# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

# Week 2 Tutorial Solutions

**This page was last updated: 03/17/2018 09:51:27**

---

**Activitity 2.1:** PEAS Descriptions

---

Present to your tutorial group the PEAS model you developed for the activity from "The PEAS Model of an Agent".

---

**Activitity 2.2:** Classifying Environments & Choice of Agent Type

---

Use the PEAS model to classify an environment from the tutorial in Week 1.

1. Classify each task according to the properties given in lectures in the Classifying Environments page of Week 2 module. Give a one-line justification for each choice.
2. Select a suitable agent type (or, discuss the relative merits of the different agent types presented).

---

**Activitity 2.3** Sorting in Prolog

---

1. Write a prolog predicate `insert(Num, List, NewList)` that takes a number `Num` along with a list of numbers `List` which is already sorted in increasing order, and binds `NewList` to the list obtained by inserting `Num` into `List` so that the resulting list is still sorted in increasing order.

```
% Base Case
insert(New,[],[New]).

% New node goes at front of list
insert(New,[Head|Tail],[New,Head|Tail]) :-
    New =< Head.

% New node is inserted into existing list
 insert(New,[Head|Tail],[Head|Tail1]) :-
     New > Head,
     insert(New,Tail,Tail1).
```

13. Write a predicate `isort(List,NewList)` that takes a List of numbers in any order, and binds `NewList` to the list containing the same numbers sorted in increasing order. Hint: your predicate should call the `insert()` predicate from part (1).

```
 % Base Case
```

```
15.  isort([],[]).
16.
17.  % Sort the Tail and then insert the Head
18.  isort([Head|Tail],List) :-
19.     isort(Tail,Tail_Sorted),
20.     insert(Head,Tail_Sorted,List).
```

21. Write a predicate `split(BigList,List1,List2)` which takes a list `BigList` and divides the items into two smaller lists `List1` and `List2`, as evenly as possible (i.e. so that the number of items in `List1` and `List2` differs by no more that 1). Can it be done without measuring the length of the list?

```
22.  % Base Case 0: empty list
23.  split([],[],[]).
24.
25.  % Base Case 1: list with one item
26.  split([A],[A],[]).
27.
28.  % Assign first item to first list, second item to second list
29.  % and recursively split the rest of the list.
30.  split([A,B|T],[A|R],[B|S]) :-
31.     split(T,R,S).
```

32. Write a predicate `merge(Sort1,Sort2,Sort)` which takes two lists `Sort1` and `Sort2` that are already sorted in increasing order, and binds `Sort` to a new list which combines the elements from `Sort1` and `Sort2`, and is sorted in increasing order.

```
33.  % If one list is empty, return the other list
34.  merge(A,[],A).
35.  merge([],B,B).
36.
37.  % If first item of first list is smaller,
38.  % it becomes first item of the merged list
39.  merge([A|R],[B|S],[A|T]) :-
40.     A =< B,
41.     merge(R,[B|S],T).
42.
43.  % If first item of second list is smaller,
44.  % it becomes first item of the merged list
45.  merge([A|R],[B|S],[B|T]) :-
46.     A > B,
47.     merge([A|R],S,T).
```

Bonus Challenge (for students who have already studied sorting algorithms in another programming language):

Write a predicate `mergesort(List,NewList)` which has the same functionality as the `isort()` predicate from part (2) above, but uses the MergeSort algorithm. Hint: you will need to use the `split()` and `merge()` predicates from parts (3) and (4) above.

```
% Base Cases: empty list or list with one item
mergesort([],[]).
mergesort([A],[A]).

% If the list has more than one item,
% split it into two lists of (nearly) equal size,
% sort the two smaller lists, and merge them.
mergesort([A,B|T],S) :-
   split([A,B|T],L1,L2),
```

```
mergesort(L1,S1),
mergesort(L2,S2),
merge(S1,S2,S).
```

---

**Activitity 2.4** (COMP3411/9814 only): Reactive Agents

---

Check that you understand the Identifying Braitenberg Vehicles activity on the Reactive Agents page.

---

# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

## Week 3 Tutorial Solutions

**This page was last updated: 03/17/2018 09:51:38**

---

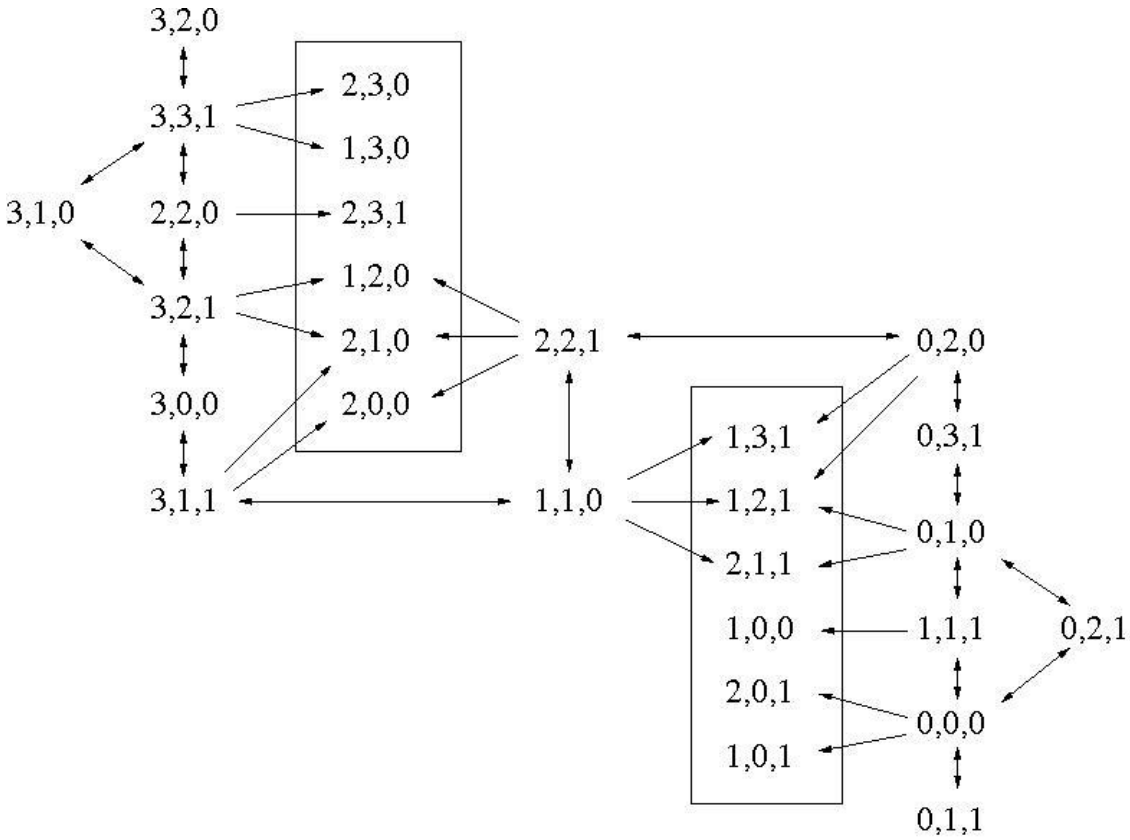
**Activity 3.1** (Exercise 3.9 from Russell & Norvig)

---

The "Missionaries and Cannibals" problem is usually stated as follows: Three missionaries and three cannibals are on one side of the 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. 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). (You can even play the puzzle on-line atwww.learn4good.com/games/puzzle/boat.htm")

1. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution, and draw a diagram of the complete state space.

   Each state can be characterized by listing the number of missionaries (0-3) and cannibals (0-3) on the original side of the river, and 1 or 0 to indicate whether or not the boat is on that side; it is assumed that whatever is not listed must be on the far side of the river. There are 4 x 4 x 2 = 32 states in total, but only 28 of them are accessible from the initial state (3,3,1). Twelve of these represent "dead" states in which one or more missionaries are eaten. Here is a diagram of the complete state space:




2. Solve the problem optimally using an appropriate search algorithm; is it a good idea to check for repeated states?

   There are four ways to get from the initial state (3,3,1) to the final state (0,0,0) in 11 steps. It is definitely a good idea to check for repeated states. Using Breadth-First Search, for example, there are 6 nodes at depth 2 and 25 at depth 3; but, if we avoid expanding previously encountered states, there will only be 2 nodes at depth 2 and 3 at depth 3. Depth-First Search is only able to avoid states which are repeated along the same branch; but the number of nodes is still reduced to 3 at depth 2 and 8 at depth 3.

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

   The step at which people seem to have most difficulty is the one in the centre of the diagram, from (1,1,0) to (2,2,1). Since the objective is to get all 6 people to the far side of the river, it seems counterintuitive to bring two people back to the original side; it violates the "heuristic" of maximizing the total number of people on the far side of the river.

---

**Activity 3.2** Refining Your Understanding of Searches using Mazes

Discuss your findings and insights from the 'Fun with Mazes' activity. Compare your findings and discuss any discrepancies.

a. Generate a random Tree Maze and compare the running time of Breadth First Search and Depth First Search on this maze. Repeat this for a couple of other random Tree Mazes. Time the algorithms with a stopwatch if you can. Which algorithm is faster?

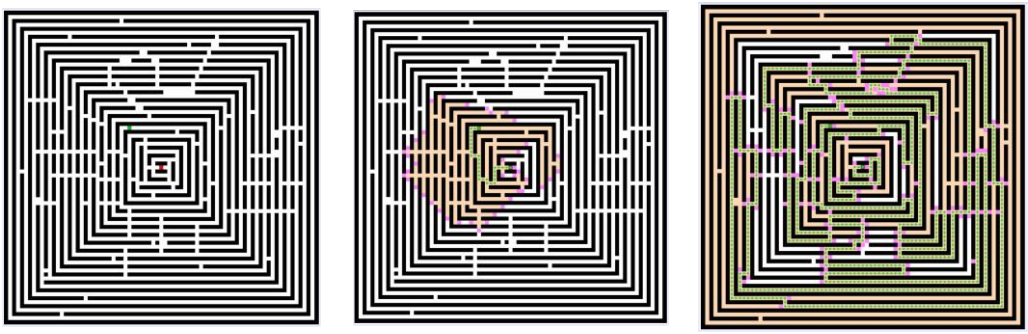
    Generally, BFS is a bit faster.

b. Repeat the steps from part (a), this time with Concentric Graph Mazes. Which algorithm is faster?

    Depth First Search is faster.

c. Use the widget to edit a maze by clicking on the coloured squares, and then clicking on the Maze. Try to design a maze for which BFS finds a solution considerably faster than DFS. You can change the structure of the maze, and specify a starting and ending point anywhere inside the maze. Try to (briefly) explain in words what makes your environment easy for BFS but hard for DFS.

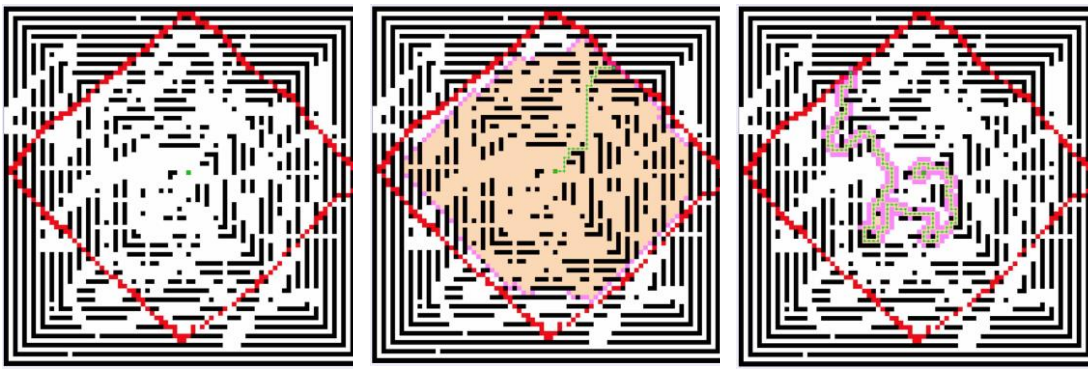
    This would be the case for a fairly open maze, with a single goal which is relatively close to the initial state. BFS will find this goal quickly. DFS may occasionally get lucky, but will often get lost exploring the vast expanses around the goal before finally reaching it.



maze Breadth First Search Depth First Search

d. Use the widget to create a maze for which DFS would find a solution considerably faster than BFS. You should assume that there can be multiple ending points (red squares) and that the algorithm only needs to reach one of them. Try to explain in words what makes your environment easy for DFS but hard for BFS.

