video). (2) The robot can only face one of 4 discrete directions, N, S, E and W. (3) The robot is assumed to be able to move precisely a given number of cells. In real life, the robot would probably be a bit imprecise and would need sensors to tell whether a wall has been hit (even if it is assumed to have a full internal representation of the maze).

e. Can you say something about the search strategy that the mouse in the video appears to be using? You are not expected to be able to give a very precise answer to this question.

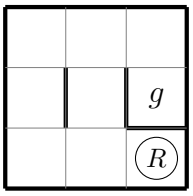
SOLUTION. It appears to be a kind of depth-first (online) search. It could be a greedy-best first search with some heuristics guiding it to the center of the maze, but this is not quite clear. Not also that the mouse does not initially know the maze, so it is really a search problem under *partial observability*, which is different from what we have considered so far.

f. Is finding your way through a maze like this a difficult problem in AI?

SOLUTION. No, in general it is a very simple problem, since it is a rather basic search problem. It can still be a challenge in very big and complex mazes, but computers would generally be much better than humans at solving mazes. They can search a bigger space much more quickly, and there is no deep intuition that a human can use to solve mazes more efficiently than using a simple search algorithm as the computer does (unlike the situation in e.g. chess or Go or natural language understanding).

Exercise 3

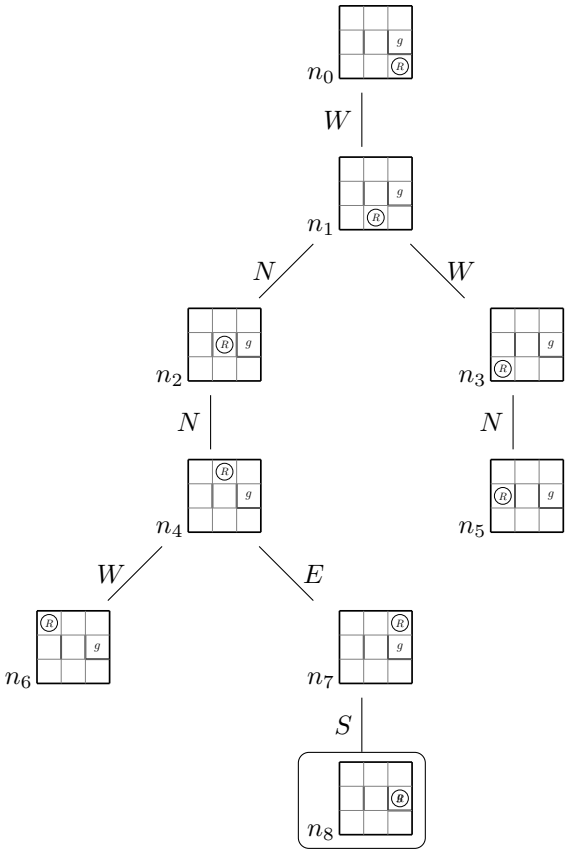
In the previous exercise, we considered a robot navigating a maze. Consider the following slight variation of the problem. The robot, R, is characterised by its position only, it does not have a direction. In each state, the robot can move one cell north (N), west (W), south (S) or east (E) (unless, of course, blocked by a wall in that direction). The goal of the robot is to reach the cell marked with a g. Consider the following instance of the problem:



You should assume that the robot—and the algorithms considered below—always explore the possible moves in the following order: N, W, S, E.

a. Illustrate how BFS graph search would solve this instance of the problem. You should: 1) draw the graph generated by BFS; 2) put numbers on the nodes of the graph according to the order in which they are generated, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the FIFO queue used for the frontier.

SOLUTION. Following the pseudocode from R&N figure 3.11, we get the search tree:



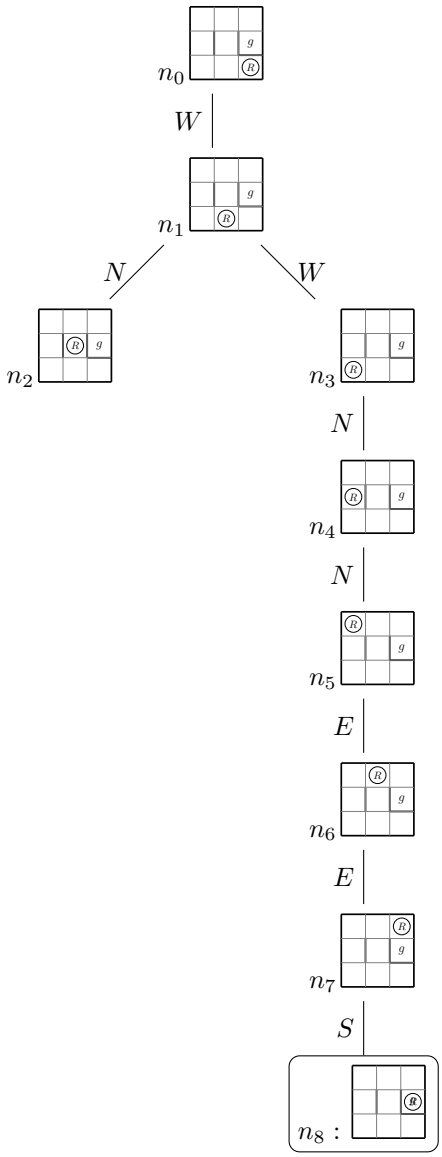
The frontier content is shown as it looks in the beginning of each loop iteration:

Iteration	**Frontier**
1	$[n_0]$
2	$[n_1]$
3	$[n_2, n_3]$
4	$[n_3, n_4]$
5	$[n_4, n_5]$
6	$[n_5, n_6, n_7]$
7	$[n_6, n_7]$
8	$[n_7]$

Because the BFS algorithm checks for goal states before adding states to the frontier, we will never see n_8 in the frontier and the algorithm terminates in the 8th iteration. The solution found is $WNNES$ of length 5. It is optimal (since we're using BFS).

b. Illustrate how DFS graph search would solve this instance of the problem. You should: 1) draw the graph generated by DFS; 2) put numbers on the nodes of the graph according to the order in which they are visited, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the LIFO queue (stack) used for the frontier.

SOLUTION. Following the pseudocode in R&N figure 3.7 for Graph-Search with a stack for the frontier, we get the search tree:



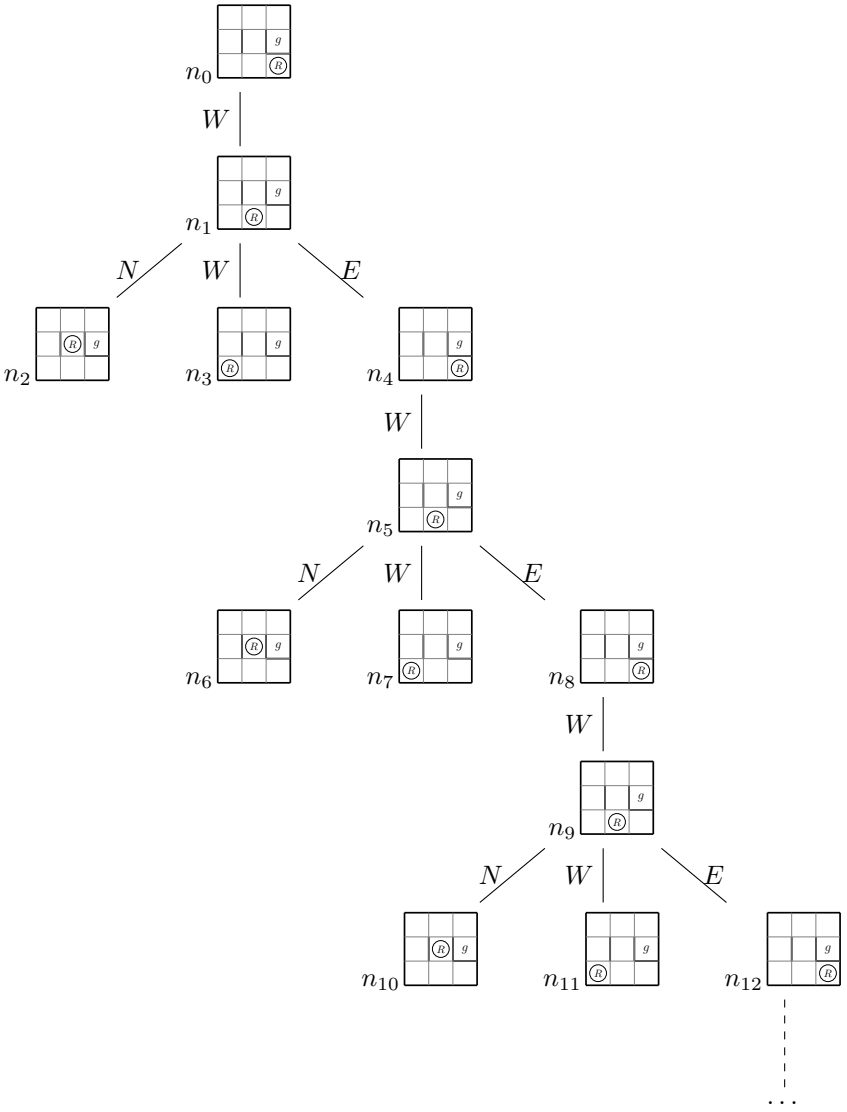
The frontier content is shown as it looks in the beginning of each loop iteration:

Iteration	Frontier
1	$[n_0]$
2	$[n_1]$
3	$[n_2, n_3]$
4	$[n_2, n_4]$
5	$[n_2, n_5]$
6	$[n_2, n_6]$
7	$[n_2, n_7]$
8	$[n_2, n_8]$

This time we see the goal state n_8 in the frontier, and terminate when we pop it in the 8th iteration. The solution found is $WWNNEES$ of length 7. It is not optimal.

c. Illustrate how DFS tree search would solve this instance of the problem. You should: 1) draw the tree generated by DFS; 2) put numbers on the nodes of the tree according to the order in which they are visited, e.g. use names $n_0, n_1, n_2, \ldots$; 3) for each iteration of the search loop, show the content of the LIFO queue (stack) used for the frontier.

SOLUTION. Following the pseudocode in R&N figure 3.7 for Tree-Search with a stack for the frontier, we get the search tree:



The frontier content is shown as it looks in the beginning of each loop iteration:

Iteration	**Frontier**
1	$[n_0]$
2	$[n_1]$
3	$[n_2, n_3, n_4]$
4	$[n_2, n_3, n_5]$
5	$[n_2, n_3, n_6, n_7, n_8]$
6	$[n_2, n_3, n_6, n_7, n_9]$
7	$[n_2, n_3, n_6, n_7, n_{10}, n_{11}, n_{12}]$
...	...

The search is stuck in a loop of exploring the same sequence of non-goal states over and over, and will never terminate.

d. *OPTIONAL.* Do the same as in (c) for iterative deepening DFS. This time you don't have to show each single node explored and each single configuration of the LIFO queue, but you should try to at least informally go through each step of the algorithm.

e. Could a different order of exploring the moves change the solution found by DFS? And the solution found by BFS?

SOLUTION. There is only one optimal solution, so any order of exploring possible moves will result in BFS finding the same (optimal) solution. Different orders of possible moves can lead DFS to find different solutions, e.g. the order W, S, E, N will result in DFS finding the optimal solution with both Graph-Search and Tree-Search. In general, there may not be any single fixed order leading DFS to find optimal solutions (or any solutions at all in the case of the Tree-Search variant).

f. Provide a table with an overview of the performance of the algorithms considered above in terms of: 1) number of nodes generated; 2) cost of the solution found (number of moves to get to the goal).

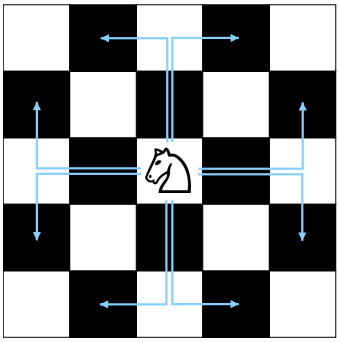
SOLUTION. The performance of each algorithm is as follows:

Algorithm	**Nodes generated**	**Solution length**
BFS	9	5
Graph-Search DFS	9	7
Tree-Search DFS	∞	N/A

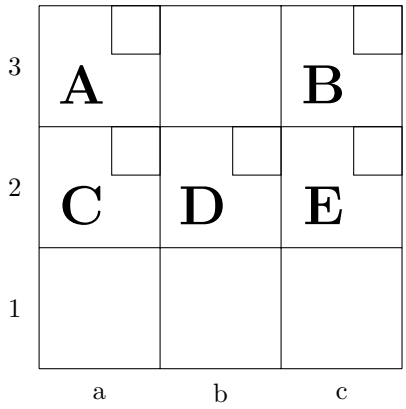
02180 Intro to AI Exercises
for week 4

Exercise 1

A knight in the game of chess can move and attack as shown by the arrows in the figure below.



In the following, we want to place five knights on a 3x3 board such that none of them can attack each other. Let A, B, C, D and E denote the five knights. Initially, the knights are placed as shown below.



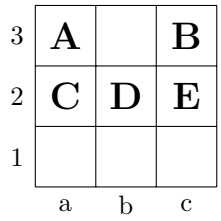
In this state, A can attack E by moving to position 2c, A can also move to position 1b without attacking another knight.

