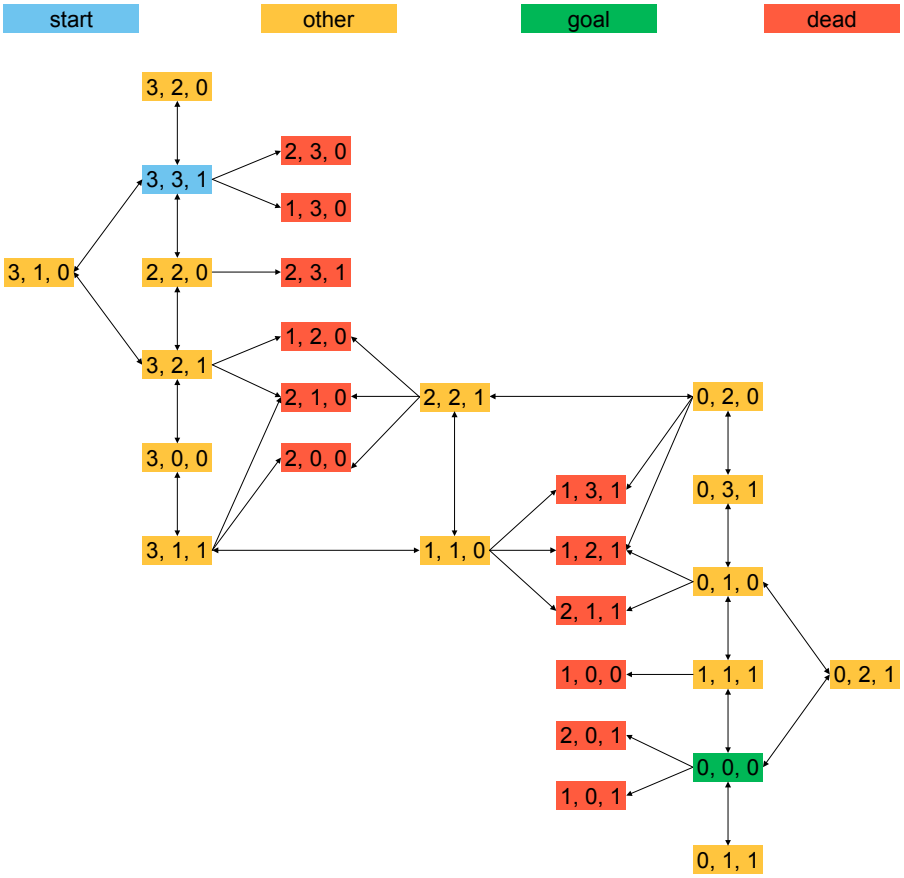
WEEK 2 TUTORIAL SOLUTIONS

PART I

Activity 1. 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).

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 characterised 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 maximising the total number of people on the far side of the river.

Activity 3. For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes, take the first one by alphabetical ordering. Make sure you understand the key properties of the different algorithms, as listed below.

State space: set of cities on Romania map
Initial state: Arad
Successor function: s(x) is the set of cities adjacent to x on the map Goal state: Bucharest
Path cost: Sum of the costs of distances between the cities on the path

(i) Depth-first:
Arad, Sibiu, Fagaras, Bucharest (note that the solution is found on the first branch only because of the rule for ordering the successors alphabetically; this is not usually the case!)

(ii) Breadth-first:
Arad, Sibiu, Timisoara, Zerind, Fagaras, Bucharest (assuming that the search stops once the goal state is generated and that when expanding a node, previously expanded nodes are checked to ensure that nodes with states already explored are not added to the frontier, e.g. Arad which is generated via the paths Arad → Sibiu → Arad and Arad → Sibiu → Oradea → Zerind → Arad)

(iii) Uniform-cost:
Arad (0), Zerind (75), Timisoara (118), Sibiu (140), Oradea (146), Rimnicu Vilcea (220), Lugoj (229), Fagaras (239), Mehadia (299), Pitesti (317), Craiova (366), Drobeta (374), Bucharest (418)
(assuming a check that nodes with states previ- ously generated are not added to the frontier, except when they have lower path cost than a node with that state already on the frontier, in which case the node with higher path cost is removed, so ignore Oradea (291) reached via Sibiu, and Sibiu (297) reached via Zerind and Oradea)

(iv) Iterative deepening:
Arad, Arad, Sibiu, Timisoara, Zerind, Arad, Sibiu, Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Lugoj, Zerind, Oradea, Arad, Sibiu, Fagaras, Bucharest (assuming a cycle check on each path, so omit Arad reached via Arad → Sibiu → Arad)

PART II

Activity 1. For the route from Arad to Bucharest, what order are nodes in the state space expanded for each of the following algorithms when searching for the shortest path between Arad and Bucharest? Where there is a choice of nodes, take the first one by alphabetical ordering. Make sure you understand the key properties of the different algorithms, as listed below.

(v) Greedy:
Arad (366), Sibiu (253), Fagaras (178), Bucharest (0)

(vi) A^*:
Arad (366), Sibiu (393), Rimnicu Vilcea (413), Pitesti (415), Fagaras (417), Bucharest (418)
(unexpanded states on the frontier are Timisoara (447), Zerind (449), Craiova (526), Oradea (671)) – for example, f(Rimnicu Vilcea) = g(Rimnicu Vilcea) + h(Rimnicu Vilcea) = 140 + 80 + 193 = 413
(remember to use the total path cost from Arad to Rimnicu Vilcea here)

Activity 2. 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.

Activity 3. 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 $0 \le w \le 1$, because it is equivalent to A^*Search using the heuristic $h'(n) = [w/(2-w)]h(n) \le h(n)$
When $w > 1$ it is not guaranteed to be optimal (however, it might work very well in practice, for some problems).

PART III

Activity 1. 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 2. 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?

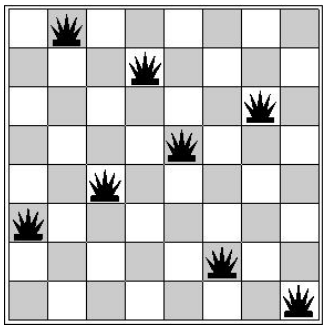
	WA	NT	Q	NSW	V	SA	T
initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
WA=Green	G	R B	R G B	R G B	R G B	R B	R G B
V = Red	G	R B	R G B	G B	R	B	R G B
NT = Red	G	R	G B	G B	R	B	R G B
SA = Blue	G	R	G	G	R	B	R G B
Q = Green	G	R	G		R	B	R G B

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 3. Consider the following state for the 8-queens problem:



- Is this a solution? No, the queens on columns 2 and 5 are attacking each other.
- What is the value of *h*? There is only one violation, so *h*=1.
- 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.)