This would be true for a search problem where there exist many solutions at a fairly deep level, such that DFS will find a solution more or less as soon as it gets to that depth for the first time. For example, imagine you are shipwrecked in the middle of a lake and you need to find a path to the shoreline, with a series of small steps either up, down, left or right. DFS will find a path quickly, whereas BFS will spend a lot of time exploring all paths shorter than the radius of the lake.



"lake" maze | Breadth First Search | Depth First Search

e. Try running Iterative Deepening Search on a random maze. Why is it so slow?

> Iterative Deepening Search generally performs poorly in situations (like this one) with a low branching factor. Another example is the Missionaries and Cannibals problem. As an extreme example, consider a problem with a branching factor of 1 at every move, and a solution at depth $n$. Depth-First Search will take exactly $n$expansions to find a solution, while Iterative Deepening Search will take $O(n^2)$.

For which type of problem (probably not a maze) would IDS be superior to both BFS and DFS?

> For problems with a larger branching factor (for example, the 8-Puzzle, with branching factor 4), IDS would be superior to both BFS and DFS (in the sense of finding a solution in reasonable time without running out of memory).

---

**Activity 3.3** Further Discussion

---

Any remaining tutorial time should be used for further discussion related to uninformed search strategies, including the Pick a Problem and Map out the State Space activity on the Path Search Problems page; for example:

- Additional Mazes and search strategies
- Fox, Goose and Bag of Beans
- Tower of Hanoi
- Bridge and Torch puzzle
- 5-Puzzle

---

**Answers to Romania Path Search Activities:**

---

- Breadth First Search, to find a path from Arad to Bucharest:

  Arad, Sibiu, Timisoara, Zerind, Fagaras.

- Uniform Cost Search, to find a path from Dobreta to Fagaras:

  Dobreta (0), Mehadia (75), Craiova (120), Lugoj(145), Timisoara (256), Pitesti (258), Rimnicu Vilcea (266), Sibiu (346), Bucharest (359), Arad (374), Urziceni (444), Fagaras (445).

- Depth First Search, to find a path from Dobreta to Zerind:

  Dobreta, Craiova, Pitesti, Bucharest, Fagaras, Sibiu, Arad, Timisoara, Lugoj, Mehadia.

---

# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

## Week 4 Tutorial Solutions

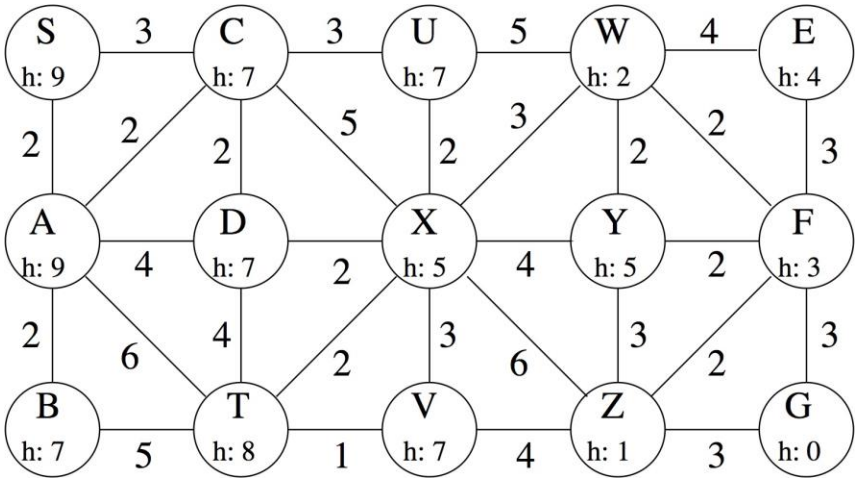
**This page was last updated: 03/17/2018 09:56:14**

---

**Activitity 4.1** Path Search Algorithms on a Graph

---

Consider the task of finding a path from start state S to goal state G, given the distances and heuristic values in this diagram:




For each of the following strategies, list the order in which the states are expanded. Whenever there is a choice of states, you should select the one that comes first in alphabetical order. In each case, you should skip any states that have previously been expanded, and you should continue the search until the goal node is expanded.

a. Breadth First Search

   S, A, C, B, D, T, U, X, V, W, Y, Z, E, F, G

b. Depth First Search

   S, A, B, T, D, C, U, W, E, F, G

c. Uniform Cost Search [Hint: first compute g() for each state in the graph]

   S(0), A(2), C(3), B(4), D(5), U(6), X(7), T(8), V(9), W(10), Y(11), F(12), Z(13), E(14), G(15)

d. Greedy Search, using the heuristic shown

   S, C, X, Z, G

e. A*Search, using the heuristic shown

   S(9), C(10), A(11), B(11), D(12), X(12), W(12), U(13), Z(14), F(15), G(15)

   Note that g(X) becomes 8 after C is expanded, but drops to 7 after D is expanded. Similarly, g(G) becomes 16 when Z is expanded, but drops to 15 after F is expanded.

---

**Activitity 4.2** A*Search for the 8-Puzzle

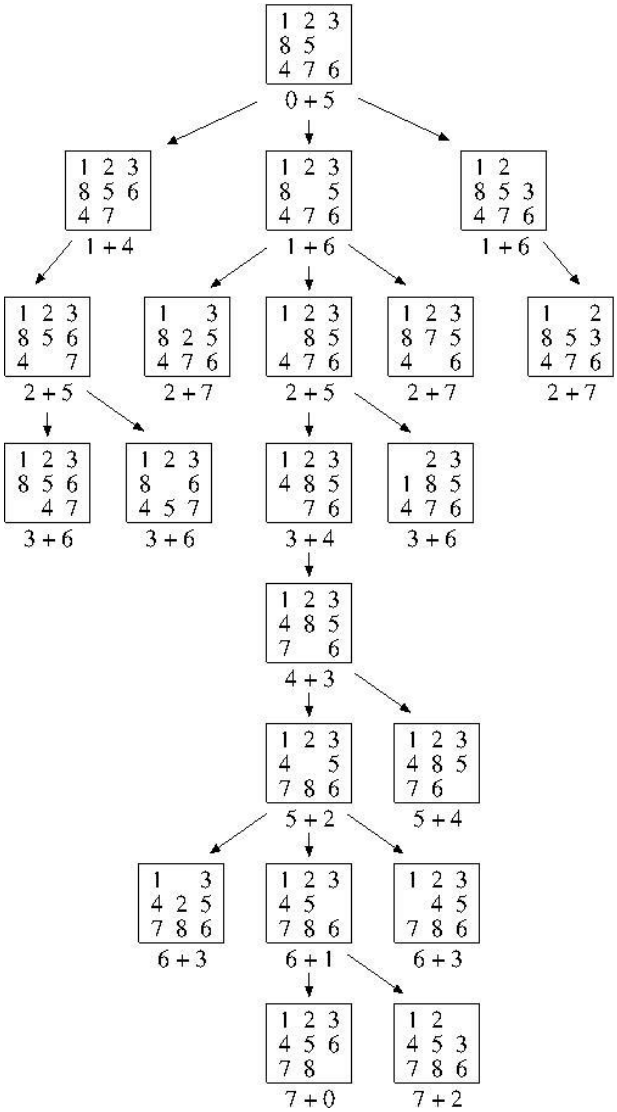
Consider the following arrangement of tiles in the 8-puzzle:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 8 | 5 | |
| 4 | 7 | 6 |

Keeping in mind that the goal state is:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Trace the A* Search algorithm using the Total Manhattan Distance heuristic, to find the shortest path from the initial state shown above, to the goal state.



Note: The algorithm begins by exploring the left branch which, according to the heuristic, seems the most promising. After a few steps the heuristic for the left branch starts to exceed that of middle branch, so exploration shifts to the middle branch, where the optimal solution is ultimately found.

---

**Activitity 3.3** Relationships Between Search Strategies

---

Prove each of the following statements, or give a counterexample:

a. Breadth First Search is a special case of Uniform Cost Search

   Uniform Cost Search reduces to Breadth First Search when all edges have the same cost.

b. Breadth First Search, Depth First Search and Uniform Cost Search are special cases of best-first search.

   best-first search reduces to Breadth-First Search when *f(n)*=number of edges from start node to *n*, to UCS when *f(n)=g(n)*;
   it can be reduced to DFS by, for example, setting *f(n)*=-(number of nodes from start state to *n*) (thus forcing deep nodes on the current branch to be searched before shallow nodes on other branches).
   Another way to produce DFS is to set *f(n)=-g(n)* (but this might produce a particularly bad choice of nodes within the DFS framework - for example, try tracing the order in which nodes are expanded when traveling from Urziceni to Craiova).

c. Uniform Cost Search is a special case of A*Search

   A*Search reduces to UCS when the heuristic function is zero everywhere, i.e. *h(n)*=0 for all *n*;
   this heuristic is clearly admissible since it always (grossly!) underestimates the distance remaining to reach the goal.

---

**Activitity 3.4** Heuristic Path Algorithm

---

The **heuristic path algorithm** is a best-first search in which the objective function is
*f(n) = (2-w)g(n) + wh(n)*
What kind of search does this perform when *w*=0? when *w*=1? when *w*=2?

This algorithm reduces to Uniform Cost Search when *w*=0, to A*Search when *w*=1 and to Greedy Search when *w*=2.

For what values of $w$ is this algorithm complete? For what values of $w$ is it optimal, assuming $h()$ is admissible?
It is guaranteed to be optimal when $0 \leq w \leq 1$, because it is equivalent to A*Search using the heuristic

$h'(n) = [w/(2-w)]h(n) \leq h(n)$

When $w > 1$ it is not guaranteed to be optimal (however, it might work very well in practice, for some problems).

---

**Activitity 4.5** Understanding Informed Search Algorithms with Mazes

---

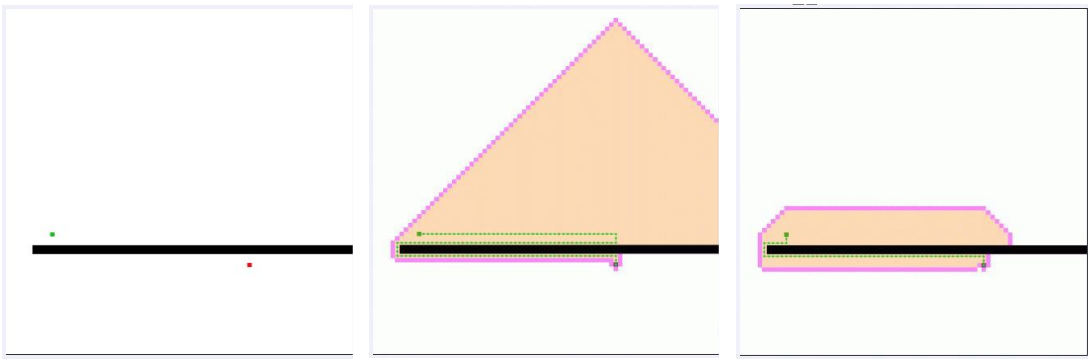
Discuss your findings and insights from the 'Fun with Mazes' activity. Compare your findings and discuss any discrepancies.

a. Use the widget to create a maze for which Bidirectional Search would find a solution faster than Breadth First Search.

> To see the real benefits of bidirectional search you would need a larger branching factor than is possible in a 2-dimensional maze. However, in a fairly open maze with the initial and goal state equally spaced at one-third and two-thirds of the width, bidirectional search should theoretically take about half the time of BFS.