Let the number of conflicts of a knight be equal to the number of knights that can attack it.

1. For the initial state of the board shown above, write the number of conflicts of each knight in the small box next to its letter.

2. Consider the search problem in which the initial state is the initial configuration of the board shown above, the applicable actions are the legal moves of the knights, and a goal state is one in which none of the knights have a conflict. The cost of each action is assumed to be 1. Consider the heuristic function $h(n)$, where the value of $h(n)$ is defined to be the total sum of conflicts of the knights in the state of n (e.g., in a state of n where A, B, C and D each has 1 conflict and E has no conflicts, we have $h(n) = 4$).

   Draw below the search tree of the GRAPH-SEARCH version of greedy best-first search using the heuristic function $h(n)$. Write each search node as a board state and take note of its h-value. Indicate the expanded nodes by ✓. The root of the tree is already shown. You can assume that no new nodes are expanded as soon as a goal state has been generated. Assume that when expanding a node, moves of knight A are considered first, then moves of knight B, etc.



$h =$

3. Enter the solution found by the algorithm into the table below. You might not need all rows.

Move number	Letter of piece	moved from	moved to
1			
2			
3			
4			
5			
6			

4. Assume $A^\star$ is used instead of greedy best-first search. Moves are considered in the same order as before, and it is still assumed that no new nodes are expanded as soon as a goal state has been generated. Will the search tree then be the same? Explain your answer.

5. Assume BFS is used instead of greedy-best search. Moves are considered in the same order as before. Will the search tree then be the same as for greedy best-first search? Explain your answer.

6. Is $h(n)$ an admissible heuristics for the search problem? Explain your answer.

Exercise 2

1. Consider the state space in Figure A below (the last page of this document). The h-values are in black, the step costs are in blue. The h-values are straight-line distances to g, and the step costs are straight-line distances between pairs of points. Run greedy best-first graph search by hand on this state space, finding a path from the initial state s_0 to the goal state g. Make

sure to take note of the order in which the nodes are expanded as well as the content of the frontier after each iteration of the main loop of GRAPH-SEARCH. Highlight each edge traversed as well as all expanded nodes.

2. What is the length of the path found by greedy best-first search? How many nodes are generated by the search? How many nodes are expanded?

3. In the slides and in the textbook, $A^\star$ is claimed to be an instance of the general graph-search algorithm. There is however one complication. $A^\star$ is a best-first search algorithm relying both on a heuristic value (h-value) and a path cost (g-value). The h-value of a node n is a constant, but the g-value will depend on the path taken to get to n from the initial state. This means that the g-value potentially needs to be updated during the search if a shorter path to n is found. Otherwise we can not be guaranteed to find the shortest path, even if the heuristics is consistent. The solution to this problem is to add the following line at the end of the GRAPH-SEARCH algorithm on page 2 of `slides03.pdf` (inside the **for**-loop):

   if $m \in frontier$ and $g(m) > g(n) + c(n, m)$ **then** let n be the new parent of m

   Here $c(n, m)$ is the cost of the edge from n to m. Note that when n is made the new parent of m, the g- and f-values of m will also implicitly be updated. Now repeat the search from above, but using $A^\star$ instead of greedy best-first search. Make sure to do the same book-keeping as for greedy best-first search and, in addition, mark each generated state with its g- and f-values.

4. What is the length of the path found by $A^\star$? How many nodes are generated by the search? How many nodes are expanded?

5. How do you know that the path produced by $A^\star$ is optimal (has minimal cost)?

6. Consider adding an extra edge from s_9 to g. What is the cost of this edge? What is now the cost (length) of a shortest path from s_0 to g? Show that if the extra line mentioned in question 3 is not added to the pseudocode of GRAPH-SEARCH, $A^\star$ will not find the optimal solution to the search problem with the extra edge added.

7. Consider the following modified search problem. We wish to find a shortest path from s_0 to g measured in *number of edges traversed*, that is, we take the step cost to be constant 1. Can we still use $A^\star$ with the existing heuristics to find such a shortest path? If not, then how can we (easily) modify the existing heuristics to ensure that $A^\star$ will find a shortest path?

8. In the example considered here, greedy best-first search generates and expands fewer nodes than $A^\star$. Provide an example of an alternative graph where greedy best-first search generates and expands *more* nodes than $A^\star$. The nodes of the graph should still represent points in the plane, the step cost of edges should be the straight-line distance, and the heuristics should be straight-line distance to the goal.

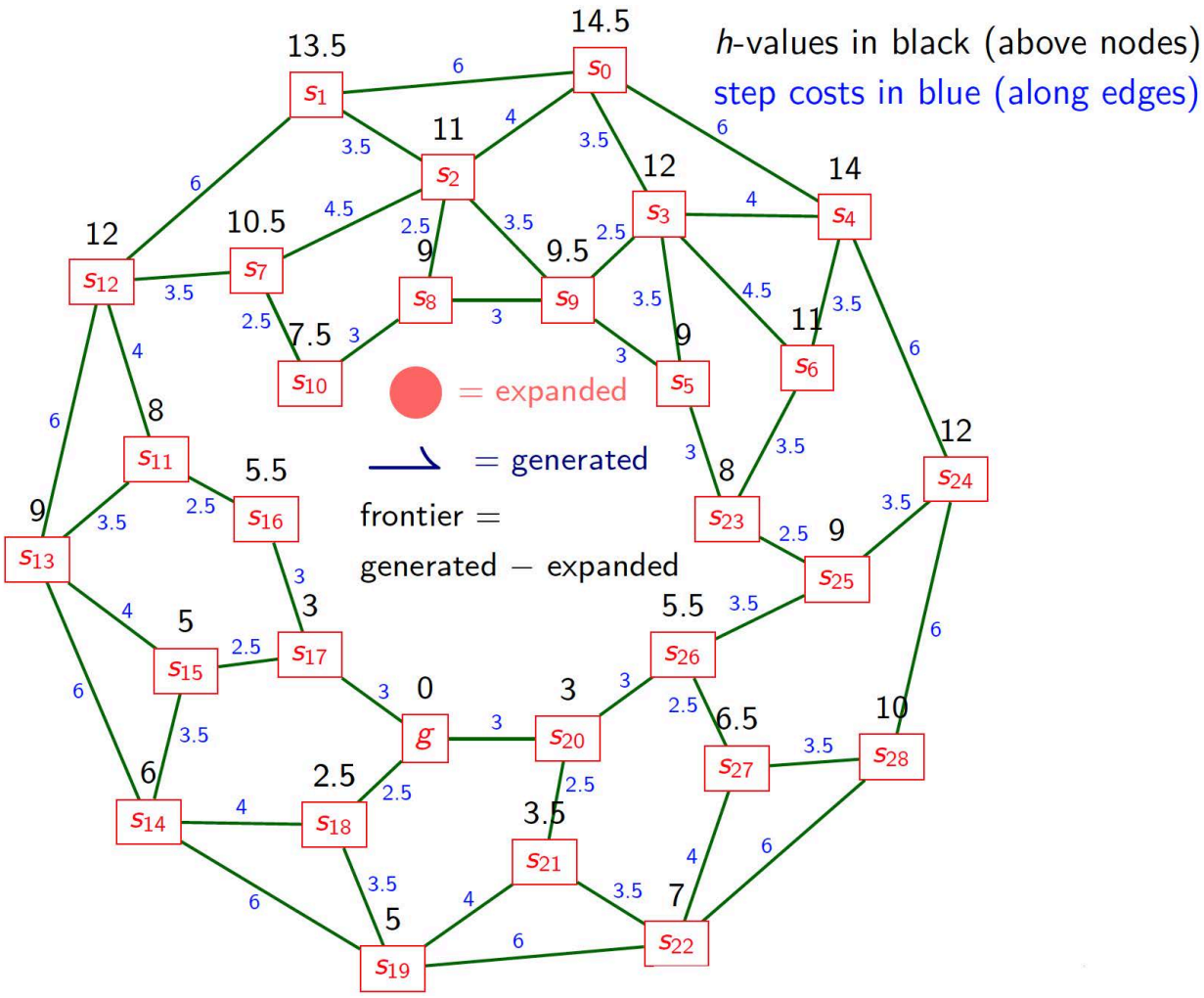
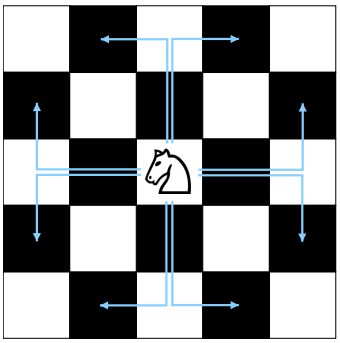


FIGURE A (Exercise 2)

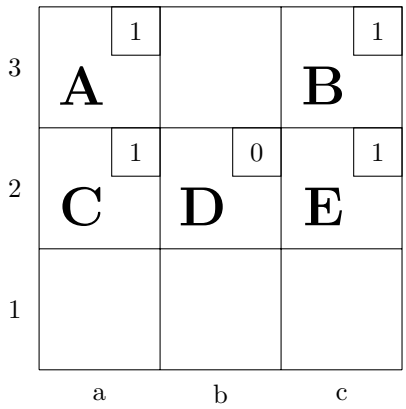
02180 Intro to AI
Exercises for week 4
SOLUTIONS

Exercise 1

A knight in the game of chess can move and attack as shown by the arrows in the figure below.



In the following, we want to place five knights on a 3x3 board such that none of them can attack each other. Let A, B, C, D and E denote the five knights. Initially, the knights are placed as shown below.



In this state, A can attack E by moving to position 2c, A can also move to position 1b without attacking another knight.

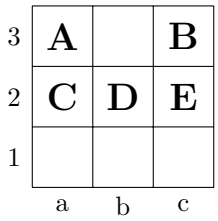
Let the number of conflicts of a knight be equal to the number of knights that can attack it.

1. For the initial state of the board shown above, write the number of conflicts of each knight in the small box next to its letter.

   SOLUTION. See above.

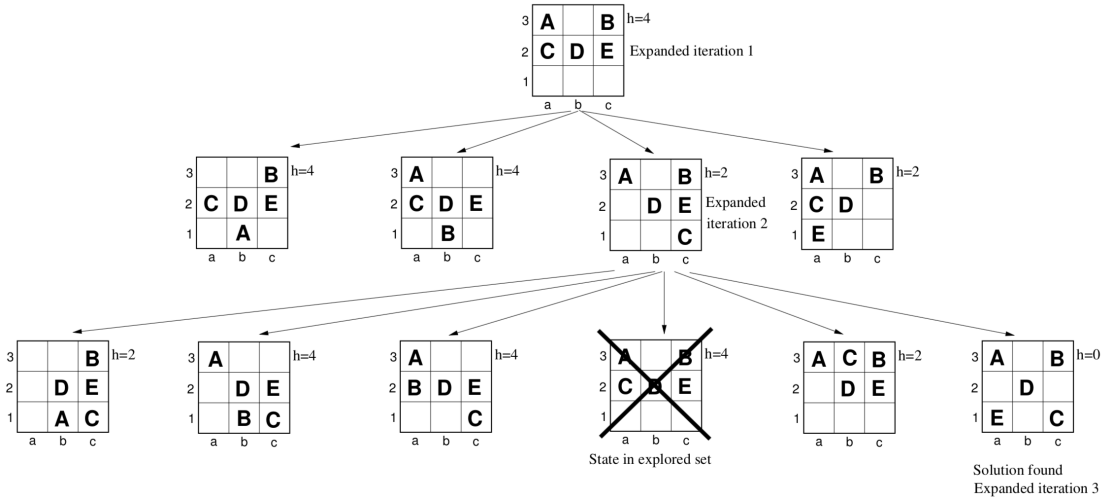
2. Consider the search problem in which the initial state is the initial configuration of the board shown above, the applicable actions are the legal moves of the knights, and a goal state is one in which none of the knights have a conflict. The cost of each action is assumed to be 1. Consider the heuristic function $h(n)$, where the value of $h(n)$ is defined to be the total sum of conflicts of the knights in the state of n (e.g., in a state of n where A, B, C and D each has 1 conflict and E has no conflicts, we have $h(n) = 4$).

   Draw below the search tree of the GRAPH-SEARCH version of greedy best-first search using the heuristic function $h(n)$. Write each search node as a board state and take note of its h-value. Indicate the expanded nodes by ✓. The root of the tree is already shown. You can assume that no new nodes are expanded as soon as a goal state has been generated. Assume that when expanding a node, moves of knight A are considered first, then moves of knight B, etc.



$h =$

   SOLUTION.



3. Enter the solution found by the algorithm into the table below. You might not need all rows.

Move number	Letter of piece	moved from	moved to
1			
2			
3			
4			
5			
6			

SOLUTION.

Move number	Letter of piece	moved from	moved to
1	C	2a	1c
2	E	2c	1a

4. Assume $A^\star$ is used instead of greedy best-first search. Moves are considered in the same order as before, and it is still assumed that no new nodes are expanded as soon as a goal state has been generated. Will the search tree then be the same? Explain your answer.

   SOLUTION. Yes. In level 1 we will have the f-values 1 higher than the h-values, but of course at level 1 we will still expand one with minimum h-value. Already at level 2 we find a solution.

5. Assume BFS is used instead of greedy-best search. Moves are considered in the same order as before. Will the search tree then be the same as for greedy best-first search? Explain your answer.