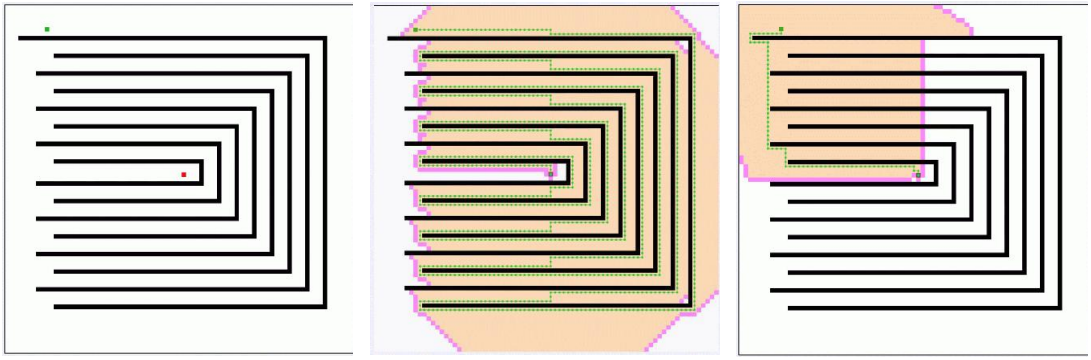
maze Breadth First Search Bidirectional Search

b. an environment for which Greedy Search takes much longer than A*Search.



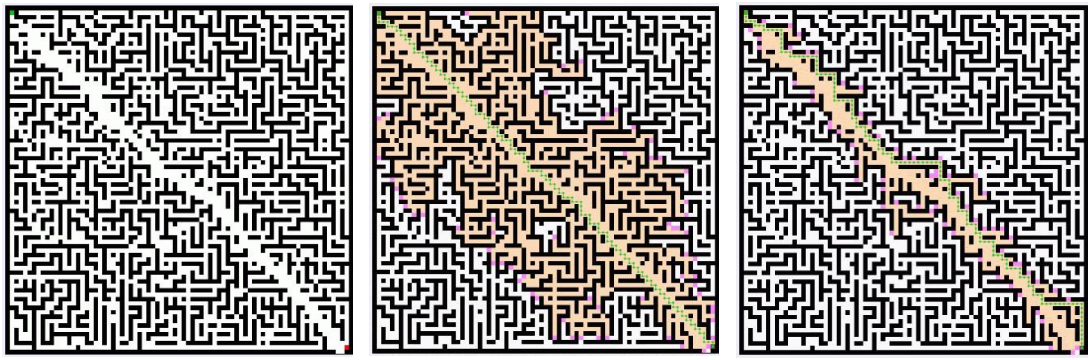
Maze Greedy Search A*Search

c. an environment for which Greedy Search produces a path that is much longer than the optimal path.



Maze Greedy Search (1127) A*Search (77)

d. an environment for which A*Search with the Euclidean Distance heuristic takes much longer than with the Manhattan Distance heuristic.



Maze A*Search (Euclidean) A*Search (Manhattan)

e. an environment that is interesting for some other reason.

# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

## Week 5 Tutorial Solutions

**This page was last updated: 04/09/2018 11:37:55**

---

**Activitity 5.1** Tic-Tac-Toe (Exercise 5.9 from R & N)

---

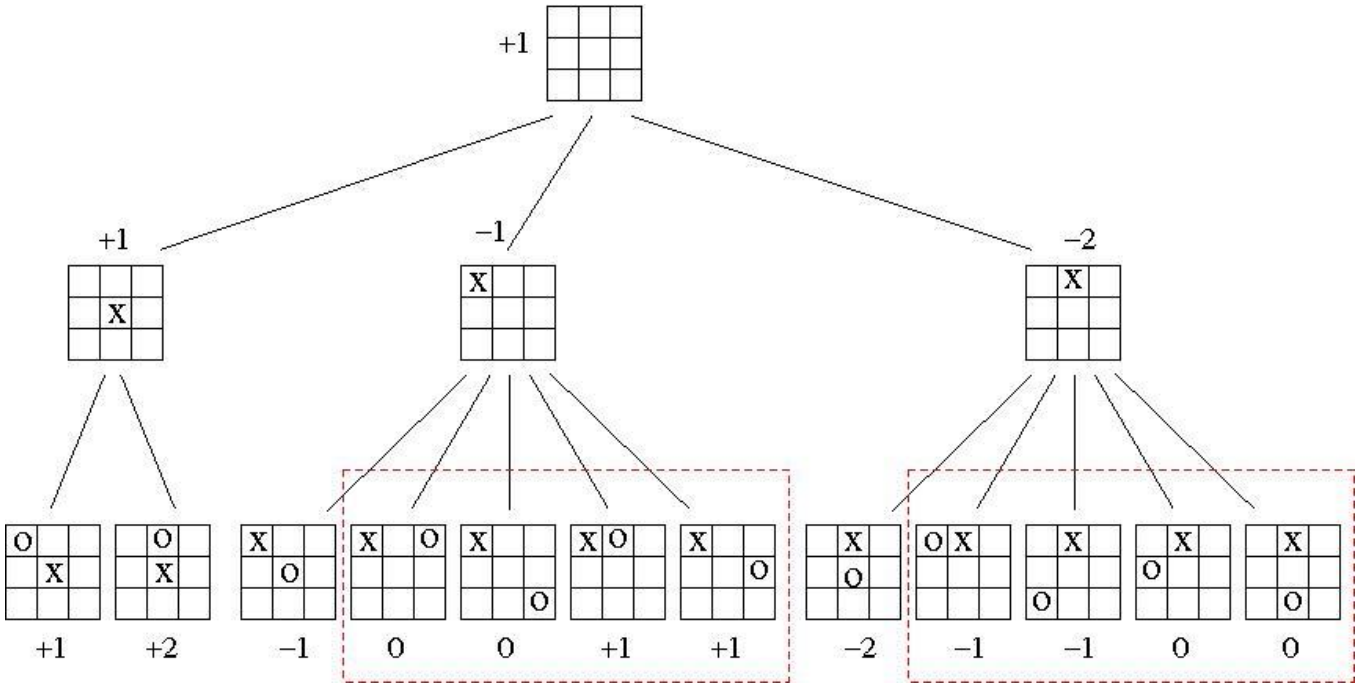
This problem exercises the basic concepts of game playing, using tic-tac-toe (naughts and crosses) as an example. We define $X_n$ as the number of rows, columns or diagonals with exactly $n$X's and no O's. Similarly, $O_n$ is the number of rows, columns or diagonals with just $n$ O's. The utility function assigns +10 to any position with $X_3>=1$ and -10 to any position with $O_3>=1$. All other terminal positions have utility 0. For the nonterminal positions, we use a linear evaluation function defined as

$$\text{Eval}(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$$

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

   There are 9 choices for the 1st move, 8 for the 2nd move, 7 for the 3rd move, etc., giving us an upper bound of 9! = 9*8*7*6*5*4*3*2*1 = 362880. But this is an overestimate, because some games end in 5, 6, 7 or 8 moves. The true figure is actually 255168.
   If we take symmetry into account, the number reduces substancially. For example, there are now only 3 choices for the first move and at most 5 choices for the second move. In fact, the total is reduced to 26830 distinct games, of which 172 end in 5 moves, 579 end in 6 moves, 5115 end in 7 moves, 7426 end in 8 moves, 8670 result in a win in 9 moves and 4868 result in a draw. There are a number of Web sites providing a full analysis. See for example http://www.se16.info/hgb/tictactoe.htm

2. 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.
3. Mark on your tree the evaluations of all the positions at depth 2.
4. 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.
5. Circle the nodes at depth 2 that would *not* be evaluated if alpha-beta pruning were applied, assuming the nodes are generated in the optimal order for alpha-beta pruning.




**Activity 5.2** Exploiting a Suboptimal Opponent

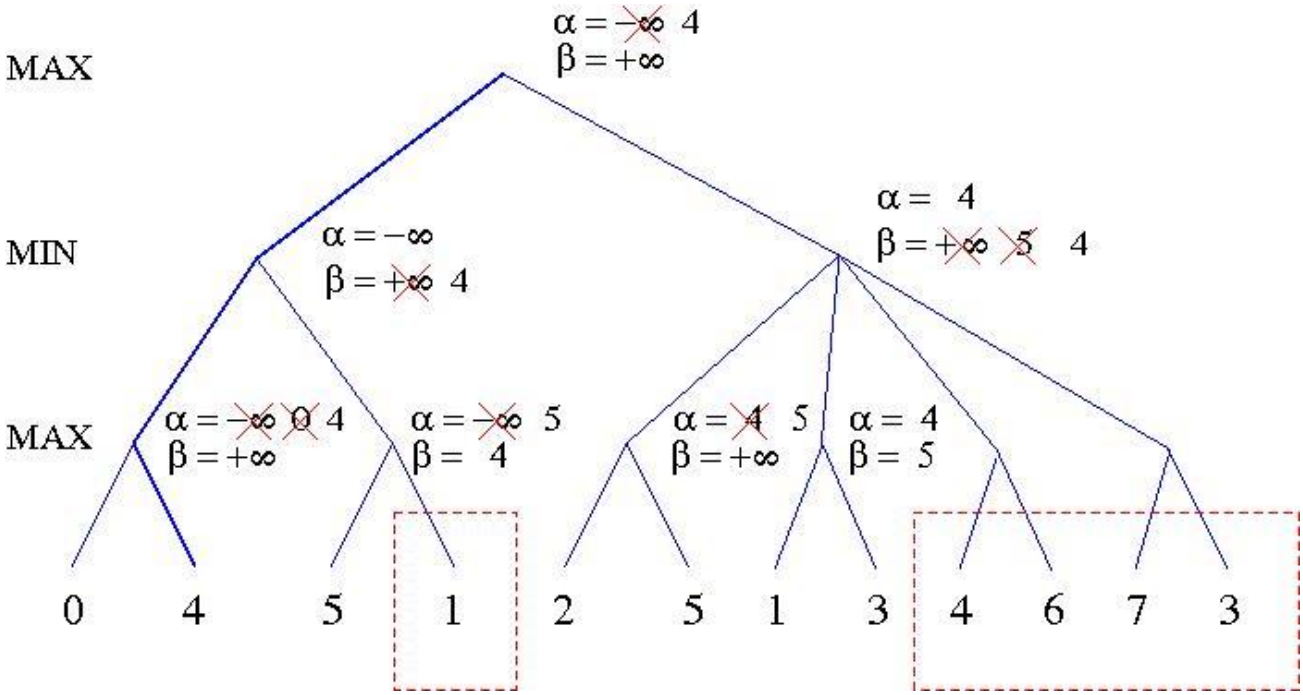
Continuing the tic-tac-toe example, if minimax (or alpha-beta search) were to search the entire game tree, it would evaluate all opening moves equally - because it assumes the opponent will play optimally, which leads to a draw in the end, no matter what the opening move. However, if we assume the opponent could make an error, we see that one particular opening move is much better than the others. Explain why.

Hint: for each leaf position from Activity 5.1, try to determine whether X or O can force a win from that position.

The best opening move for X is in a corner (i.e. the second branch in the game tree above). Check each of the five children of this node and you will see that, for all but one of them, X can use a "fork" tactic to force a win. Even if O chooses the correct move (which is in the centre), X can play into the opposite corner giving him yet another chance to win (if O is silly enough to play in a remaining corner).

**Activity 5.3** Applying Alpha-Beta Search

Apply the alpha-beta search algorithm to the following game tree, indicating clearly the values of alpha and beta at each node as the algorithm progresses, and circling the nodes that would not be evaluated.