   SOLUTION. No. The leftmost child of the root will be expanded before the child with minimal h-value.

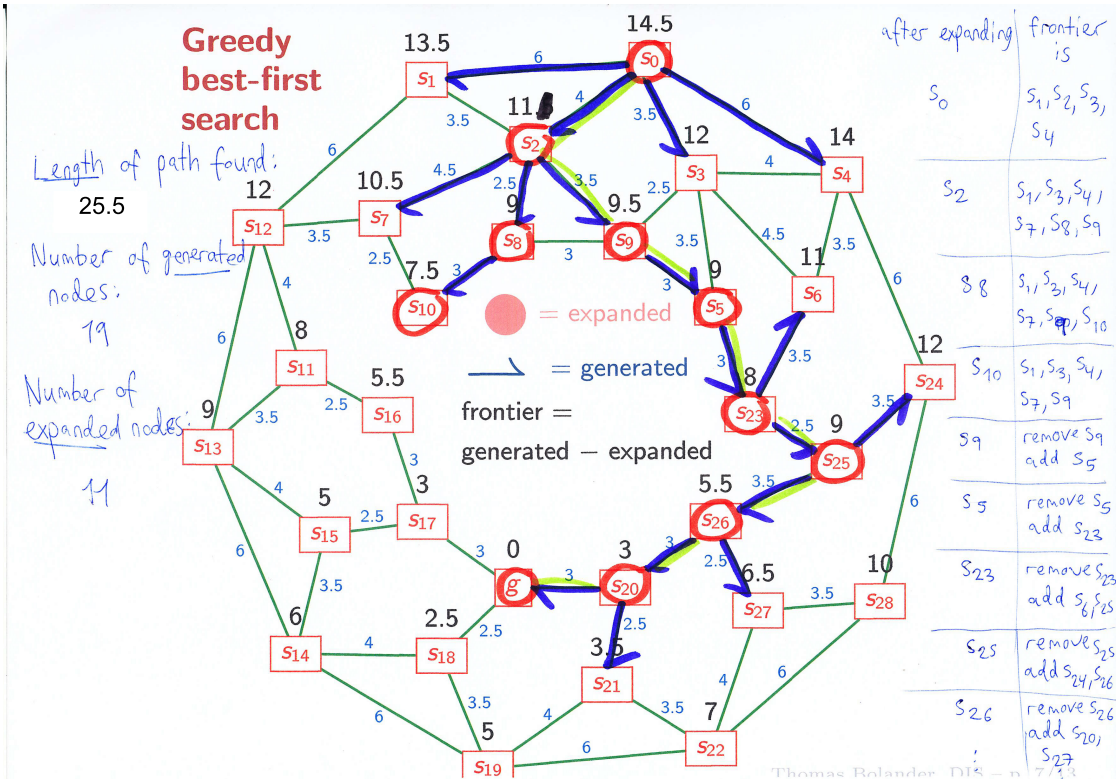
6. Is $h(n)$ an admissible heuristics for the search problem? Explain your answer.

   SOLUTION. No. The expanded child of the root above has $h = 2$ but is only one move away from the goal.

Exercise 2

1. Consider the state space on slide 8 from today (`slides03.pdf`). The h-values are in black, the step costs are in blue. The h-values are straight-line distances to g, and the step costs are straight-line distances between pairs of points. Run greedy best-first graph search by hand on this state space, finding a path from the initial state s_0 to the goal state g. Make sure to take note of the order in which the nodes are expanded as well as the content of the frontier after each iteration of the main loop of Graph-Search. Highlight each edge traversed as well as all expanded nodes.

SOLUTION. In the solution below, the frontier is only shown for the first iterations of the loop.



2. What is the length of the path found by greedy best-first search? How many nodes are generated by the search? How many nodes are expanded?

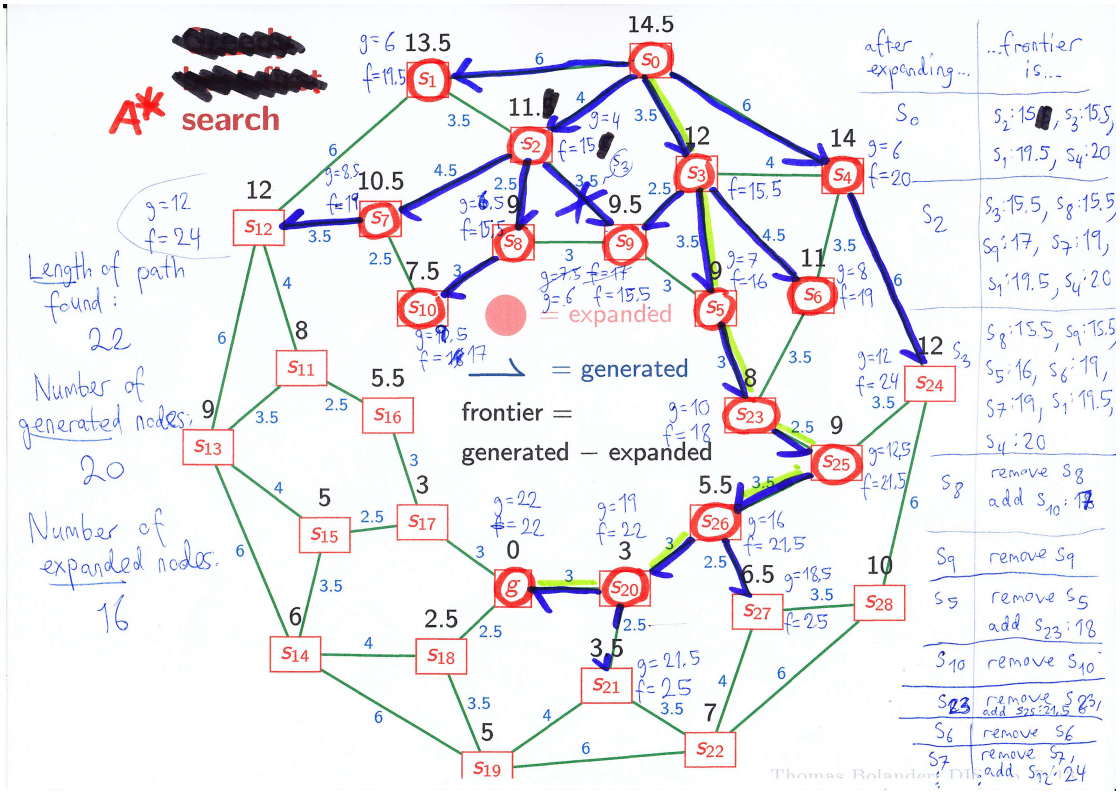
   SOLUTION. These numbers are included in the picture above.

3. In the slides and in the textbook, $A^\star$ is claimed to be an instance of the general graph-search algorithm. There is however one complication. $A^\star$ is a best-first search algorithm relying both on a heuristic value (h-value) and a path cost (g-value). The h-value of a node n is a constant, but the g-value will depend on the path taken to get to n from the initial state. This means that the g-value potentially needs to be updated during the search if a shorter path to n is found. Otherwise we can not be guaranteed to find the shortest path, even if the heuristics is consistent. The solution to this problem is to add the following line at the end of the GRAPH-SEARCH algorithm on page 2 of `slides03.pdf` (inside the **for**-loop):

   if $m \in frontier$ and $g(m) > g(n) + c(n, m)$ **then** let n be the new parent of m

   Here $c(n, m)$ is the cost of the edge from n to m. Note that when n is made the new parent of m, the g- and f-values of m will also implicitly be updated. Now repeat the search from above, but using $A^\star$ instead of greedy best-first search. Make sure to do the same book-keeping as for greedy best-first search and, in addition, mark each generated state with its g- and f-values.

SOLUTION. In the solution below, the frontier is only shown for the first iterations of the loop.



4. What is the length of the path found by $A^\star$? How many nodes are generated by the search? How many nodes are expanded?

   SOLUTION. These numbers are included in the picture above.

5. How do you know that the path produced by $A^\star$ is optimal (has minimal cost)?

   SOLUTION. The heuristics and step costs are based on straight-line distances, so if m is reached by executing a in n, then

   $$\begin{array}{rcl} h(n) & = & \text{straight-line distance from } n \text{ to } g \\ & \leq & \text{straight-line distance from } n \text{ to } g \text{ via } m \\ & = & \text{straight-line distance from } n \text{ to } m \text{ plus straight-line distance from } m \text{ to } g \\ & = & c(n,a,m) + h(m). \end{array}$$

   This proves that h is consistent. $A^\star$ graph-search is optimal when h is consistent, hence we can be guaranteed that the path found is optimal.

6. Consider adding an extra edge from s_9 to g. What is the cost of this edge? What is now the cost (length) of a shortest path from s_0 to g? Show that if the extra line mentioned in question 3 is not added to the pseudocode of Graph-Search, $A^\star$ will not find the optimal solution to the search problem with the extra edge added.

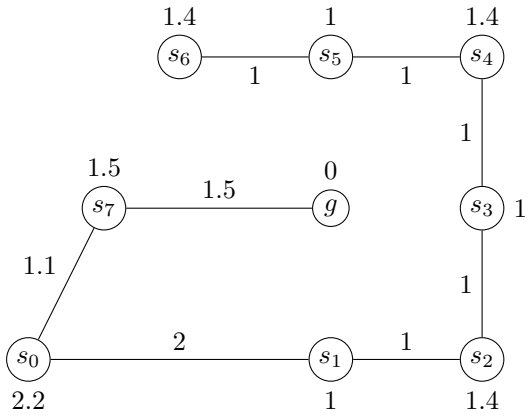
SOLUTION. Cost of new edge = heuristic value of $s_9 = 9.5$. Shortest path is (s_0, s_3, s_9, g) and it has a length of $3.5 + 2.5 + 9.5 = 15.5$. However, $A^\star$ without the added line will find the non-optimal path going via s_2 and having a cost of $4 + 3.5 + 9.5 = 17$.

7. Consider the following modified search problem. We wish to find a shortest path from s_0 to g measured in *number of edges traversed*, that is, we take the step cost to be constant 1. Can we still use $A^\star$ with the existing heuristics to find such a shortest path? If not, then how can we (easily) modify the existing heuristics to ensure that $A^\star$ will find a shortest path?

   SOLUTION. Divide all heuristic values by the length of the longest edge = 6. Then the consistency condition will be satisfied when the edge cost is 1. Since before we had $h(n) \leq c(n, n') + h(n')$. Call the new h values h'. Then $6h'(n) \leq c(n, n') + 6h'(n')$ and hence $h'(n) \leq \frac{c(n,n')}{6} + h'(n') \leq 1 + h'(n')$, which is the consistency condition for edge cost 1.

8. In the example considered here, greedy best-first search generates and expands fewer nodes than $A^\star$. Provide an example of an alternative graph where greedy best-first search generates and expands *more* nodes than $A^\star$. The nodes of the graph should still represent points in the plane, the step cost of edges should be the straight-line distance, and the heuristics should be straight-line distance to the goal.

   SOLUTION.

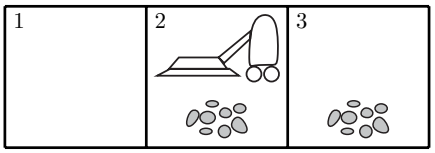


   $A^\star$ will first generate children s_7 and s_1 with $f(s_7) = 1.1 + 1.5 = 2.6$ and $f(s_1) = 3$. Then $f(s_7)$ will be expanded and we get $f(g) = 2.6$. Next g will be expanded, and we are done. So with $A^\star$, we expand s_0, s_7, g, in total 3 nodes. We additionally generate only s_1, so 4 nodes are generated. With greedy best-first search, the expansion order will be $s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, g$. So all 9 nodes will be expanded (and hence also generated).

02180 Intro to AI
Exercises for week 5

Exercise 1

To begin with, consider the deterministic and fully observable vacuum cleaner world with three squares, and the initial state being:



The goal states are the states where all squares are clean.

a. Draw a graph of the state space up to and including distance 2 from the initial state. Don't forget states that *lead* to other reachable states, and not only those that are reachable.

b. How many states are there up to distance 2 from the initial state? How many states are there in total for the problem with 3 squares? How many are goal states? How many of the states are reachable from the initial state?

   If there were n squares (horizontally) instead of 3, and there could be dirt in any and all of them in the initial states, how many states are there then in the state space, and how many of those are goal states?

Exercise 2

Now we move on to consider the *erratic* vacuum cleaner world with 3 squares, where the `Suck` action cleans the square of the robot, but also possibly any of the adjacent squares (e.g. when in square 2, performing a `Suck` action also possibly cleans square 1 and/or 3).

a. Using the tuple representation of states from the lecture, e.g. the initial state is $(c, d, d, 2)$, determine:

   1. RESULTS$((d, c, d, 1)$, `Suck`$)$
   2. RESULTS$((d, d, d, 1)$, `Suck`$)$
   3. RESULTS$((d, d, d, 2)$, `Suck`$)$
   3. RESULTS$((d, c, d, 2)$, `Suck`$)$

3. RESULTS$((c, d, d, 3)$, `Suck`$)$

b. Draw a graph of the state space up to and including distance 2 from the same initial state as the previous question. Each state can now have several outgoing edges with the same action label leading to different states.

c. Draw the AND-OR tree up to depth 2 (counting each and-or transition as 1 depth) with the given initial state as root. Mark the OR-nodes as:

- *GOAL*, if they're a goal state;
- *LOOP*, if they occur on the path to the root; and
- *OPEN*, if they're at depth 2 and are neither a goal or a loop state.