Note:

1. It is helpful to first do a full minimax search, to tell us the "line of best play" (which is shown in bold).
2. The algorithm prunes off the last four nodes in one stroke, implicitly using logic which might be paraphrased as follows:

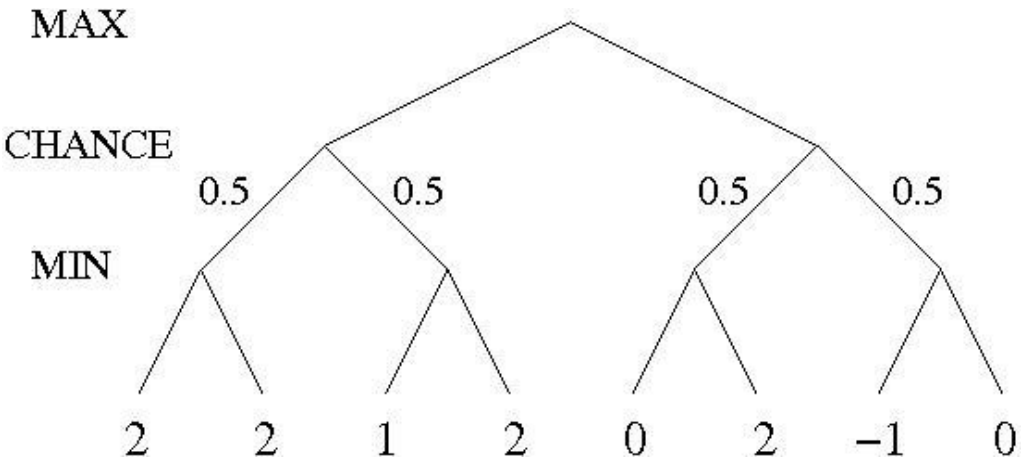
   > "By choosing the left option, MAX can guarantee a reward of at least 4. The right option would allow MIN to reduce the reward below 4, so MAX can reject this option without further evaluation."

---

**Activitity 5.4** Pruning in Games with Chance Nodes

---

This question considers pruning in games with chance nodes. This figure shows the complete game tree for a very simple such game. Assume that the leaf nodes are to be evaluated in left-to-right order, and that before a leaf node is evaluated, we know nothing about its value -- the range of possible values is -∞ to ∞.




1. Copy the figure, mark the value of all internal nodes, and indicate the best move at the root with an arrow.
2. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaf? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.

   The value of the left branch is 1.5 . After the first six leaves, we still need to keep evaluating, because if the 7th and 8th leaf were both greater than 3 then the right branch would be preferred. Once the 7th leaf is evaluated, we know that the value of the right branch is at most -0.5, so we do not need to evaluate the 8th leaf.

3. Suppose the leaf node values are known to lie between -2 and 2 inclusive. After the first two leaves are evaluated, what is the value range for the left-hand chance node?

   It is between 0 and 2 (inclusive).

4. Circle all the leaves that need to be evaluated under the assumption in Part 3.

   Only the first five leaves need to be evaluated. After that, we know that the right branch is between -2 and 1, so it is definitely inferior to the left branch.

---

**Activitity 5.5** Further Discussion

---

1. Describe an optimal strategy for a simple version of the game Nim, which uses only a single heap of *n* stones. Players take turns to remove either 1, 2 or 3 stones from the heap. The player who takes the last stone (or, all the remaining stones) wins. For what values of n can the first player force a win? For what values can the second player force a win?

If *n* is divisible by 4, the second player can force a win; otherwise, the first player can force a win. The winning strategy is to always remove a number of stones equal to *n* mod 4.

2. Discuss what you found out from the Tree Search for Simple Games activity on the Alpha-Beta Pruning page, concerning one or more of these games:
    - Hexapawn on a 3x3 board
    - Connect-4, or a simplifed version with four columns, where you only need to get three in a row in order to win
    - Sprouts, with two initial dots
    - another simple game of your choosing
3. The Chinook checkers program makes extensive use of endgame databases, which provide exact values for every position with eight or fewer pieces. How might such databases be efficiently generated, stored and accessed?

An indexing function can be found which assigns a unique number to each position, and classifies the positions into "slices", based on the number of kings and checkers, and the rank of the most advanced checker, for each colour. These different "slices" of the database can be compressed and decompressed as needed during a game. The Chinook database contains 443,748,401,247 positions, but compresses into roughly 6 gigabytes.

The database is essentially a gigantic lookup table, whose values can be generated by a technique known as "Retrograde Analysis". We assume inductively that all positions with N-1 or fewer pieces have been evaluated, before extending to positions with N pieces. Positions with a forced capture are evaluated first (because they lead directly to a position with fewer pieces). The remaining positions are initially marked as "unlabeled", and then we repeatedly loop through them, each time labeling some positions as "win" or "loss". Positions can be labeled using either a "forward" or "backward" approach. The "forward" approach looks at an unlabeled position and generates all possible successor positions. The "backward" approach looks at a labeled position and generates all possible predecessor positions. If all the successors of a state have previously been labeled as "win" (for the opponent) then that state is labeled as "loss" (for the current player). If any successor state is labeled as "loss", then the current state is labeled as "win". If the successor states are all "unlabeled", or a mix of "win" and "unlabeled", then the current state remains unlabeled. We keep looping through the remaining positions until we get through a loop with no new positions being labeled. It follows that all remaining unlabeled positions must result in a draw.

Details can be found at http://www.cs.ualberta.ca/~chinook/publications/

---

# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

# Tutorial Solutions - Week 6

**This page was last updated: 04/15/2018 11:37:58**

---

**Activity 6.1** Decision Trees

---

Consider the task of predicting whether children are likely to be hired to play members of the Von Trapp Family in a production of The Sound of Music, based on these data:

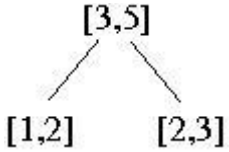
| height | hair | eyes | hired |
|---|---|---|---|
| short | blond | blue | + |
| tall | red | blue | + |
| tall | blond | blue | + |
| tall | blond | brown | – |
| short | dark | blue | – |
| tall | dark | blue | – |
| tall | dark | brown | – |
| short | blond | brown | – |

a. Compute the information (entropy) gain for each of the three attributes (height, hair, eyes) in terms of classifying objects as belonging to the class, + or – .

There are 3 objects in class '+' and 5 in '-', so the entropy is:

Entropy(parent) = $\Sigma_i$ $P_i$ $\log_2 P_i$ = -(3/8)log(3/8) - (5/8)log(5/8) = 0.954

Suppose we split on height:




Of the 3 'short' items, 1 is '+' and 2 are '-', so Entropy(short) = -(1/3)log(1/3) - (2/3)log(2/3) = 0.918

Of the 5 'tall' items, 2 are '+' and 3 are '-', so Entropy(tall) = -(2/5)log(2/5) - (3/5)log(3/5) = 0.971

The average entropy after splitting on 'height' is Entropy(height) = (3/8)(0.918) + (5/8)(0.971) = 0.951

The information gained by testing this attribute is: 0.954 - 0.951 = 0.003 (i.e. very little)

If we try splitting on 'hair' we find that the branch for 'dark' has 3 items, all '-' and the branch for 'red' has 1 item, in '+'. Thus, these branches require no further information to make a decision. The branch for 'blond' has 2 '+' and 2 '-' items and so requires 1 bit. That is,

Entropy(hair) = (3/8)(0) + (1/8)(0) + (4/8)(1) = 0.5

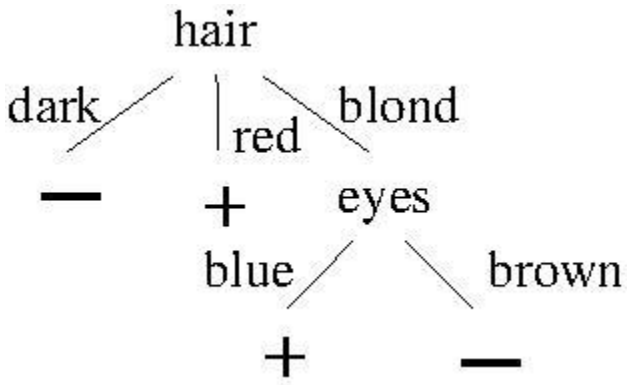
and the information gained by testing hair is 0.954 - 0.5 = 0.454 bits.

By a similar calculation, the entropy for testing 'eyes' is (5/8)(0.971) + (3/8)(0) = 0.607, so the information gained is 0.954 - 0.607 = 0.347 bits.

Thus 'hair' gives us the maximum information gain.

b. Construct a decision tree based on the minimum entropy principle.

Since the 'blond' branch for hair still contains a mixed population, we need to apply the procedure recursively to these four items. Note that we now only need to test 'height' and 'eyes' since the 'hair' attribute has already been used. If we split on 'height', the branch for 'tall' and 'short' will each contain one '+' and one '-', so the entropy gain is zero. If we split on 'eyes', the 'blue' brach contains two '+'s and the 'brown' branch two '-'s, so the tree is complete:

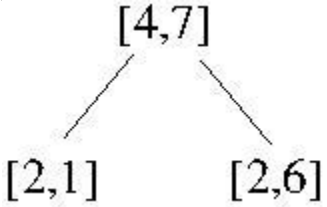


**Activity 6.2** Laplace Pruning

The Laplace error estimate for pruning a node in a Decision Tree is given by:

$$E = 1 - \frac{n + 1}{N + k}$$

where $N$ is the total number of items, $n$ is the number of items in the majority class and $k$ is the number of classes. Given the following subtree, should the children be pruned or not? Show your calculations.




Error(Parent) = 1 - (7+1)/(11+2) = 1 - 8/13 = 5/13 = 0.385

Error(Left) = 1 - (2+1)/(3+2) = 1 - 3/5 = 2/5 = 0.4

Error(Right) = 1 - (6+1)/(8+2) = 1 - 7/10 = 3/10 = 0.3

Backed Up Error = (3/11)*(0.4) + (8/11)*(0.3) = 0.327 < 0.385

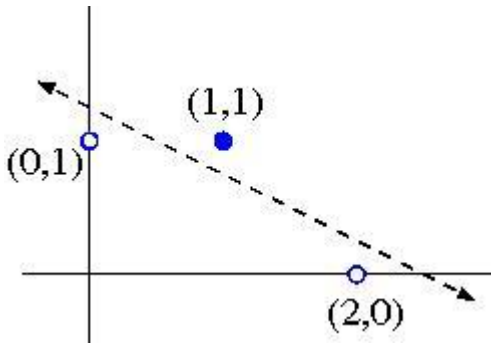
Since Error of Parent is larger than Backed Up Error ⇒ Don't Prune

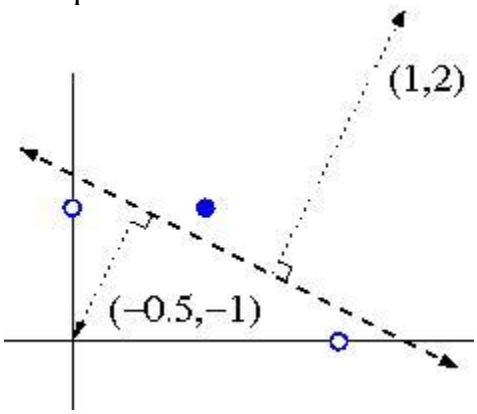
**Activity 6.3** Perceptron Learning

1. Construct by hand a Perceptron which correctly classifies the following data; use your knowledge of plane geometry to choose appropriate values for the weights $w_0$, $w_1$ and $w_2$.

| Training Example | $x_1$ | $x_2$ | Class |
|---|---|---|---|
| a. | 0 | 1 | -1 |
| b. | 2 | 0 | -1 |
| c. | 1 | 1 | +1 |

2. The first step is to plot the data on a 2-D graph, and draw a line which separates the positive from the negative data points:

3. 

4. This line has slope -1/2 and $x_2$-intersect 5/4, so its equation is:
5. $x_2 = 5/4 - x_1/2$,
   i.e. $2x_1 + 4x_2 - 5 = 0$.
6. Taking account of which side is positive, this corresponds to these weights:
7. $w_0 = -5$
   $w_1 = 2$
   $w_2 = 4$
8. Alternatively, we can derive weights $w_1=1$ and $w_2=2$ by drawing a vector normal to the separating line, in the direction pointing towards the positive data points:
9. 