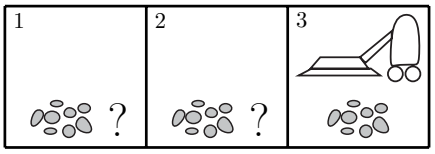
d. What is the maximal outgoing degree of any AND-node in the tree? Is there a state from which the AND-node of some action would have a higher out-degree, and if so then what state, action, and degree would that be?

e. Is there a solution in the AND-OR tree? Highlight the solution if it exists in the tree, and otherwise expand more of the tree so that it contains a solution and then highlight that. You can expand in any order you prefer. Write the conditional plan in the language from the slides; use parentheses to resolve ambiguous syntax.

f. Does the AND-OR tree correspond to any run of the AND-OR-GRAPH-SEARCH algorithm? If it doesn't, then draw a new AND-OR tree which could result from running the algorithm.

Exercise 3

Now consider the deterministic and partially observable *local-sensing* vacuum cleaner world, where the agent knows which square it is in, and senses whether that square is either clean or dirty after each action. The `Suck` action again only cleans the square that the agent is in. Consider the following initial state:



where the agent knows that there is dirt in square 3, but not whether square 1 or 2 are dirty or clean.

a. What is the initial belief state? You can use the tuple representation for each physical state. How many states are there in the belief state space? If there were n squares, then how many belief states are there?

b. We will use the notation $X@Y$ where X is `clean` or `dirty` and Y is 1, 2, or 3 to denote percepts for the squares being clean or dirty. Hence, `dirty`@2 is the percept that square 2 is dirty. Now define the following physical states:

s_0 : $(c, d, c, 3)$

s_1 : $(c, d, c, 2)$

s_2 : $(c, d, d, 3)$

s_3 : $(c, c, c, 3)$

s_4 : $(c, c, d, 3)$

s_5 : $(d, c, d, 3)$

s_6 : $(d, d, d, 3)$

s_7 : $(d, c, c, 3)$

Using the introduced percept notation, determine the following:

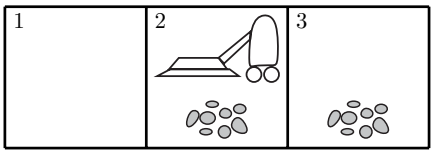
- PERCEPT(s_0)
- PERCEPT(s_1)
- PERCEPT(s_7)
- POSSIBLE-PERCEPTS($\{s_0, s_2, s_3, s_4\}$)
- UPDATE($\{s_0, s_2, s_3, s_4\}$, `dirty`@3)
- RESULTS($\{s_2, s_4, s_5, s_6\}$, `Left`)

d. Assume the modelling change so that the robot has a vague sensor which can only determine which of the following two is the case: 1) there is dirt in the current square or an adjacent square; 2) the current square and all adjacent squares are clean. We can model this new situation with only two percepts, `dirty` and `clean`. Explain why. Then determine the same function values as in b, except the percept `dirty@3` is replaced by `dirty`.

02180 Intro to AI
Exercises for week 5
SOLUTIONS

Exercise 1

To begin with, consider the deterministic and fully observable vacuum cleaner world with three squares, and the initial state being:



The goal states are the states where all squares are clean.

a. Draw a graph of the state space up to and including distance 2 from the initial state. Don't forget states that *lead* to other reachable states, and not only those that are reachable.

SOLUTION. See Figure 1.

b. How many states are there up to distance 2 from the initial state? How many states are there in total for the problem with 3 squares? How many are goal states? How many of the states are reachable from the initial state?

If there were n squares (horizontally) instead of 3, and there could be dirt in any and all of them in the initial states, how many states are there then in the state space, and how many of those are goal states?

SOLUTION. There are 8 states up to distance 2 from the initial state, found by counting in the drawn state space.

We can represent each state as a quadruple (s_1, s_2, s_3, r) where $s_1, s_2, s_3 \in \{\mathtt{c}, \mathtt{d}\}$ for square one to three being clean or dirty respectively, and $r \in \{1, 2, 3\}$ for the robot being in squares one to three respectively. This gives a total of $2 \cdot 2 \cdot 2 \cdot 3 = 24$ possible states, since no combinations represent invalid states. 3 of these states are goal states, namely $(\mathtt{c}, \mathtt{c}, \mathtt{c}, 1)$, $(\mathtt{c}, \mathtt{c}, \mathtt{c}, 2)$, and $(\mathtt{c}, \mathtt{c}, \mathtt{c}, 3)$. From the initial state, we can never reach a state which has dirt in square 1, but we can reach every other state. This means we can reach all states represented by tuples of the form $(\mathtt{c}, s2, s3, r)$ of which there are $2 \cdot 2 \cdot 3 = 12$.

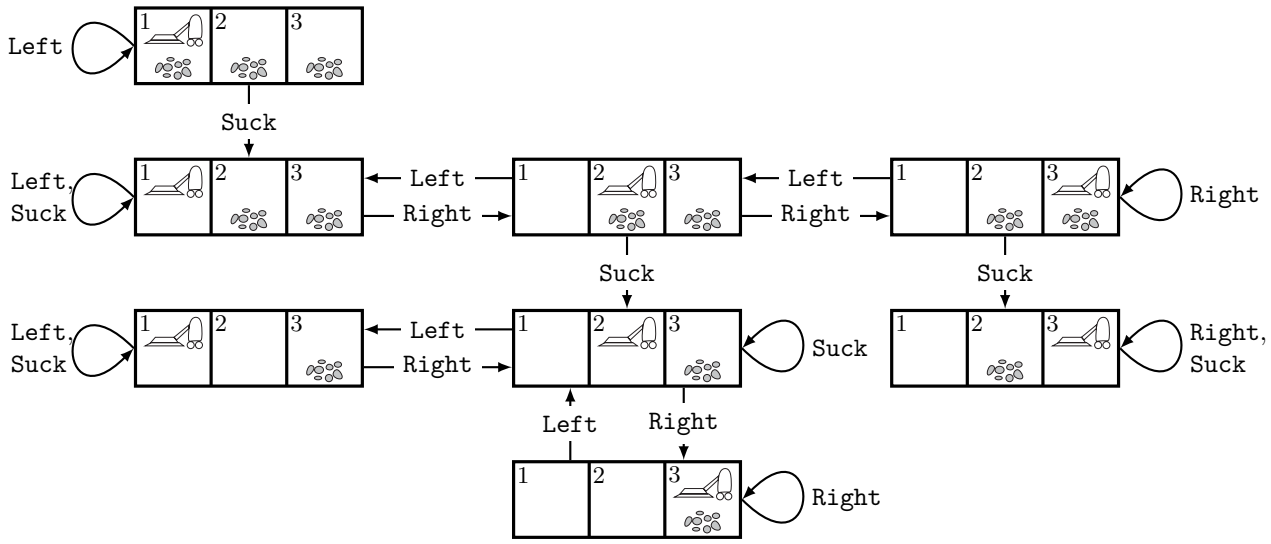


Figure 1: Solution for Exercise 1a.

If there are n squares, then each square still has two possible values in a state, and the agent can be in n squares. A tuple representation of such states could be $(s_1, s_2, \ldots, s_n, r)$ with values like before, for a total of $2^n \cdot n$ states, and of these only n are goal states.

Exercise 2

Now we move on to consider the *erratic* vacuum cleaner world with 3 squares, where the `Suck` action cleans the square of the robot, but also possibly any of the adjacent squares (e.g. when in square 2, performing a `Suck` action also possibly cleans square 1 and/or 3).

a. Using the tuple representation of states from the lecture, e.g. the initial state is $(c, d, d, 2)$, determine:

   1. RESULTS($(d, c, d, 1)$, `Suck`)
   2. RESULTS($(d, d, d, 1)$, `Suck`)
   3. RESULTS($(d, d, d, 2)$, `Suck`)
   3. RESULTS($(d, c, d, 2)$, `Suck`)
   3. RESULTS($(c, d, d, 3)$, `Suck`)

SOLUTION.

1. RESULTS($(d, c, d, 1)$, `Suck`) $= \{(c, c, d, 1)\}$.
2. RESULTS($(d, d, d, 1)$, `Suck`) $= \{(c, d, d, 1), (c, c, d, 1)\}$.

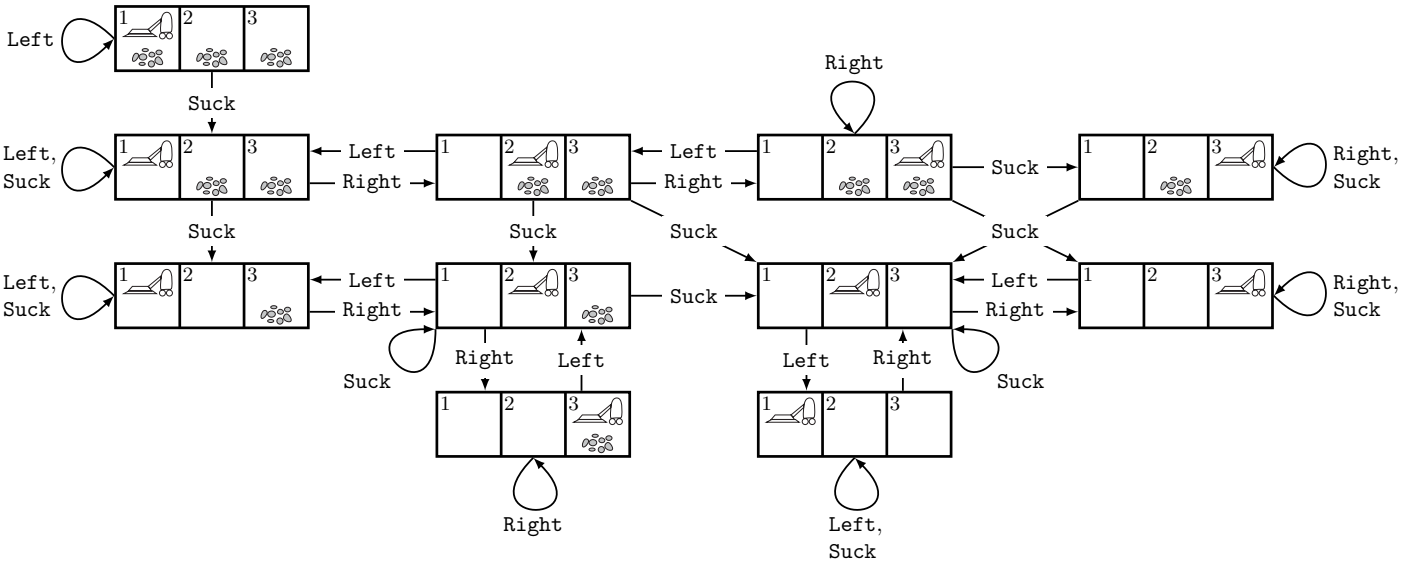


Figure 2: Solution for Exercise 2b.

3. RESULTS$((d, d, d, 2),$ `Suck`$) = \{(d, c, d, 2), (c, c, d, 2), (d, c, c, 2), (c, c, c, 2)\}$.
3. RESULTS$((d, c, d, 2),$ `Suck`$) = \{(d, c, d, 2), (c, c, d, 2), (d, c, c, 2), (c, c, c, 2)\}$.
3. RESULTS$((c, d, d, 3),$ `Suck`$) = \{(c, d, c, 3), (c, c, c, 3)\}$.

b. Draw a graph of the state space up to and including distance 2 from the same initial state as the previous question. Each state can now have several outgoing edges with the same action label leading to different states.

SOLUTION. See Figure 2.

c. Draw the AND-OR tree up to depth 2 (counting each and-or transition as 1 depth) with the given initial state as root. Mark the OR-nodes as:

- *GOAL*, if they're a goal state;
- *LOOP*, if they occur on the path to the root; and
- *OPEN*, if they're at depth 2 and are neither a goal or a loop state.

SOLUTION. See Figure 3, disregarding the red nodes and edges (at depth 3).

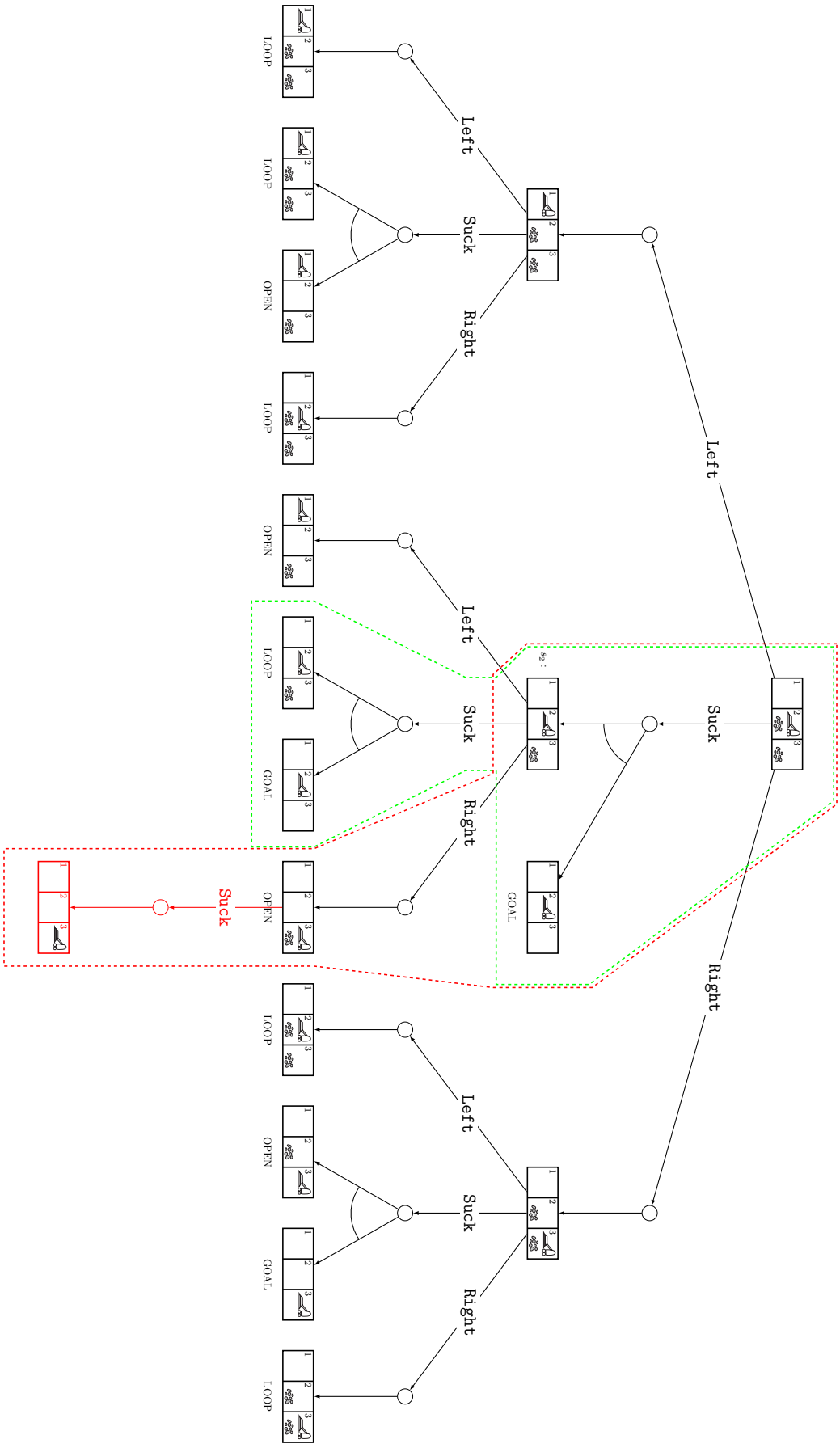


Figure 3: Solution for Exercise 2c and 2e.

d. What is the maximal outgoing degree of any AND-node in the tree? Is there a state from which the AND-node of some action would have a higher out-degree, and if so then what state, action, and degree would that be?

SOLUTION. Every AND-node from a `Suck` action in the tree has outgoing degree 2, which is maximal for this tree.

e. Is there a solution in the AND-OR tree? Highlight the solution if it exists in the tree, and otherwise expand more of the tree so that it contains a solution and then highlight that. You can expand in any order you prefer. Write the conditional plan in the language from the slides; use parentheses to resolve ambiguous syntax.

SOLUTION. There is no solution.

It is, however, possible to define so-called *cyclic solutions* that allow loops, but such solutions are only guaranteed to be succesful under some further conditions. First of all, a *fairness assumption* has to be made: If a non-deterministic action has several possible outcomes and is executed an infinite number of times, then eventually every possible outcome will be realised. In this exercise, it would imply that if executing `Suck` an infinite number of times in s_2, eventually the dirt in square 3 will disappear. Under these assumptions, even cyclic plans can be succesful, if the fairness assumption implies that all cycles in the solution will eventually be broken. A cyclic solution is highlighted in the green dashed line, and would have the conditional plan (using the label syntax of R&N):

$$[\texttt{Suck},\ \textbf{if}\ \mathit{state} = s_2\ \textbf{then}\ (L : [\texttt{Suck},\ \textbf{if}\ \mathit{state} = s_2\ \textbf{then}\ L\ \textbf{else}\ \epsilon])\ \textbf{else}\ \epsilon]$$

If we disregard cyclic plans, then we could expand the tree with the red nodes and edges, to get the solution contained in the red dashed line. The solution has the conditional plan:

$$[\texttt{Suck},\ \textbf{if}\ \mathit{state} = s_2\ \textbf{then}\ [\texttt{Right}, \texttt{Suck}]\ \textbf{else}\ \epsilon]$$

f. Does the AND-OR tree correspond to any run of the AND-OR-GRAPH-SEARCH algorithm? If it doesn't, then draw a new AND-OR tree which could result from running the algorithm.

SOLUTION. No. The AND-OR-GRAPH-SEARCH algorithm does a depth-first search in the state space, and thus must exhaust all options in any previous branches before exploring down new branches of the AND-OR search tree. In our AND-OR tree, we have explored a bit down every branch from the root, but not entirely exhausted any, which is not possible in a depth-first search. What we have done could result from a breadth-first exploration of the state space, but never a depth-first exploration like the considered search algorithm.

An AND-OR tree that might result from a run of the algorithm is shown in Figure 4.

Exercise 3

Now consider the deterministic and partially observable *local-sensing* vacuum cleaner world, where the agent knows which square it is in, and senses whether that square is either clean or dirty after each action. The `Suck` action again only cleans the square that the agent is in. Consider the following initial state:

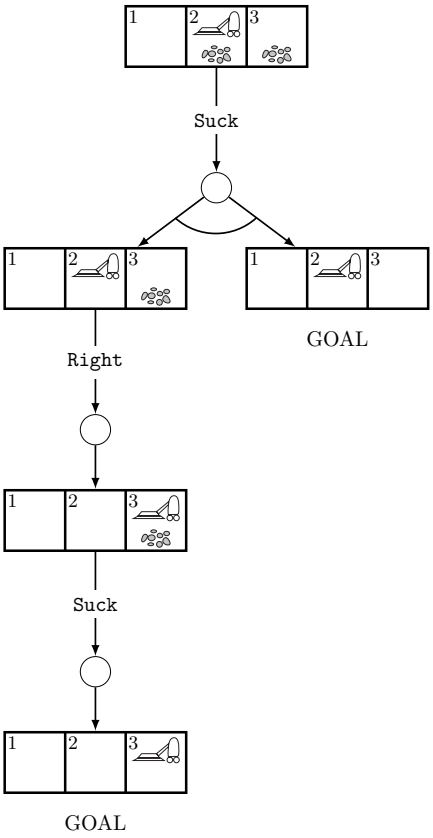
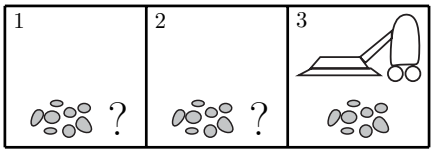


Figure 4: Solution for Exercise 2f.



where the agent knows that there is dirt in square 3, but not whether square 1 or 2 are dirty or clean.

a. What is the initial belief state? You can use the tuple representation for each physical state. How many states are there in the belief state space? If there were n squares, then how many belief states are there?

SOLUTION. The initial belief state is $\{(c, c, d, 3), (c, d, d, 3), (d, c, d, 3), (d, d, d, 3)\}$, encoding that the agent is in square 3 and that square 3 is dirty (consistent between all the physical states in the belief state), but that the remaining two squares can be in any configuration of clean and/or dirty (i.e. their state is unknown).

Given that there are 24 physical states, and that a belief state is any set of physical states, we get that there are $2^{24} = 16,777,216$ possible belief states.

When there are n squares, we found that there were $2^n \cdot n$ physical states, which in turn means there are $2^{2^n \cdot n}$ possible belief states, a double-exponential number of belief states in terms of n. For $n = 10$, we have $2^{2^{10} \cdot 10} = 2^{1024 \cdot 10} = 2^{10240} = 10^{\log 2 \cdot 10240} \approx 10^{3083}$.

b. We will use the notation $X@Y$ where X is `clean` or `dirty` and Y is 1, 2, or 3 to denote percepts for the squares being clean or dirty. Hence, `dirty`@2 is the percept that square 2 is dirty. Now define the following physical states:

s_0 : $(c, d, c, 3)$

s_1 : $(c, d, c, 2)$

s_2 : $(c, d, d, 3)$

s_3 : $(c, c, c, 3)$

s_4 : $(c, c, d, 3)$

s_5 : $(d, c, d, 3)$

s_6 : $(d, d, d, 3)$

s_7 : $(d, c, c, 3)$

Using the introduced percept notation, determine the following:

- PERCEPT(s_0)
- PERCEPT(s_1)
- PERCEPT(s_7)
- POSSIBLE-PERCEPTS($\{s_0, s_2, s_3, s_4\}$)
- UPDATE($\{s_0, s_2, s_3, s_4\}$, `dirty`@3)
- RESULTS($\{s_2, s_4, s_5, s_6\}$, `Left`)

SOLUTION.

- PERCEPT(s_0) = `clean@3`.
- PERCEPT(s_1) = `dirty@2`.
- PERCEPT(s_7) = `clean@3`.
- POSSIBLE-PERCEPTS($\{s_0, s_2, s_3, s_4\}$) = {`clean@3`, `dirty@3`}.
- UPDATE($\{s_0, s_2, s_3, s_4\}$, `dirty`@3) = $\{s_2, s_4\}$.
- RESULTS($\{s_2, s_4, s_5, s_6\}$, `Left`) = $\{\{(c, c, d, 2), (d, c, d, 2)\}, \{(c, d, d, 2), (d, d, d, 2)\}\}$.

d. Assume the modelling change so that the robot has a vague sensor which can only determine which of the following two is the case: 1) there is dirt in the current square or an adjacent square; 2) the current square and all adjacent squares are clean. We can model this new situation with only two percepts, `dirty` and `clean`. Explain why. Then determine the same function values as in b, except the percept `dirty@3` is replaced by `dirty`.

SOLUTION. The percept `dirty` covers case 1, where the robot senses dirt in the current or an adjacent square. The percept `clean` covers case 2, where no dirt is sensed (in the current location or any adjacent square).

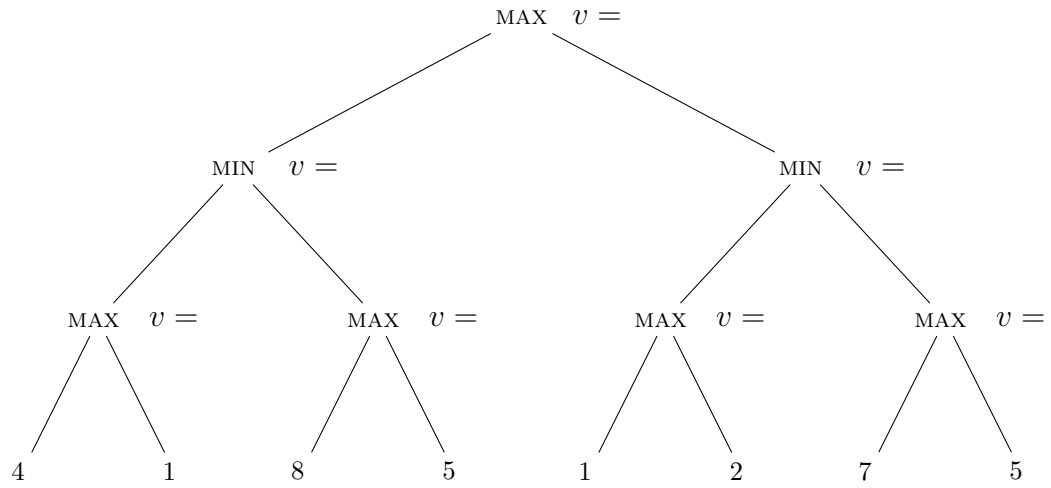
- PERCEPT(s_0) = `dirty`.
- PERCEPT(s_1) = `dirty`.
- PERCEPT(s_7) = `clean`.
- POSSIBLE-PERCEPTS($\{s_0, s_2, s_3, s_4\}$) = {`clean`, `dirty`}.
- UPDATE($\{s_0, s_2, s_3, s_4\}$, `dirty`) = $\{s_0, s_2, s_4\}$.
- RESULTS($\{s_2, s_4, s_5, s_6\}$, `Left`) = $\{\{(c, c, d, 2), (d, c, d, 2), (c, d, d, 2), (d, d, d, 2)\}\}$.

02180 Intro to AI
Exercises for week **6**

Exercise 3

This is an exam exercise from the course in spring 2018.

1. Consider the game tree below with MAX and MIN nodes explicitly marked. Add the MINIMAX value of each node to the game tree (after "$v =$"). Then highlight the edge corresponding to the best move of MAX in the initial state of the game.



2. Assume ALPHA-BETA-SEARCH (MINIMAX with α-β pruning) is used to explore the game tree. Assume that the children of a node are visited in order from left to right. Mark any cuts in the tree (pruned subtrees) above and, for each, indicate whether it is an α-cut or a β-cut.

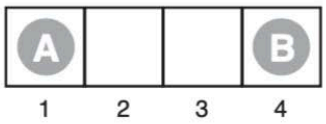


Figure 5.17 The starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in either direction. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space if any. (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, then the value of the game to A is -1.