10. The bias weight $w_0$ can then be found by computing the dot product of the normal vector with a perpendicular vector from the separating line to the origin. In this case $w_0 = 1(-0.5) + 2(-1) = -2.5$
11. (Note: these weights differ from the previous ones by a normalizing constant, which is fine for a Perceptron)
12. Demonstrate the Perceptron Learning Algorithm on the above data, using a learning rate of 1.0 and initial weight values of

    $w_0 = -0.5$
    $w_1 = 0$
    $w_2 = 1$

    In your answer, you should clearly indicate the new weight values at the end of each training step.

| Iteration | $w_0$ | $w_1$ | $w_2$ | Training Example | $x_1$ | $x_2$ | Class | $s=w_0+w_1x_1+w_2x_2$ | Action |
|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | -0.5 | 0 | 1 | a. | 0 | 1 | - | +0.5 | Subtract |
| 2 | -1.5 | 0 | 0 | b. | 2 | 0 | - | -1.5 | None |
| 3 | -1.5 | 0 | 0 | c. | 1 | 1 | + | -1.5 | Add |
| 4 | -0.5 | 1 | 1 | a. | 0 | 1 | - | +0.5 | Subtract |
| 5 | -1.5 | 1 | 0 | b. | 2 | 0 | - | +0.5 | Subtract |
| 6 | -2.5 | -1 | 0 | c. | 1 | 1 | + | -3.5 | Add |
| 7 | -1.5 | 0 | 1 | a. | 0 | 1 | - | -0.5 | None |
| 8 | -1.5 | 0 | 1 | b. | 2 | 0 | - | -1.5 | None |
| 9 | -1.5 | 0 | 1 | c. | 1 | 1 | + | -0.5 | Add |
| 10 | -0.5 | 1 | 2 | a. | 0 | 1 | - | +1.5 | Subtract |
| 11 | -1.5 | 1 | 1 | b. | 2 | 0 | - | +0.5 | Subtract |
| 12 | -2.5 | -1 | 1 | c. | 1 | 1 | + | -2.5 | Add |
| 13 | -1.5 | 0 | 2 | a. | 0 | 1 | - | +0.5 | Subtract |
| 14 | -2.5 | 0 | 1 | b. | 2 | 0 | - | -2.5 | None |
| 15 | -2.5 | 0 | 1 | c. | 1 | 1 | + | -1.5 | Add |
| 16 | -1.5 | 1 | 2 | a. | 0 | 1 | - | +0.5 | Subtract |
| 17 | -2.5 | 1 | 1 | b. | 2 | 0 | - | -0.5 | None |
| 18 | -2.5 | 1 | 1 | c. | 1 | 1 | + | -0.5 | Add |
| 19 | -1.5 | 2 | 2 | a. | 0 | 1 | - | +0.5 | Subtract |
| 20 | -2.5 | 2 | 1 | b. | 2 | 0 | - | +1.5 | Subtract |
| 21 | -3.5 | 0 | 1 | c. | 1 | 1 | + | -2.5 | Add |
| 22 | -2.5 | 1 | 2 | a. | 0 | 1 | - | -0.5 | None |
| 23 | -2.5 | 1 | 2 | b. | 2 | 0 | - | -0.5 | None |
| 24 | -2.5 | 1 | 2 | c. | 1 | 1 | + | +0.5 | None |

---

**Activity 6.5** Evolution (COMP3411/9814 only)

---

Consider the following four algorithms applied to the task of searching through the space of bitstrings of length 20 to find the bitstring which maximizes a given fitness function:

a. completely random search
b. hillclimbing
c. genetic algorithm (GA) using mutation only (no crossover)
d. genetic algorithm using one-point crossover as well as mutation

For concreteness, let's assume that the GA has a population of 200, with 100 individuals replaced at each generation.

For each of the fitness functions listed below, which of the four algorithms would find the solution fastest? which would be slowest? or would two of them take about the same amount of time?

1. *f(x)* = the number of 1's in *x*.
2. *f(x)* = 100, if *x* = 00000000000000000000,
   0, otherwise.
3. *f(x)* = 100, if *x* = 00000000000000000000,
   the number of 1's in *x*, otherwise.
4. *f(x)* = 14, if *x* contains 7 consecutive 1's **and** 7 consecutive 0's,
   7, if *x* contains 7 consecutive 1's **or** 7 consecutive 0's,
   0, otherwise.

1. This fitness landscape presents a nice "hill" to climb. Hillclimbing would find the solution much faster than random search, and so would the genetic algorithm (with or without crossover)
2. This is called a "needle in a haystack" problem. The fitness landscape is completely flat, until you randomly stumble upon the solution. All four algorithms would take about the same amount of time.
3. This is called a "deceptive" landscape. The slope tends to draw you **away** from the global maximum, and towards a local maximum. In this case, random search would find the solution faster than either hillclimbing or the GA.
4. In this case, the GA with crossover would be fastest, GA without crossover would be somewhat slower, hillclimbing would be slower still and random search would be slowest. The strings of 7 consecutive 1's and 7 consecutive 0's behave as "building blocks", which independently increase their prevalence in the population and are then combined, through crossover, into a single individual.

---

# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

## Tutorial Solutions - Week 7

**This page was last updated: 04/29/2018 10:40:25**

---

**Activity 10.2** Multi-Layer Neural Networks to Compute Logical Functions

---

Explain how each of the following could be constructed:

1. Perceptron to compute the OR function of *m* inputs

   Set the bias weight to -½, all other weights to 1.

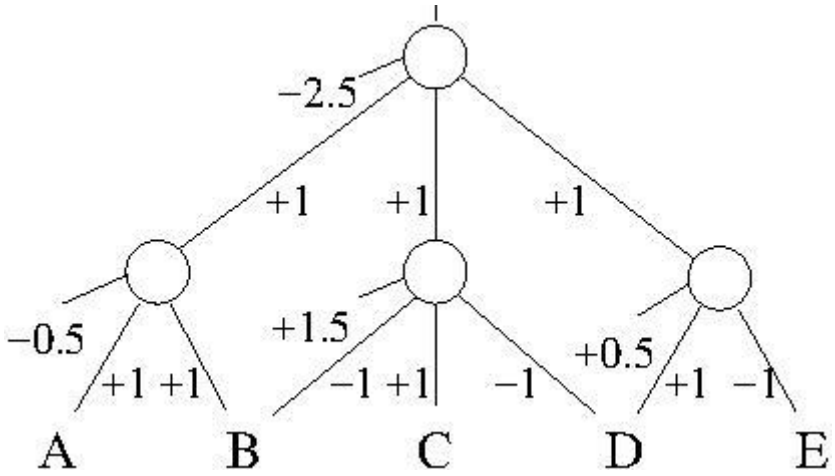
2. Perceptron to compute the AND function of *n* inputs

   Set the bias weight to (½ - *n*), all other weights to 1.

3. 2-Layer Neural Network to compute any (given) logical expression, assuming it is written in Conjunctive Normal Form.

   Each hidden node should compute one disjunctive term in the expression. The weights should be -1 for items that are negated, +1 for the others. The bias should be (*k* - ½) where *k* is the number of items that are negated. The output node then computes the conjunction of all the hidden nodes, as in part 2.

   For example, here is a network that computes (A ∨ B) ∧ (¬ B ∨ C ∨ ¬ D) ∧ (D ∨ ¬ E)




---

**Activity 10.3** Computing XOR with One Hidden Unit

---

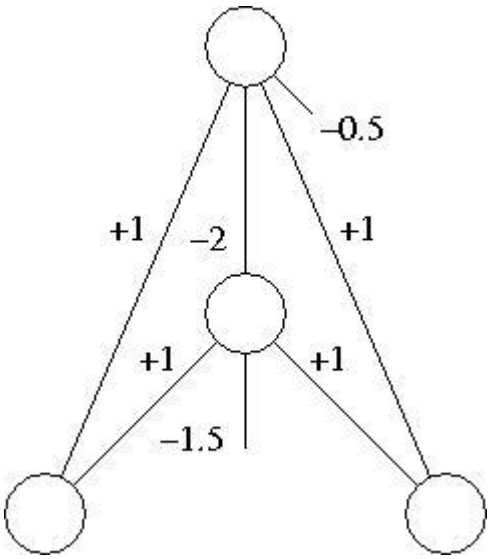
We have seen how to compute XOR using a Neural Network with two inputs, two hidden units and one output.

Can you construct a Neural Network to compute XOR which has only one hidden unit, but also includes shortcut connections from the two inputs directly to the (one) output?

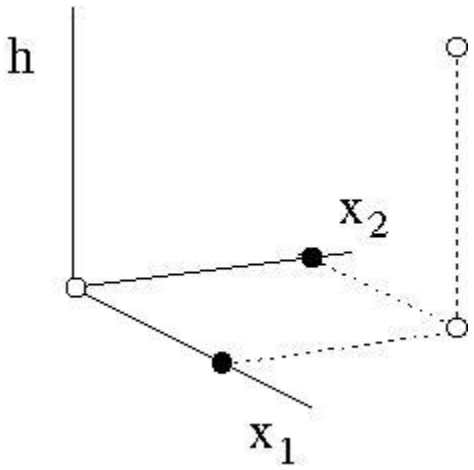
Hint: start with a network that computes the inclusive OR, and then try to think of how it could be modified.

Inclusive and exclusive OR are similar, but differ in the case where both inputs are True, i.e. where ($x_1$ AND $x_2$) is True.

We therefore introduce a single hidden unit which computes ($x_1$ AND $x_2$). This hidden unit is connected to the output with a negative weight, thus forcing this input to be classified negatively.




The addition of this hidden "feature" creates a 3-dimensional space in which the points can be linearly separated by a plane. The weights for the output unit (+1,+1,-2) specify a vector perpendicular to the separating plane, and its distance from the origin is determined by the output bias divided by the length of this vector.




---

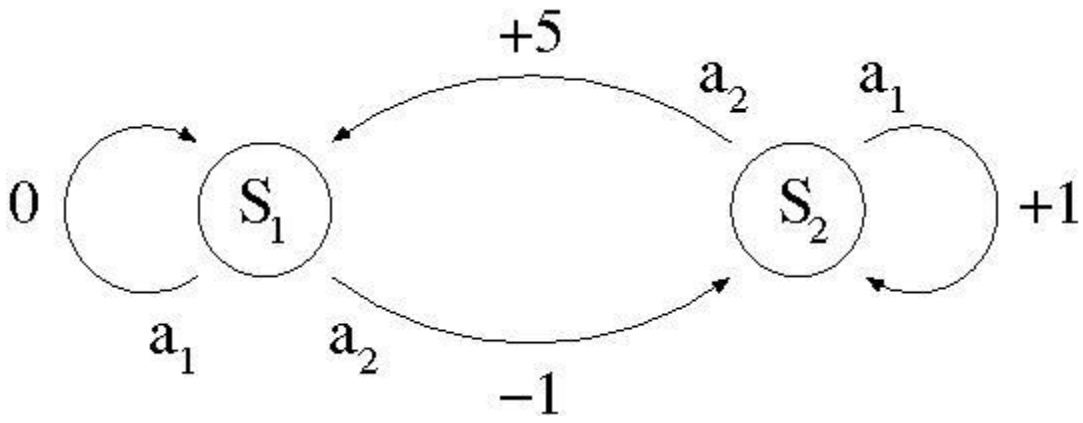
**Activity 7.4** Q-Learning [COMP3411/9814 only]

---

Consider a world with two states S = {$S_1$, $S_2$} and two actions A = {$a_1$, $a_2$}, where the transitions δ and reward *r* for each state and action are as follows:

$\delta(S_1, a_1) = S_1$, $r(S_1, a_1) = 0$
$\delta(S_1, a_2) = S_2$, $r(S_1, a_2) = -1$
$\delta(S_2, a_1) = S_2$, $r(S_2, a_1) = +1$
$\delta(S_2, a_2) = S_1$, $r(S_2, a_2) = +5$

1. Draw a picture of this world, using circles for the states and arrows for the transitions.




2. Assuming a discount factor of $\gamma = 0.9$, determine:
   a. the optimal policy $\pi^* : S \rightarrow A$

      $\pi^*(S_1) = a_2$
      $\pi^*(S_2) = a_2$

   b. the value function $V^* : S \rightarrow R$

      $V(S_1) = -1 + \gamma V(S_2)$
      $V(S_2) = 5 + \gamma V(S_1)$

      So $V(S_1) = -1 + 5\gamma + \gamma^2 V(S_1)$
      i.e. $V(S_1) = (-1 + 5\gamma) / (1 - \gamma^2) = 3.5 / 0.19 = 18.42$

      $V(S_2) = 5 + \gamma V(S_1) = 5 + 0.9 * 3.5 / 0.19 = 21.58$

   c. the "Q" function $Q : S \times A \rightarrow R$

      $Q(S_1, a_1) = \gamma V(S_1) = 16.58$
      $Q(S_1, a_2) = V(S_1) = 18.42$
      $Q(S_2, a_1) = 1 + \gamma V(S_2) = 20.42$
      $Q(S_2, a_2) = V(S_2) = 21.58$

3. Write the Q values in a matrix:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | 16.58 | 18.42 |
| $S_2$ | 20.42 | 21.58 |

4. Trace through the first few steps of the Q-learning algorithm, with all Q values initially set to zero. Explain why it is necessary to force exploration through probabilistic choice of actions, in order to ensure convergence to the true Q values.

| current state | chosen action | new Q value |
|---|---|---|
| $S_1$ | $a_1$ | $0 + \gamma*0 = 0$ |
| $S_1$ | $a_2$ | $-1 + \gamma*0 = -1$ |
| $S_2$ | $a_1$ | $1 + \gamma*0 = +1$ |

5. At this point, the table looks like this:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | 0 | -1 |
| $S_2$ | 1 | 0 |

6. If we do not force exploration, the agent will always prefer action $a_1$ in state $S_2$, so it will never explore action $a_2$. This means that $Q(S_2, a_2)$ will remain zero forever, instead of converging to the true value of 21.58 . If we force exploration, the next few steps might look like this:

| current state | chosen action | new Q value |
|---|---|---|
| $S_2$ | $a_2$ | $5 + \gamma*0 = 5$ |
| $S_1$ | $a_1$ | $0 + \gamma*0 = 0$ |
| $S_1$ | $a_2$ | $-1 + \gamma*5 = 3.5$ |
| $S_2$ | $a_1$ | $1 + \gamma*5 = 5.5$ |
| $S_2$ | $a_2$ | $5 + \gamma*3.5 = 8.15$ |

7. Now we have this table:

| Q | $a_1$ | $a_2$ |
|---|---|---|
| $S_1$ | 0 | 3.5 |
| $S_2$ | 5.5 | 8.15 |

8. From this point on, the agent will prefer action $a_2$ both in state $S_1$ and in state $S_2$. Further steps refine the Q value estimates, and, in the limit, they will converge to their true values.

| current state | chosen action | new Q value |
|---|---|---|
| $S_1$ | $a_1$ | 0 + γ*3.5 = 3.15 |
| $S_1$ | $a_2$ | -1 + γ*8.15 = 6.335 |
| $S_2$ | $a_1$ | 1 + γ*8.15 = 8.335 |
| $S_2$ | $a_2$ | 5 + γ*6.34 = 10.70 |

9. etc...

---

# COMP3411/9414/9814 Artificial Intelligence
# Session 1, 2018

# Tutorial Solutions - Week 8

**This page was last updated: 05/03/2018 16:29:44**

---

**Activity 8.1** Cryptarithmetic

---

Solve the famous Cryptarithmetic puzzle

```
  S E N D +
  M O R E
-----------
M O N E Y
```

What heuristics and strategies did you use along the way?

The sum of two 4-digit numbers cannot exceed 1998, so M=1.
10+O = S+1 or S+1+1, i.e. S = O+9 or O+8, but 1 has already been used, so O=0.
Therefore S=9, because there is no possibility of a carry from E+O.
We then have E+1 = N and 10+E = N+R or N+R+1, so R = 8 or 9, but 9 has already been assigned, so R=8.
(Note how Minimum Remaining Values has been used at each step.)
The puzzle now looks like this:

```
  9 E N D +
  1 0 8 E
-----------
1 0 N E Y
```

This gives us the two constraints:
E+1 = N
D+E = 10+Y

The remaining values are 2,3,4,5,6,7.

We have D+E <= 6+7 = 13, so Y = 2 or 3. (Note: MRV again).
But if Y=3 (Most Constraining Value) then all three variables D,E,N would need to take values 6 or 7, which is impossible (Constraint Propagation).
Therefore Y=2, E=5, N=6 and D=7.

---

**Activity 8.2** Forward Checking and Arc Consistency

---

Use Forward Checking to show that the Australia map-colouring problem has no solution when we assign WA=green, V=Red, NT=Red. If we apply Arc Consistency as well, can the inevitable failure be detected further up the tree?

| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| initial domains | R G B | R G B | R G B | R G B | R G B | R G B | R G B |
| WA=Green | G | R B | R G B | R G B | R G B | R B | R G B |
| V = Red | G | R B | R G B | G B | R | B | R G B |
| NT = Red | G | R | G B | G B | R | B | R G B |
| SA = Blue | G | R | G | G | R | B | R G B |
| Q = Green | G | R | G | | R | B | R G B |

No options remain for NSW, so there is no solution.

If we also apply Arc Consistency, the question can be resolved further up the tree (but with extra computation at each node) as follows:

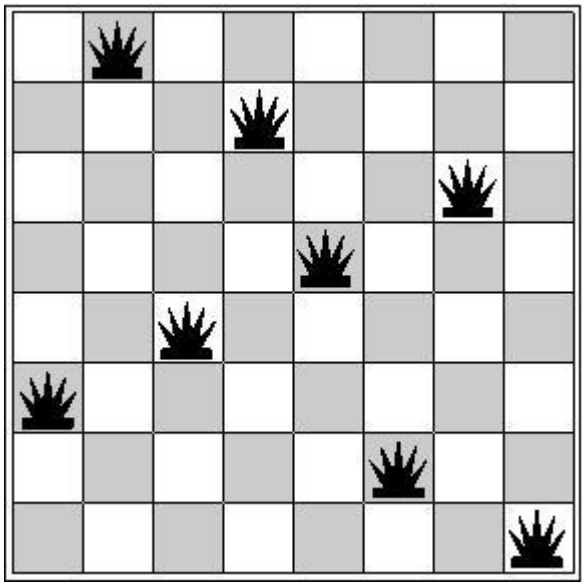
| | WA | NT | Q | NSW | V | SA | T |
|---|---|---|---|---|---|---|---|
| initial domains | R G B | R G B | R G B | R G B | R G B | R G B | R G B |
| WA=Green | G | R B | R G B | R G B | R G B | R B | R G B |
| V = Red | G | R B | R G B | G B | R | B | R G B |
| NT→SA, Q→SA, NSW→SA | G | R | R G | G | R | B | R G B |
| Q → NT | G | R | G | G | R | B | R G B |
| Q → NSW | G | R | | G | R | B | R G B |

---

**Activity 8.3** Local Search

---

Consider the following state for the 8-queens problem:



a. Is this a solution?

   No, the queens on columns 2 and 5 are attacking each other.

b. What is the value of *h*?

   There is only one violation, so $h$=1.

c. Explain why Hill-climbing with Min Conflicts would get stuck in this state, but Simulated Annealing may be able to "escape" and eventually find a solution.

   For each column, moving the queen on that column (while keeping the other queens in place) would result in an increase to *h*. Therefore, any such move will be rejected by Hill-climbing. Simulated Annealing, however, can accept such a move with probability $\exp(-(h_1-h_0)/T)$, thus bumping the system out of this local optimum and allowing it to continue the search for a global optimum.
   (Note: when started from a random initial state, Hill-climbing will get stuck 86% of the time on this problem, and will need to be continually re-started from a new random state each time, until it succeeds.)

---

**Activity 8.4** Logic Puzzle (Exercise 6.6 from R & N)

---

Consider the following logic puzzle: In five houses, each with a different color, live five persons of different nationalities, each of whom prefers a different brand of candy, a different drink, and a different pet. Given the following facts, the

questions to answer are "Where does the zebra live, and in which house do they drink water?".

- The Englishman lives in the red house.
- The Spaniard owns a dog.
- The Norwegian lives in the first house on the left.
- The Green house is immediately to the right of the ivory house.
- The man who eats Hershey bars lives in the house next to the man with the fox.
- Kit Kats are eaten in the yellow house.
- The Norwegian lives next to the blue house.
- The Smarties eater owns snails.
- The Snickers eater drinks orange juice.
- The Ukranian drinks tea.
- The Japanese eats Milky Ways.
- Kit Kats are eaten in a house next to the house where the horse is kept.
- Coffee is drunk in the green house.
- Milk is drunk in the middle house.

Discuss different representations of this problem as a CSP. Why might we prefer one representation over another?
**Bonus Challenge**: Write a Prolog program to solve this puzzle.

We can arbitrarily assign a number between 1 and 5 to each attribute value, as follows:

| Color | Nationality | Candy | Drink | Pet |
|---|---|---|---|---|
| 1. Red | 1. English | 1. Hershey bar | 1. Orange Juice | 1. Dog |
| 2. Green | 2. Spanish | 2. KitKat | 2. Tea | 2. Fox |
| 3. Ivory | 3. Norwegian | 3. Smarties | 3. Coffee | 3. Snails |
| 4. Yellow | 4. Ukranian | 4. Snickers | 4. Milk | 4. Horse |
| 5. Blue | 5. Japanese | 5. Milky Way | 5. Water | 5. Zebra |

We can then construct a 5 by 5 table where the columns are attributes, the rows are values and the interior entries indicate which house has the specified value for the specified attribute. The constraints can then be encoded as follows:

| | Color | Nation | Candy | Drink | Pet |
|---|---|---|---|---|---|
| 1 | A | A | D | G | B |
| 2 | C | B | E | H | D±1 |
| 3 | C-1 | 1 | F | C | F |
| 4 | E | H | G | 3 | E±1 |
| 5 | 2 | J | J | W | Z |

There is an additional constraint that all the numbers 1, 2, 3, 4, 5 must occur exactly once in each column.

We can deduce that E=1, and derive the following exclusions:

```
    1  2  3  4  5
  +---------------
A| x  x
B| x  x
C| x  x  x
D| x
E|    x  x  x  x
F| x  x
G| x     x
H| x     x
J| x
  +---------------
```

Since G, H, C cannot be 1, this implies that W=1, i.e. Water is drunk in the first house on the left.

If C=4, then A=5, G=5, H=2, so F, B, J must be chosen from {2, 3} and two of them must be equal (which is not allowed).
Therefore C=5, and A=3.
If H=4, G=J=2 which is not allowed.
Therefore, H=2, G=4, J=5, B=4, F=3, D=2 and Z=5, i.e. the Zebra is in the last house on the right.

An alternative representation for the problem is a table where the rows are the house numbers and the interior entries indicate the value of the specified attribute for the specified house. This representation is not so convenient for solving the problem by hand (or with an imperative language like C). However, it can readily be encoded in a Prolog program, as shown here.

Solutions for a number of similar puzzles can be found at www.ypologist.com/mmalita17/HOMEPAGE/logic

Yet another approach is to line up the houses in order, and treat the constraints as pieces of a jigsaw puzzle, as described here.

---

# COMP3411 Artificial Intelligence
# Session 1, 2018

## Tutorial Solutions - Week 8

**This page was last updated: 05/08/2018 21:48:25**

## **Activity 8.1** Three Godesses in a Temple

Three goddesses were sitting in an old Indian temple. Their names were Truth, Lie and Wisdom. Truth always told the truth, Lie always lied and Wisdom sometimes told the truth and sometimes lied. A man entered the temple. He first asked the goddess on the left: "Who is sitting next to you?" "Truth," she answered. He then asked the middle one: "Who are you?" "Wisdom." Finally he asked the one on the right: "Who is your neighbor?" "Lie," she replied. Can you say which goddess was which?