5.8 Consider the two-player game described in Figure 5.17.

a. Draw the complete game tree, using the following conventions:

- Write each state as (s_A, s_B), where s_A and s_B denote the token locations.
- Put each terminal state in a square box 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 their value is unclear, annotate each with a "?" in a circle.

b. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.

c. 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 (b). Does your modified algorithm give optimal decisions for all games with loops?

d. This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

5.9 This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. The utility function assigns $+1$ to any position with $X_3 = 1$ and -1 to any position with $O_3 = 1$. All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as $Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$.

a. Approximately how many possible games of tic-tac-toe are there?

b. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.

c. Mark on your tree the evaluations of all the positions at depth 2.

d. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.

02180 Intro to AI
Exercises for week 6
SOLUTIONS

Exercise 1

Solve questions (a), (b) and (c) of Exercise 8 in Chapter 5 of R&N: "Consider the two-player game described in Figure 17..."

SOLUTION.

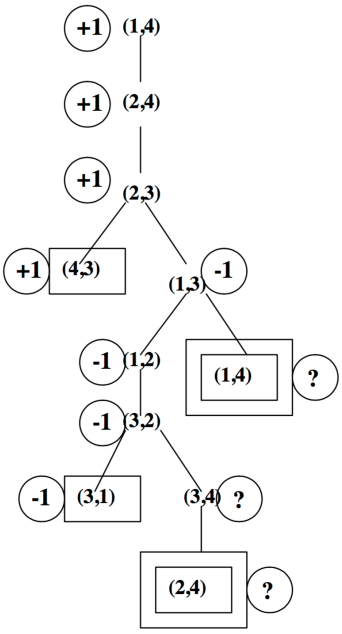
5.8 Consider the two-player game described in Figure 5.17.

a. Draw the complete game tree, using the following conventions:

- Write each state as (s_A, s_B), where s_A and s_B denote the token locations.
- Put each terminal state in a square box 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 their value is unclear, annotate each with a "?" in a circle.

b. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.

c. 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 (b). Does your modified algorithm give optimal decisions for all games with loops?



a)

b) The "?" values are handled by assuming that an agent with a choice between winning the game and entering a "?" state will always choose the win. That is, $min(-1, ?)$ is -1 and $max(+1, ?)$ is $+1$. If all successors are "?", the backed-up value is "?".

c) Standard minimax would fail by going into an infinite loop and never produce values for the loop states. It can be fixed by keeping track of loop states, mark them by "?" and use the revised min and max functions from the previous questions.

The algorithm can not be guaranteed to give optimal decisions for all games with loops. For example, it is not clear how to compare "?" with a draw position: a draw 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.

Exercise 2

Solve questions (a), (b), (c) and (d) of Exercise 9 in Chapter 5 of R&N: "This problem exercises the basic concepts of game playing, using tic-tac-toe..."

SOLUTION.

5.9 This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals

with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. The utility function assigns $+1$ to any position with $X_3 = 1$ and -1 to any position with $O_3 = 1$. All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as $Eval(s) = 3X_2(s)+X_1(s)-(3O_2(s)+O_1(s))$.

a. Approximately how many possible games of tic-tac-toe are there?

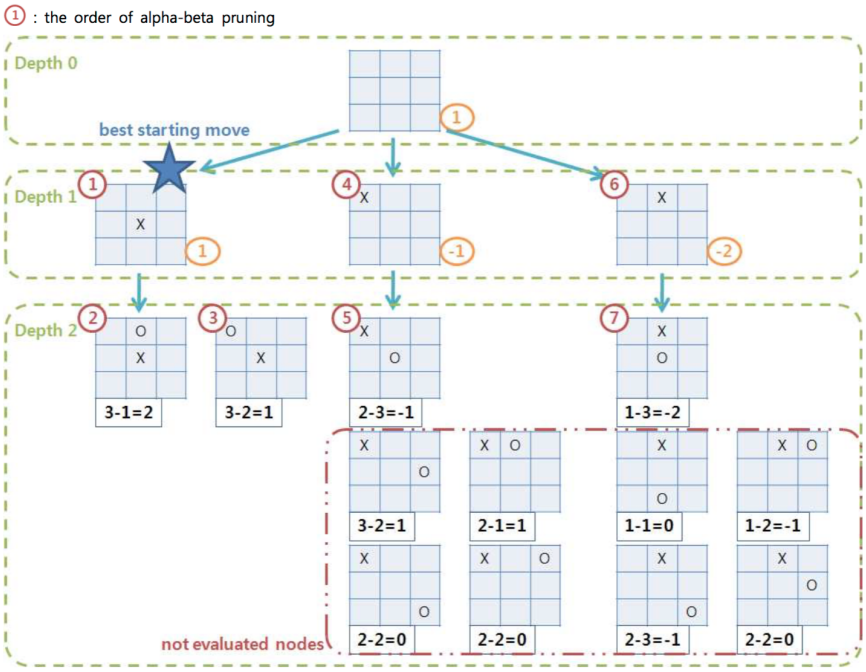
b. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.

c. Mark on your tree the evaluations of all the positions at depth 2.

d. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.

a) There are 9! ways of placing the Xs and Os on the board (ignoring symmetry). So a simple upper bound on the number of games is $9! = 362.880$.

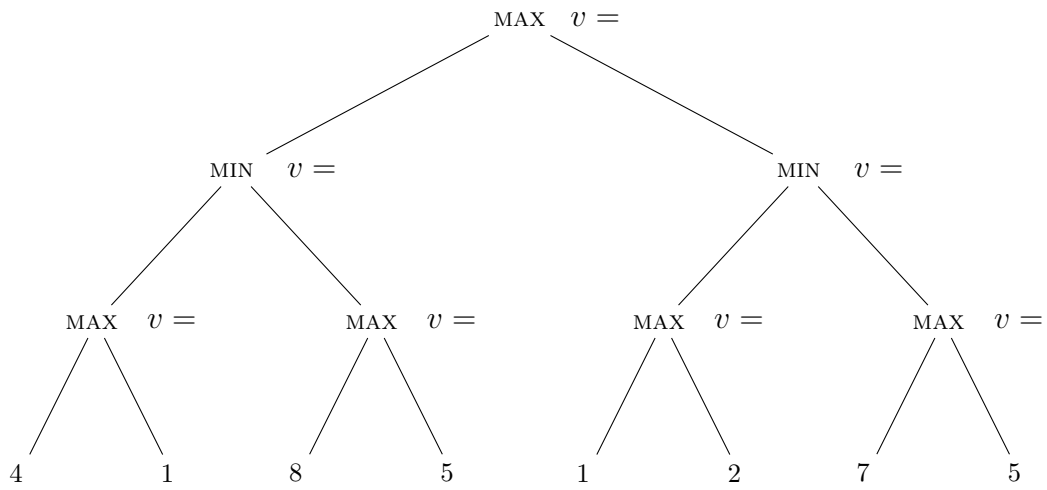
b)



Exercise 3

This is an exam exercise from the course in spring 2018.

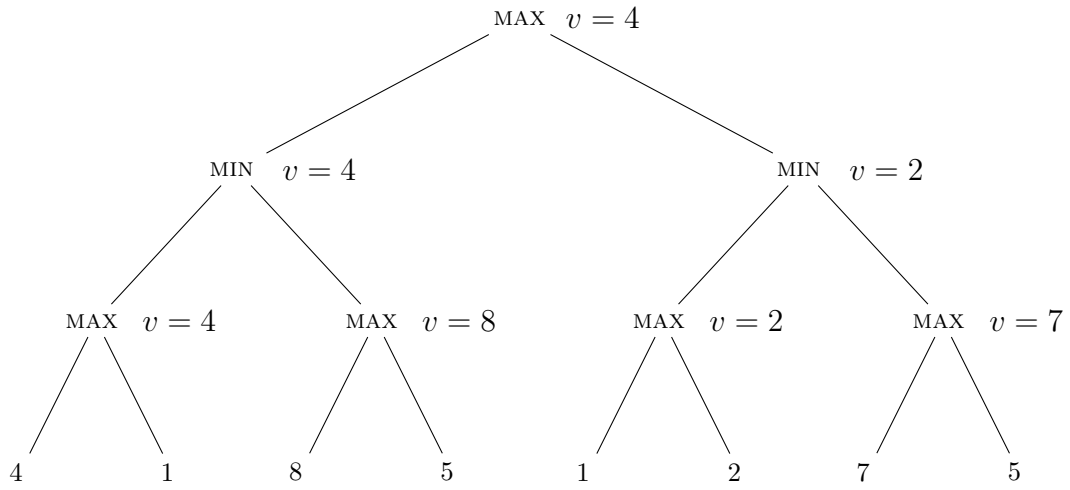
1. Consider the game tree below with MAX and MIN nodes explicitly marked. Add the MINIMAX value of each node to the game tree (after "$v =$"). Then highlight the edge corresponding to the best move of MAX in the initial state of the game.



2. Assume ALPHA-BETA-SEARCH (MINIMAX with α-β pruning) is used to explore the game tree. Assume that the children of a node are visited in order from left to right. Mark any

cuts in the tree (pruned subtrees) above and, for each, indicate whether it is an α-cut or a β-cut.

SOLUTION.



The edge from the root to the left child should be highlighted. There are two cuts: a β-cut to the leftmost leaf with value 5. There is an α-cut on the rightmost MAX-subtree of depth 1.

Introduction to AI
Exercises at week 8

Ex. 1 True or false?

1. $\top \models \bot$
2. $\bot \models \top$
3. $p \wedge q \models p \leftrightarrow q$
4. $p \leftrightarrow q \models p \wedge q$
5. $p \leftrightarrow q \models \neg p \wedge q$
6. $(p \vee q) \wedge (\neg r \vee \neg s \vee t) \models (p \vee q \vee r) \wedge (q \vee \neg r \vee s \rightarrow t)$
7. $(p \vee q) \wedge \neg(p \rightarrow q)$ is satisfiable
8. $(p \leftrightarrow q) \wedge (\neg p \vee q)$ is satisfiable
9. $(p \leftrightarrow q) \leftrightarrow r$ has the same number of models as $(p \leftrightarrow q)$ for any fixed set of proposition symbols that includes p, q, r.

Ex. 2 How many models are there for the following sentences (assuming we only have p, q, r, s in the vocabulary)?

1. $q \vee r$
2. $\neg p \vee \neg q \vee \neg r \vee \neg s$
3. $(p \rightarrow q) \wedge p \wedge \neg q \wedge r \wedge s$

Ex. 3 We have defined four binary logical connectives.

1. Are there others that can be useful?
2. How many logical connectives can there be?
3. Why are some of them not very useful?

Ex. 4 Decide for each of the following, is it valid, unsatisfiable or neither?

1. $Smoke \rightarrow Smoke$
2. $Smoke \rightarrow Fire$
3. $(Smoke \rightarrow Fire) \rightarrow (\neg Smoke \rightarrow \neg Fire)$
4. $Smoke \vee Fire \vee \neg Fire$
5. $Big \vee Long \vee (Big \rightarrow Long)$
6. $(Big \wedge Long) \vee \neg Long$

Ex. 5 (EXTRA exercise, to recall truth-tables, if needed) For each of the following formulas decide: are they tautologies? are they satisfiable?

1. $p \vee \neg p$
2. $p \wedge \neg p$
3. $(p \wedge q) \rightarrow p$
4. $(p \wedge q) \rightarrow \neg p$
5. $(p_1 \wedge p_2) \rightarrow p_3) \rightarrow (p_2 \rightarrow (p_1 \rightarrow p_3))$
6. $(((p \wedge q) \rightarrow s) \wedge ((p \wedge q) \rightarrow t)) \rightarrow ((p \wedge q) \rightarrow (s \wedge t))$

Introduction to AI: Solutions to Logic Exercises week 08

March 23rd, 2021

Ex. 1 True or false?

1. $\top \models \bot$
2. $\bot \models \top$
3. $p \wedge q \models p \leftrightarrow q$
4. $p \leftrightarrow q \models p \wedge q$
5. $p \leftrightarrow q \models \neg p \wedge q$
6. $(p \vee q) \wedge (\neg r \vee \neg s \vee t) \models (p \vee q \vee r) \wedge (q \vee \neg r \vee s \rightarrow t)$
7. $(p \vee q) \wedge \neg(p \rightarrow q)$ is satisfiable
8. $(p \leftrightarrow q) \wedge (\neg p \vee q)$ is satisfiable
9. $(p \leftrightarrow q) \leftrightarrow r$ has the same number of models as $(p \leftrightarrow q)$ for any fixed set of proposition symbols that includes p, q, r.

Answers:

1. false
2. true
3. true
4. false
5. false
6. false
7. true
8. true
9. true

Ex. 2 How many models are there for the following sentences (assuming we only have p, q, r, s in the vocabulary)?

1. $q \vee r$
2. $\neg p \vee \neg q \vee \neg r \vee \neg s$
3. $(p \rightarrow q) \wedge p \wedge \neg q \wedge r \wedge s$

Answers:

1. 12
2. 15
3. 0

Ex. 3 We have defined four binary logical connectives.

1. Are there others that can be useful?
2. How many logical connectives can there be?
3. Why are some of the not very useful?