The goddess on left cannot be True because she said someone else was True. The middle one cannot be True either, so the one on the right must be True. This means the middle one is Lie and the left goddess is Wisdom.

## **Activity 8.2** Validity and Satisfiability

(Exercise 7.10 from R & N)

Decide whether each of the following sentences is valid, satisfiable, or neither. Verify your decisions using truth tables or equivalence and inference rules. For those that are satisfiable, list all the models that satisfy them.

a. Smoke ⇒ Smoke

   Valid [implication, excluded middle]

b. Smoke ⇒ Fire

   Satisfiable

| **Smoke** | **Fire** | **Smoke ⇒ Fire** |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

   Models are: {Smoke, Fire}, {Fire}, {}

c. ( Smoke ⇒ Fire ) ⇒ ( ¬ Smoke ⇒ ¬ Fire )

   Satisfiable

| Smoke | Fire | Smoke ⇒ Fire | ¬ Smoke ⇒ ¬ Fire | KB |
|---|---|---|---|---|
| T | T | T | T | T |
| T | F | F | T | T |
| F | T | T | F | F |
| F | F | T | T | T |

Models are: {Smoke, Fire}, {Smoke}, {}

d. Smoke ∨ Fire ∨ ¬ Fire

Valid

e. (( Smoke ∧ Heat ) ⇒ Fire ) ⇔ (( Smoke ⇒ Fire ) ∨ ( Heat ⇒ Fire ))

Valid

$$
\begin{aligned}
((S \wedge H) \Rightarrow F) &\Leftrightarrow (F \vee \neg(S \wedge H)) && \text{[implication]}\\
&\Leftrightarrow (F \vee \neg S \vee \neg H) && \text{[de Morgan]}\\
&\Leftrightarrow (F \vee \neg S \vee F \vee \neg H) && \text{[idempotent, commutativity]}\\
&\Leftrightarrow (S \Rightarrow F) \vee (H \Rightarrow F) && \text{[implication]}
\end{aligned}
$$

f. ( Smoke ⇒ Fire ) ⇒ (( Smoke ∧ Heat ) ⇒ Fire )

Valid

$$
\begin{aligned}
(S \Rightarrow F) &\Leftrightarrow (F \vee \neg S) && \text{[implication]}\\
&\Rightarrow (F \vee \neg S \vee \neg H) && \text{[generalization]}\\
&\Rightarrow (F \vee \neg (S \wedge H)) && \text{[de Morgan]}\\
&\Rightarrow ((S \wedge H) \Rightarrow F) && \text{[conditional]}
\end{aligned}
$$

g. Big ∨ Dumb ∨ ( Big ⇒ Dumb )

Valid

$$
\begin{aligned}
\text{Big} \vee \text{Dumb} \vee (\text{Big} \Rightarrow \text{Dumb}) &\Leftrightarrow \text{Big} \vee \text{Dumb} \vee \text{Dumb} \vee \neg \text{Big} && \text{[implication]}\\
&\Leftrightarrow \text{Big} \vee \neg \text{Big} \vee \text{Dumb} && \text{[idempotent]}\\
&\Leftrightarrow \text{TRUE} \vee \text{Dumb} && \text{[excluded middle]}\\
&\Leftrightarrow \text{TRUE}
\end{aligned}
$$

h. ( Big ∧ Dumb ) ∨ ¬ Dumb

Satisfiable

| Big | Dumb | (Big ∧ Dumb) | (Big ∧ Dumb) ∨ ¬ Dumb |
|---|---|---|---|
| T | T | T | T |
| T | F | F | T |
| F | T | F | F |
| F | F | F | T |

Models are: {Big, Dumb}, {Big}, {}

---

**Activity 8.3** Resolution and Conjunctive Normal Form

---

**(Exercise 7.2 from R & N)**

Consider the following Knowledge Base of facts:

If the unicorn in mythical, then it is immortal, but if it is not mythical, then it is mortal and a mammal. If the unicorn in either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

1. Translate the above statements into Propositional Logic.

   Myth ⇒ ¬ Mortal
   ¬Myth ⇒ ( Mortal ∧ Mammal )
   ¬Mortal ∨ Mammal ⇒ Horned
   Horned ⇒ Magic

2. Convert this Knowledge Base into Conjunctive Normal Form.

   (¬Myth ∨ ¬ Mortal) ∧ (Myth ∨ Mortal) ∧ (Myth ∨ Mammal) ∧ (Mortal ∨ Horned) ∧ (¬Mammal ∨ Horned) ∧ (¬Horned ∨ Magic)

3. Use a series of resolutions to prove that the unicorn is Horned.

   Using Proof by Contradiction, we add to the database the negative of what we are trying to prove:

   ¬Horned

   We then try to derive the "empty clause" by a series of Resolutions:

¬Horned ∧ (Mortal ∨ Horned)
Mortal

¬Horned ∧ (¬Mammal ∨ Horned)
¬Mammal

Mortal ∧ (¬Myth ∨ ¬Mortal)
¬Myth

¬Myth ∧ (Myth ∨ Mammal)
Mammal

Mammal ∧ ¬Mammal

Having derived the empty clause, the proof (of Horned) is complete.

4. Give all models that satisfy the Knowledge Base. Can you prove that the unicorn is Mythical? How about Magical?

   Because of the rule (Horned ⇒ Magic), Magic must also be True.

   We can construct a truth table for the remaining three variables:

| Myth | Mortal | Mammal | Myth ⇒ ¬ Mortal | ¬ Myth ⇒ (Mortal ∧ Mammal) | KB |
|---|---|---|---|---|---|
| T | T | T | F | T | F |
| T | T | F | F | T | F |
| T | F | T | T | T | T |
| T | F | F | T | T | T |
| F | T | T | T | T | T |
| F | T | F | T | F | F |
| F | F | T | T | F | F |
| F | F | F | T | F | F |

   There are three models which satisfy the entire Knowledge Base: {Horned, Magic, Myth, Mammal}, {Horned, Magic, Myth}, {Horned, Magic, Mortal, Mammal}.
   We cannot prove that the unicorn is Mythical, because of the third model where Mythical is False.

---

**Activity 8.4** Sentences in First Order Logic

---

Represent the following sentences in first-order logic, using a consistent vocabulary.

a. Some students studied French in 2016.

$$\exists\, x\ Student(x) \wedge Study(x,French,2016)$$

b. Only one student studied Greek in 2015.

$$\exists x\ Study(x,Greek,2015) \wedge \forall y\ (Study(y,Greek,2015) \Rightarrow y = x)$$

sometimes written as

$$\exists !x\ Study(x,Greek,2015)$$

c. The highest score in Greek is always higher than the highest score in French.

$$\forall t\ \exists x\ \forall y\ Score(x,Greek,t) > Score(y,French,t)$$

d. Every person who buys a policy is smart.

$$\forall x,p\ Person(x) \wedge Policy(p) \wedge Buy(x,p) \Rightarrow Smart(x)$$

e. No person buys an expensive policy.

$$\neg\, \exists x,p\ Person(x) \wedge Policy(p) \wedge Expensive(p) \wedge Buy(x,p)$$

f. There is a barber who shaves all men in town who do not shave themselves.

$$\exists b\ Barber(b) \wedge \forall m\ (Man(m) \wedge InTown(m) \wedge \neg Shave(m,m) \Rightarrow Shave(b,m))$$

g. Politicians can fool some of the people all of the time, and they can fool all of the people some of the time, but they can't fool all of the people all of the time.

$$\forall p\ (\ Politician(p) \Rightarrow ((\ \exists x\ \forall t\ Fool(p,x,t)) \wedge (\ \exists t\ \forall x\ Fool(p,x,t)) \wedge (\neg\ \forall x\ \forall t\ Fool(p,x,t))))$$

---

**Activity 8.5** The Interplanetary Visitor

---

On a certain planet there are 100 highly intelligent but also highly religious inhabitants who all know and see each other on a daily basis. Some number $n$ of them have blue eyes, the rest have brown eyes. The religious laws of the planet dictate that anybody who is able to prove that their own eyes must be blue has to ritually commit suicide the same evening, at midnight. Everybody knows that everybody else will obey this law without question. For this reason, it has become forbidden on the planet to discuss eye colour, and all mirrors, cameras and other such devices have long since been destroyed. One day a visitor comes from a neighboring planet whose inhabitants are known to always speak the truth. Everyone has gathered for his departure. Just before closing the door of his spaceship, he says in a voice loud enough for everyone to hear: "At least one of you has blue eyes." What will happen in the days (and nights) that follow? (Hint: consider first the case $n$=1, then $n$=2, $n$=3, etc.)

First consider the case where there is only one inhabitant with blue eyes. When he looks around and sees that everyone else has brown eyes, he will conclude that he must have blue eyes and therefore commit suicide at midnight on the day of the visitor's departure.

If two inhabitants have blue eyes, each of them will see one other person with blue eyes. Seeing the other person still alive the next day, each of them will conclude that they must have blue eyes as well, so they will both commit suicide at midnight on the second day. We can proceed inductively to conclude that, if $n$ inhabitants have blue eyes, all of them will commit suicide at midnight on the $n^{th}$ day.

This result is somewhat surprising because, in the case where more than two inhabitants have blue eyes, everyone already knew that there was at least one inhabitant with blue eyes, and everyone knew that everyone else knew it; so it may at first appear that the visitor is not providing any new information. But this is not the case. For example, if exactly three inhabitants A, B and C had blue eyes, then B knew that C knew that somebody had blue eyes, but A didn't know that B knew that C knew that somebody had blue eyes. So there is always some new information provided by the visitor.

---

# COMP3411 Artificial Intelligence
# Session 1, 2018

## Tutorial Solutions - Week 11

**This page was last updated: 05/16/2018 16:56:37**

---

**Activity 11.1** Conditional Probability

Only 4% of the population are color blind, but 7% of men are color blind. What percentage of color blind people are men?

If we assume 50% of the population are men, then the fraction of "color blind men" is 0.5 * 0.07 = 0.035
This means the fraction of "color blind women" is 0.04 - 0.035 = 0.005
Therefore, the fraction of color blind people who are men is 0.035 / 0.04 = 87.5%.

---

**Activity 11.2** Enumerating Probabilities

---

(Exercise 13.8 from Russell & Norvig)

| | **toothache** | | **¬toothache** | |
|---|---|---|---|---|
| | **catch** | **¬catch** | **catch** | **¬catch** |
| **cavity** | .108 | .012 | .072 | .008 |
| **¬cavity** | .016 | .064 | .144 | .576 |

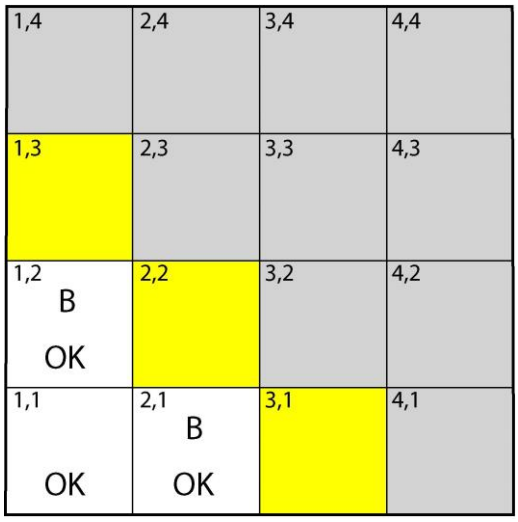
1. Given the full joint distribution shown in Figure 13.3 (also on page 17 of the Uncertainty lecture slides), calculate the following:
   a. P( toothache $\wedge$ $\neg$ catch ) = 0.012 + 0.064 = 0.076
   b. P( catch ) = 0.108 + 0.016 + 0.072 + 0.144 = 0.34
   c. P( cavity | catch ) = P( cavity $\wedge$ catch ) / P( catch )
      = ( 0.108 + 0.072 )/( 0.108 + 0.072 + 0.016 + 0.144 ) = 0.18 / 0.34 = 0.53
   d. P( cavity | toothache $\vee$ catch ) =
      P( cavity $\wedge$ (toothache $\vee$ catch)) / P( toothache $\vee$ catch )
      = ( 0.108 + 0.012 + 0.072 )/( 0.108 + 0.012 + 0.072 + 0.016 + 0.064 + 0.144 )
      = 0.192 / 0.416 = 0.46
2. Verify the conditional independence claimed in the lecture slides by showing that P( catch | toothache, cavity ) = P( catch | cavity )

   P( catch | toothache $\wedge$ cavity ) =
   P( catch $\wedge$ toothach $\wedge$ cavity ) / P( toothache $\wedge$ cavity )
   = 0.108 /( 0.108 + 0.012 ) = 0.108 / 0.12 = 0.9

   P( catch | cavity ) = P( catch $\wedge$ cavity ) / P( cavity )
   = ( 0.108 + 0.072 )/( 0.108 + 0.012 + 0.072 + 0.008 ) = 0.18 / 0.2 = 0.9

---

**Activity 11.3** Wumpus World with Three Pits

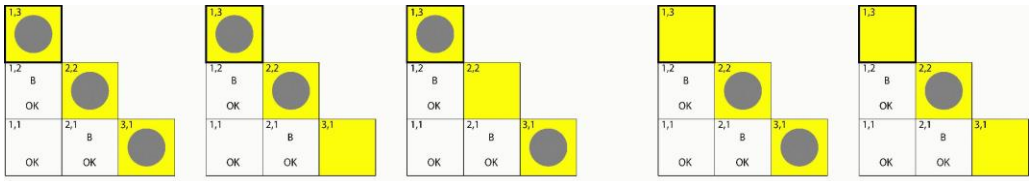



Consider the same Wumpus World situation shown above, but this time making a different prior assumption about the placement of the Pits - namely, that exactly three Pits have been placed randomly among the sixteen squares at the beginning of the game. Using this new prior, calculate:

1. the probability of a Pit in (1,3)
2. the probability of a Pit in (2,2)

We start with these equations from the Lecture Notes:
$P(Pit_{1,3} \mid known) = \Sigma_{fringe \in F} P(Pit_{1,3} \mid fringe) \Sigma_{other} P(known \mid fringe,other)P(fringe,other) / P(known)$

$P(known) = \Sigma_{fringe \in F} \Sigma_{other} P(known \mid fringe,other) P(fringe,other)$



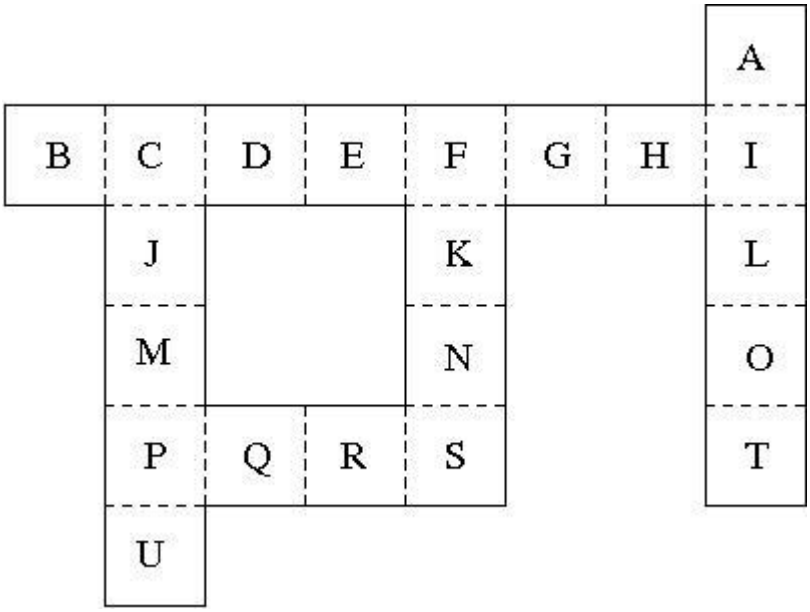
In this case, $P(known \mid fringe,other) = 1$ if and only if $fringe \in F$ and *fringe, other* between them contain exactly three Pits. For these configurations, $P(fringe,other)$ becomes a constant that can be cancelled from the numerator and denominator. If there are two pits in the fringe, the other pit can be placed in 10 different squares. If there is only one pit in the fringe, the other two pits can be placed in 45 different ways. Therefore,

$P(Pit_{1,3} \mid known) = (1+10+10)/(1+10+10+10+45) = 21/76 \approx 0.276$

$P(Pit_{2,2} \mid known) = (1+10+10+45)/(1+10+10+10+45) = 66/76 \approx 0.868$

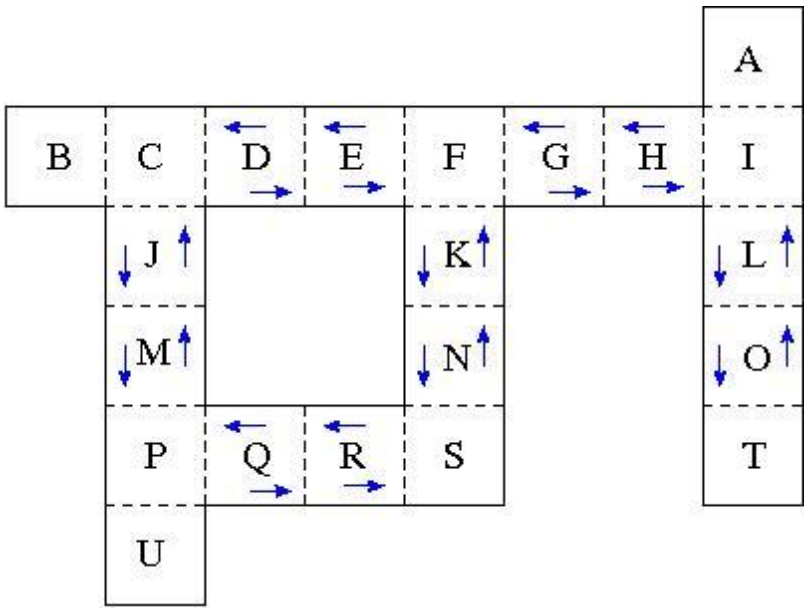
**Activity 11.4** Robot Navigation

Consider a robot which has an accurate map of its environment (shown below), but is faced with the task of determining its own location within that environment. The robot has limited sensors which only enable it to perceive up to the boundaries of the square it currently occupies (i.e. it can tell whether there is a wall or another square, in each direction).




1. The robot actually begins in square N, facing Up (but does not yet know it). Based on its limited sensors, how many possibilities are there for its initial position and orientation, consistent with what it currently perceives?

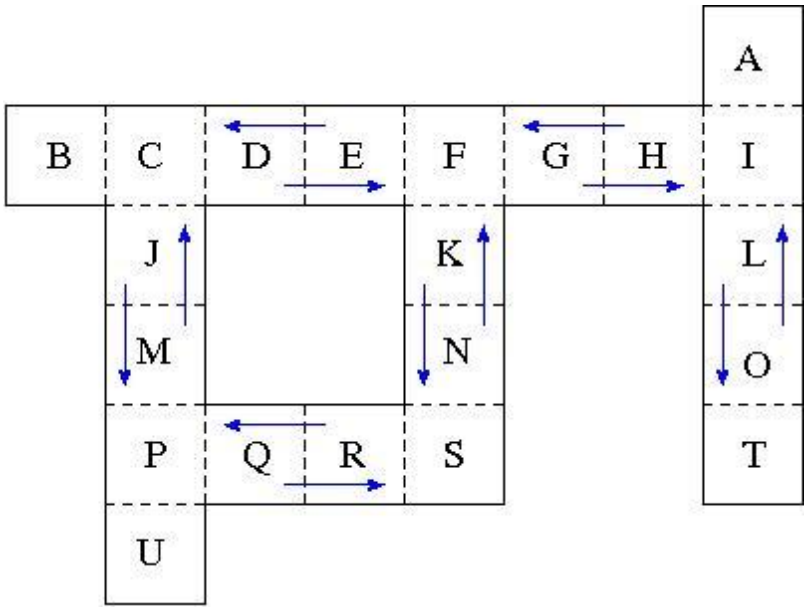
   There are 24 possibilities:



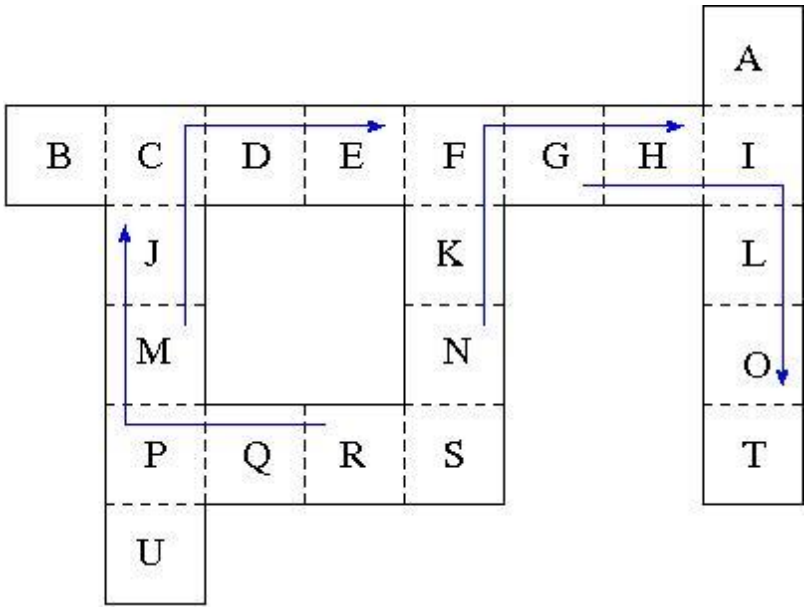
It could be in squares J, K, L, M, N, O facing Up or Down, or in squares D, E, G, H, Q, R facing Left or Right.

2. The robot then moves through the squares in this order: K, F, G, H, I, L, O, T. How many possibilities are there, at each step, for its position and orientation, consistent with its entire sequence of observations? At what point does it know its location with (effective) certainty?

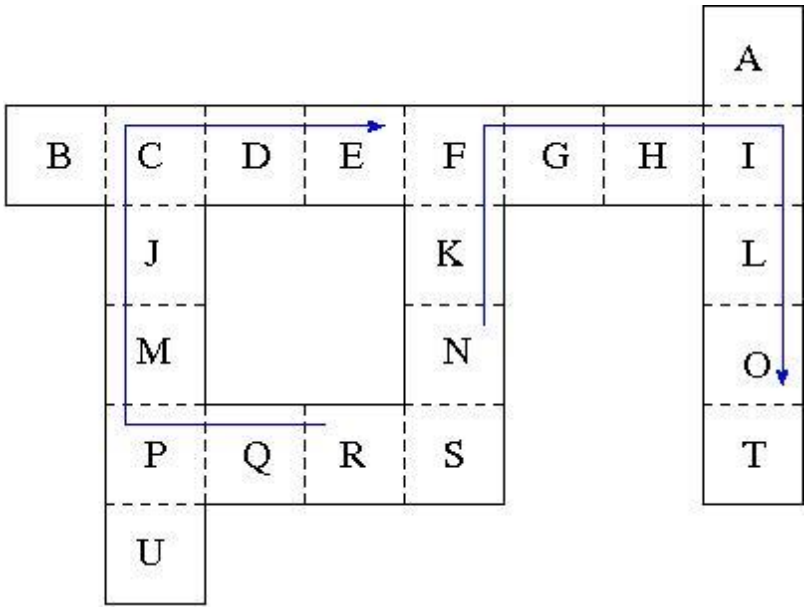
   After moving to K, only 12 possibilities remain:




   J, K, L facing Up; M, N, O facing Down; D, G, Q facing Left; E, H, R facing Right.
   After moving to F, only 4 possibilities remain, and these persist through G and H:



Upon reaching I, only 2 possibilities remain, and these persist through L and O:




Upon reaching T, the robot finally knows its position with (effective) certainty.