Answers:

1. Yes, for instance the familiar connective XOR (exclusive or). It has multiple uses in computer science and in particular in AI.
2. As many as there are boolean functions of two arguments, i.e., $2^{2^2} = 16$.
3. It depends on the context of course. For instance, the NAND operator (Sheffer stroke, $p|q$, meaning not-conjunction) is functionally complete, i.e., it's sole use allows expressing any other possible connective. But then the formulas can get very long and unintuitive.

Ex. 4 Decide for each of the following, is it valid, unsatisfiable or neither?

1. $Smoke \rightarrow Smoke$
2. $Smoke \rightarrow Fire$
3. $(Smoke \rightarrow Fire) \rightarrow (\neg Smoke \rightarrow \neg Fire)$
4. $Smoke \vee Fire \vee \neg Fire$
5. $Big \vee Long \vee (Big \rightarrow Long)$
6. $(Big \wedge Long) \vee \neg Long$

Answers:

1. valid
2. neither
3. neither
4. valid
5. valid
6. neither

Introduction to AI
Exercises at week 10

part a

Ex. 1 Consider the following knowledge base $KB = \{\neg p \to q, q \to p, p \to r \wedge s\}$. Decide if the formula $p \wedge r \wedge s$ follows from KB. Try using each of the following methods: truth tables, Modus Ponens only, resolution.

Ex. 2 Come up with a propositional logic formula which uses only implication and negation, and which can not be represented as a Horn clause.

Ex. 3 (7.11 and 7.18 a from the handbook) Any propositional logic sentence is logically equivalent to the assertion that:

- each possible world in which it would be false is not the case;
- some possible world in which it would be true is in fact the case.

From those prove that any sentence can be written in CNF and in DNF.

Ex. 4 Construct an algorithm that converts any sentence in propositional logic into DNF.

Ex. 5 There are three suspects for murder: Jørgen, Thomas, and Nina. Jørgen, says 'I didn't do it. The victim was old acquaintance of Thomas'. But Nina hated him.' Thomas states 'I didn't do it. I didn't know the guy. Besides I was out of town all the week.' Nina says 'I didn't do it. I saw both Jørgen and Thomas downtown with the victim that day; one of them must have done it.' Assume that the two innocent people are telling the truth, but that the guilty might not be. Write out the facts as sentences in Propositional Logic, and use propositional resolution to solve the crime.

Ex. 6 (7.19 from the book) Convert the following set of sentences to clausal form.

- $A \leftrightarrow (C \vee E)$
- $E \to D$
- $B \wedge F \to \neg C$
- $E \to C$
- $C \to F$
- $C \to B$

Give a trace of the execution of DPLL on the conjunction of these clauses.

Introduction to AI: Solutions to Logic Exercises week 10

part a

Ex. 1 Consider the following knowledge base $KB = \{\neg p \to q,\ q \to p,\ p \to r \wedge s\}$. Decide if the formula $p \wedge r \wedge s$ follows from KB. Try using each of the following methods: truth tables, Modus Ponens only, resolution.

Answer: **Truth tables** approach draw a truth table for four variables p, q, r, s, compute the truth values under all assignments fot the formulas in KB and for the goal formula. Check if in all rows in which all the formulas in KB are true, the goal formula is also true. If that is so, the goal formula follows from KB, otherwise it does not. **Modus Ponens** method is not applicable since the goal formula does not appear as a consequent of an implication formula in any of the elements of KB. **Resolution** method is used in the following way. Proof by refutation:

1. $A \vee B$ premise
2. $\neg B \vee A$ premise
3. $\neg A \vee C$ premise
4. $\neg A \vee D$ premise
5. $\neg A \vee \neg C \vee \neg D$ negated thesis
6. A resolution 1, 2
7. C resolution 3, 6
8. D resolution 4, 6
9. $\neg C \vee \neg D$ resolution 5, 6
10. $\neg D$ resolution 7, 9
11. $\{\}$ resolution 8, 10

Ex. 2 Come up with a propositional logic formula which uses only implication and negation, and which can not be represented as a Horn clause.

A possible answer: $(\neg p \to q) \to \neg r$.

Ex. 3 (7.11 and 7.18 a from the handbook) Any propositional logic sentence is logically equivalent to the assertion that:

- each possible world in which it would be false is not the case;
- some possible world in which it would be true is in fact the case.

From those prove that any sentence can be written in CNF and in DNF. Then keep app

Answer: Each possible world can be written as a conjunction of literals, e.g. $(A \wedge B \wedge \neg C)$. Asserting that a possible world is not the case can be written by negating that, e.g. $\neg(A \wedge B \wedge \neg C)$, which can be rewritten as $(\neg A \vee \neg B \vee C)$. This is the form of a clause; a conjunction of these clauses is a CNF sentence, and can list the negations of all the possible worlds that would make the sentence false.

Each possible world can be expressed as the conjunction of all the literals that hold in the model. The sentence is then equivalent to the disjunction of all these conjunctions, i.e., a DNF expression.

Ex. 4 Construct an algorithm that converts any sentence in propositional logic into DNF.

Short answer: convert to negation normal form with De Morgan laws then distribute AND over OR.

Ex. 5 There are three suspects for murder: Jørgen, Thomas, and Nina. Jørgen, says 'I didn't do it. The victim was old acquaintance of Thomas'. But Nina hated him.' Thomas states 'I didn't do it. I didn't know the guy. Besides I was out of town all the week.' Nina says 'I didn't do it. I saw both Jørgen and Thomas downtown with the victim that day; one of them must have done it.' Assume that the two innocent people are telling the truth, but that the guilty might not be. Write out the facts as sentences in Propositional Logic, and use propositional resolution to solve the crime.

Answer: Convert to propositional logic and look for contradictions:

- Jørgen says: $\neg M_J \wedge K_T \wedge H_N$
- Thomas says: $\neg M_T \wedge \neg K_T \wedge \neg I_T$
- Nina says: $\neg M_N \wedge I_J \wedge I_T$

where J, T and N denote Jørgen, Thomas and Nina respectively. M_X denotes that X is the murderer, K_X that X knew the victim, H_X that X hated the victim and I_X that X was in town.

Jørgen says that Thomas knew the victim while Thomas says he did not (contradiction), thus one of them must be lying. Similarly, Nina says that Thomas was in town while Thomas says he wasn't (contradiction), and one of them must be lying as well. Assuming that the two innocent people are telling the truth, Thomas must be the murderer.

Ex. 6 (7.19 from the book) Convert the following set of sentences to clausal form.

- $A \leftrightarrow (C \vee E)$
- $E \rightarrow D$
- $B \wedge F \rightarrow \neg C$
- $E \rightarrow C$
- $C \rightarrow F$
- $C \rightarrow B$

Give a trace of the execution of DPLL on the conjunction of these clauses.

Answer:

- $(\neg A \vee C \vee E) \wedge (\neg C \vee A) \wedge (\neg E \vee A)$
- $(\neg E \vee D)$
- $(\neg B \vee \neg F \vee \neg C)$
- $(\neg E \vee C)$
- $(\neg C \vee F)$
- $(\neg C \vee B)$

Introduction to AI
Exercise at week 10

April 13th, 2021

Introduction Many directions of research in Artificial Intelligence proceed by drawing parallels with human cognitive ability, matching and, possibly, surpassing it. In this sense Artificial Intelligence and Cognitive Science are tightly knit together. The research into cognitive aspect of problem-solving often involves constructing a *cognitive model*, which can be viewed as an AI engine. In such cases however, the priority of efficient design, so common in AI engineering, are sacrificed for more faithful representation of human reasoning. Moreover, human behaviour often displays erratic and non-logical patterns, which both Cognitive Science and Artificial Intelligence often need to account for. In this exercise session we invite you to analyse the well-known game of Mastermind in terms of logical reasoning involved in the game. One of the take-away messages from this exercise, apart from practising the techniques introduced in the course, is that finding an efficient or correct solution to the problem at hand (so-called *normative perspective*), can be drastically different from human experience of attempting to solve a problem (so-called *descriptive perspective*).

Mastermind is a code-breaking game for two players. It consists of a decoding board, code pegs of k colors, and feedback pegs of red and white (see Figure 1). There are two players, the

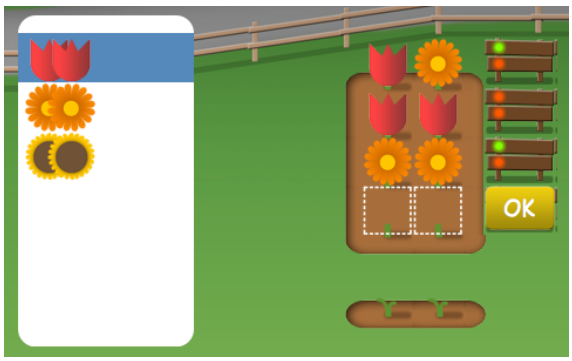
Figure 1: The modern Mastermind Game game with pegs was invented in 1970 by Mordecai Meirowitz, but the game resembles an earlier pen and paper game called Bulls and Cows. The figure shows the board version of Mastermind game as it is known today (source: Wikipedia).

code-maker, who chooses a secret pattern of ℓ code pegs (color duplicates are allowed), and the code-breaker, who guesses the pattern in a given n rounds. Each round consists of the code-breaker making a guess by placing a row of ℓ code pegs, and of the code-maker providing the feedback: a red peg for each code peg of correct color and position, and a white peg for each peg

of correct color but wrong position. Guesses and feedbacks continue to alternate until either the code-breaker guesses correctly, or n incorrect guesses have been made. The code-breaker wins if she obtains the solution within n rounds; the code-maker wins otherwise.

Mastermind is an *inductive inquiry* game that involves *trials of experimentation and evaluation*. The game has been used to investigate the acquisition of complex skills and strategies in the domain of reasoning about others [6]. Existing mathematical results on Mastermind focus on finding strategies that allow winning the game in the smallest number of rounds [see 2, 3, 4, 5]. The deductive reasoning processes involved in Mastermind has been studied in [1]. I also upload the **paper and slides of my lecture in Discrete Mathematics Course 2018**, for those interested in how to AI engine designed to solve the game can be used to draw predictions and explanations of human behaviour.

In this exercise you are invited to consider a simpler version of the Mastermind game, called Deductive Mastermind, proposed in [1]. Figure shows an example of a Deductive Mastermind game. It consists of a decoding board and the domain of flowers to choose from while constructing the solution. The goal of the game is to guess the correct sequence of flowers on the basis of the clues, consisting of earlier flower-sequences with their corresponding feedbacks given on the decoding board, upfront. Thus each row of flowers is accompanied by a feedback on the small board on the right side: one green dot for each flower of the correct color and position, one orange dot for each flower of correct color but in a wrong position, and one red dot for each flower that does not appear in the correct secret sequence at all.



Ex. 1 What is the solution to this game? Give an appropriate sequence of flowers.

Ex. 2 Show, using the resolution technique, that your solution indeed follows from the premises. Hint: Translate each pair (conjecture, feedback) and your solution into propositional logic formulas, using propositions such as s_1 := 'sunflower in the first position'. Convert each of the resulting formulas into their CNF-form. Resolve.

Ex. 3 [Open Question] Reflect on how the resolution-based method for verifying the solution differed from your initial way of solving the problem. What methods would you use to implement your introspective method of solving the problem?

Ex. 4 [*] Describe a general procedure for translating any flower-sequence with feedback, for an arbitrary number of flowers k and arbitrary number of positions ℓ in the code. (Remember about the orange feedback, absent in the previous exercise!)

References

[1] Nina Gierasimczuk, Han L. J. van der Maas, and Maartje E. J. Raijmakers. An analytic tableaux model for deductive mastermind empirically tested with a massively used online learning system. *Journal of Logic, Language and Information*, 22(3):297–314, Jul 2013.

[2] R. W. Irving. Towards an optimum Mastermind strategy. *Journal of Recreational Mathematics*, 11:81–87, 1978.

[3] Donald E. Knuth. The computer as master mind. *Journal of Recreational Mathematics*, 9(1):1–6, 1977.

[4] Barteld Kooi. Yet another Mastermind strategy. *ICGA Journal*, 28(1):13–20, 2005.

[5] Mami Koyama and Tony Lai. An optimal Mastermind strategy. *Journal of Recreational Mathematics*, 25:251–256, 1993.

[6] R. Verbrugge and L. Mol. Learning to apply theory of mind. *Journal of Logic, Language and Information*, 17(4):489–511, 2008.

Introduction to AI
Exercise at week 10

part b - solution

Introduction Many directions of research in Artificial Intelligence proceed by drawing parallels with human cognitive ability, matching and, possibly, surpassing it. In this sense Artificial Intelligence and Cognitive Science are tightly knit together. The research into cognitive aspect of problem-solving often involves constructing a *cognitive model*, which can be viewed as an AI engine. In such cases however, the priority of efficient design, so common in AI engineering, are sacrificed for more faithful representation of human reasoning. Moreover, human behaviour often displays erratic and non-logical patterns, which both Cognitive Science and Artificial Intelligence often need to account for. In this exercise session we invite you to analyse the well-known game of Mastermind in terms of logical reasoning involved in the game. One of the take-away messages from this exercise, apart from practising the techniques introduced in the course, is that finding an efficient or correct solution to the problem at hand (so-called *normative perspective*), can be drastically different from human experience of attempting to solve a problem (so-called *descriptive perspective*).

Mastermind is a code-breaking game for two players. It consists of a decoding board, code pegs of k colors, and feedback pegs of red and white (see Figure 1). There are two players, the

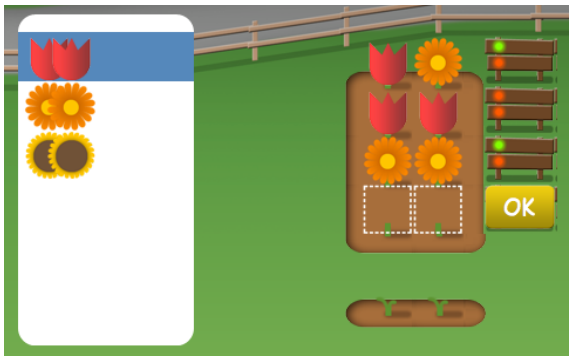
Figure 1: The modern Mastermind Game game with pegs was invented in 1970 by Mordecai Meirowitz, but the game resembles an earlier pen and paper game called Bulls and Cows. The figure shows the board version of Mastermind game as it is known today (source: Wikipedia).

code-maker, who chooses a secret pattern of ℓ code pegs (color duplicates are allowed), and the code-breaker, who guesses the pattern in a given n rounds. Each round consists of the code-breaker making a guess by placing a row of ℓ code pegs, and of the code-maker providing the feedback: a red peg for each code peg of correct color and position, and a white peg for each peg

of correct color but wrong position. Guesses and feedbacks continue to alternate until either the code-breaker guesses correctly, or n incorrect guesses have been made. The code-breaker wins if she obtains the solution within n rounds; the code-maker wins otherwise.

Mastermind is an *inductive inquiry* game that involves *trials of experimentation and evaluation*. The game has been used to investigate the acquisition of complex skills and strategies in the domain of reasoning about others [6]. Existing mathematical results on Mastermind focus on finding strategies that allow winning the game in the smallest number of rounds [see 2, 3, 4, 5]. The deductive reasoning processes involved in Mastermind has been studied in [1]. I also upload the **paper and slides of my lecture in Discrete Mathematics Course 2018**, for those interested in how to AI engine designed to solve the game can be used to draw predictions and explanations of human behaviour.

Ex. 1 In this exercise you are invited to consider a simpler version of the Mastermind game, called Deductive Mastermind, proposed in [1]. Figure shows an example of a Deductive Mastermind game. It consists of a decoding board and the domain of flowers to choose from while constructing the solution. The goal of the game is to guess the correct sequence of flowers on the basis of the clues, consisting of earlier conjecture with feedback given on the decoding board. Thus each row of flowers forms a conjecture that is accompanied by a feedback on the small board on the right side: one green dot for each flower of the correct color and position, one orange dot for each flower of correct color but in a wrong position, and one red dot for each flower that does not appear in the correct secret sequence at all.



(a) What is the solution to this game? Give an appropriate sequence of flowers.

Answer: Sunflower followed by a daisy.

(b) Show, using the resolution technique, that your solution indeed follows from the premises. Hint: Translate each pair (conjecture, feedback) and your solution into propositional logic formulas, using propositions such as s_1 := 'sunflower in the first position'. Convert each of the resulting formulas into their CNF-form. Resolve.

Answer: First assign propositional symbols to each atomic fact, for instance: s_1 for the sunflower in the first position, t_2 for the tulip in the second position, etc. We take d for the orange daisy.

Our knowledge base must contain the basic assumptions of the game (background knowledge): (a) there must be one and (b) can only be one flower in a given position.

(a1) $(t_1 \vee s_1 \vee d_1)$

(a2) $(t_2 \vee s_2 \vee d_2)$

(b1) $(\neg t_1 \vee \neg s_1) \wedge (\neg s_1 \vee \neg d_1) \wedge (\neg d_1 \vee \neg t_1)$

(b2) $(\neg t_2 \vee \neg s_2) \wedge (\neg s_2 \vee \neg d_2) \wedge (\neg d_2 \vee \neg t_2)$

Then we formalise premises given in the game:

1. $(t_1 \wedge \neg d_2) \vee (d_2 \wedge \neg t_1) \equiv (t_1 \vee d_2) \wedge (\neg d_2 \vee \neg t_1)$
2. $\neg t_1 \wedge \neg t_2$
3. $(d_1 \wedge \neg d_2) \vee (d_2 \wedge \neg d_1) \equiv (d_1 \vee d_2) \wedge (\neg d_2 \vee \neg d_1)$

The conclusion is:

4. $(s_1 \wedge d_2)$

Our KB is now consists now of the formulas in (a), (b) and (1-3). In other to check if $KB \models (4)$, we add it's negation, (4'): $\neg(s_1 \wedge d_2)$, to KB. Was your answer entailed by the premises?

Next, we transform the formulas in KB into their respective CNFs, obtaining the following set of clauses:

(a1) $t_1 \vee s_1 \vee d_1$

(a2) $t_2 \vee s_2 \vee d_2$

(b11) $\neg t_1 \vee \neg s_1$

(b12) $\neg s_1 \vee \neg d_1$

(b13) $\neg d_1 \vee \neg t_1$

(b21) $\neg t_2 \vee \neg s_2$

(b22) $\neg s_2 \vee \neg d_2$

(b23) $\neg d_2 \vee \neg t_2$

1.1 $t_1 \vee d_2$

1.2 $\neg d_2 \vee \neg t_1$

2.1 $\neg t_1$

2.2 $\neg t_2$

3.1 $d_1 \vee d_2$

3.2 $\neg d_2 \vee \neg d_1$

4'. $\neg s_1 \vee \neg d_2$

One possible resolution sequence is:

$$r(r(r(2.1, 1.1), 4'), r(r(r(2.1, 1.1), 3.2), r(2.1, a1)))$$

(c) [Open Question] Reflect on how the resolution-based method for solving the problem differed from your initial way of solving the problem. What methods would you use to implement your introspective method of solving the problem?

(d) [*] Describe a general procedure for translating any pair $(conjecture, feedback)$ for an arbitrary number of flowers k and arbitrary number of positions ℓ in the code. (Remember about the orange feedback, absent in the previous exercise!)

Answer: Each DMM game consists of a sequence of conjectures.

A *conjecture* of length ℓ (consisting of ℓ pins) over k colors is any sequence given by a total assignment, $h : \{1, \ldots, \ell\} \to \{c_1, \ldots, c_k\}$. The *goal sequence* is a distinguished conjecture, $goal : \{1, \ldots, \ell\} \to \{c_1, \ldots, c_k\}$.

Every non-goal conjecture is accompanied by a feedback that indicates how similar h is to the given goal assignment. The three feedback colors, green, orange, and red will be represented by letters g, o, and r.

Let h be a conjecture and let *goal* be the goal sequence, both of length ℓ over k colors. The *feedback f for h with respect to goal* is a sequence

$$\overbrace{g \ldots g}^{a} \; \overbrace{o \ldots o}^{b} \; \overbrace{r \ldots r}^{c} = g^a o^b r^c,$$

where $a, b, c \in \{0, 1, 2, 3, \ldots\}$ and $a + b + c = \ell$. The feedback consists of:

- exactly one g for each $i \in G$, where $G = \{i \in \{1, \ldots, \ell\} \mid h(i) = goal(i)\}$.
- exactly one o for every $i \in O$, where

  $O = \{i \in \{1, \ldots, \ell\} \backslash G \mid$ there is a $j \in \{1, \ldots, \ell\} \backslash G$, s. t. $i \neq j$ and $h(i) = goal(j)\}$.
- exactly one r for every $i \in \{1, \ldots, \ell\} \backslash (G \cup O)$.

As literals of our Boolean formulae we take $h(i) = goal(j)$, where $i, j \in \{1, \ldots \ell\}$. They can be viewed as propositional variables $p_{i,j}$, for $i, j \in \{1, \ldots \ell\}$. With respect to sets G, O, and R that induce a partition of $\{1, \ldots, \ell\}$, we define $\varphi^g_G, \varphi^o_{G,O}, \varphi^r_{G,O}$, the propositional formulae that correspond to different parts of the feedback, in the following way:

- $\varphi^g_G := \bigwedge_{i \in G} h(i) {=} goal(i) \wedge \bigwedge_{j \in \{1, \ldots, \ell\} \backslash G} h(j) {\neq} goal(j)$,
- $\varphi^o_{G,O} := \bigwedge_{i \in O} (\bigvee_{j \in \{1, \ldots, \ell\} \backslash G, i \neq j} h(i) = goal(j))$,
- $\varphi^r_{G,O} := \bigwedge_{i \in \{1, \ldots, \ell\} \backslash (G \cup O), j \in \{1, \ldots \ell\} \backslash G, i \neq j} h(i) \neq goal(j)$.

Observe that there will be as many substitutions of each of the above schemes of formulae, as there are ways to choose the corresponding sets G and O. To fix the domain of those possibilities we set $\mathbb{G} := \{G | G \subseteq \{1, \ldots, \ell\} \wedge card(G) {=} a\}$, and, if $G \subseteq \{1, \ldots, \ell\}$, then $\mathbb{O}^G = \{O | O \subseteq \{1, \ldots, \ell\} \backslash G \wedge card(O) {=} b\}$. Finally, we can set $Bt(h, f)$, the Boolean translation of (h, f), to be given by:

$$Bt(h, f) := \bigvee_{G \in \mathbb{G}} (\varphi^g_G \wedge \bigvee_{O \in \mathbb{O}^G} (\varphi^o_{G,O} \wedge \varphi^r_{G,O})).$$

References

[1] Nina Gierasimczuk, Han L. J. van der Maas, and Maartje E. J. Raijmakers. An analytic tableaux model for deductive mastermind empirically tested with a massively used online learning system. *Journal of Logic, Language and Information*, 22(3):297–314, Jul 2013.

[2] R. W. Irving. Towards an optimum Mastermind strategy. *Journal of Recreational Mathematics*, 11:81–87, 1978.

[3] Donald E. Knuth. The computer as master mind. *Journal of Recreational Mathematics*, 9(1):1–6, 1977.

[4] Barteld Kooi. Yet another Mastermind strategy. *ICGA Journal*, 28(1):13–20, 2005.

[5] Mami Koyama and Tony Lai. An optimal Mastermind strategy. *Journal of Recreational Mathematics*, 25:251–256, 1993.

[6] R. Verbrugge and L. Mol. Learning to apply theory of mind. *Journal of Logic, Language and Information*, 17(4):489–511, 2008.

Introduction to AI
Exercises at week 12

Ex. 1 Assume Bob's belief set $B = Cn(\{p, p \leftrightarrow q, \neg r\})$. Come up with an appropriate prior plausibility order on W (the set of all possible truth assignments over p, q, r), which will satisfy both of the requirements below:

1. after revision with r Bob would believe that $\neg q$;
2. after contraction with $p \rightarrow q$ Bob would believe that p.

Ex. 2 Let $A = \{p, q, p \wedge q, p \vee q, p \rightarrow q\}$ be a belief base. Which of the following sets are elements of $A \perp q$?

set of formulas	yes	no
$\{p, p \vee q\}$		
$\{p \rightarrow q\}$		
$\{p \vee q, p \rightarrow q\}$		
$\{p \vee q\}$		

Ex. 3 Give an example (different than the one presented in Lecture 11) of two belief bases that are statically equivalent (i.e., they generate the same belief sets), but are not dynamically equivalent.

Introduction to AI
Solutions to exercises at week 12

Ex. 1 Assume Bob's belief set $B = Cn(\{p, p \leftrightarrow q, \neg r\})$. Come up with an appropriate prior plausibility order on W (the set of all possible truth assignments over p, q, r) which will satisfy the given requirements.

Solution:

p, q, r	$p, q, \bar{r}$	$p, \bar{q}, r$	$p, \bar{q}, \bar{r}$	$\bar{p}, q, r$	$\bar{p}, q, \bar{r}$	$\bar{p}, \bar{q}, r$	$\bar{p}, \bar{q}, \bar{r}$
a			d	e	f	g	h
		c					
	b						

Table 1: An example of a prior plausibility

1. After revision with r Bob would believe that $\neg q$: after revision with r the minimal world becomes c, and $\neg q$ is true in c.
2. After contraction with $p \rightarrow q$ Bob would believe that p: after contraction with $p \rightarrow q$, c is the minimal world which does not satisfy $p \rightarrow q$, since p is true in c and q is false in c. Note that, $p \in B \div (p \rightarrow q)$ because p is true in both b and c.

Ex. 2 Let $A = \{p, q, p \wedge q, p \vee q, p \rightarrow q\}$ be a belief base. Which of the following sets are elements of $A \perp q$?

Solution:

set of formulas	yes	no
$\{p, p \vee q\}$	x	
$\{p \rightarrow q\}$	x	
$\{p \vee q, p \rightarrow q\}$		x
$\{p \vee q\}$		x

Ex. 3 Give an example (different than the one presented in Lecture 11) of two belief bases that are statically equivalent (i.e., they generate the same belief sets), but are not dynamically equivalent.

Solution: For instance, two belief bases $A = \{p, p \rightarrow q\}$ and $B = \{p, q\}$. $Cn(A) = Cn(B)$, but $Cn(A * \neg p) \neq Cn(B * \neg p)$.