1. Consider the two-player game described in Figure 7.
   a. Draw the complete game tree, using the following conventions:
      i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
      ii. Put each terminal state in square boxes and write its game value in a circle,
      iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a”?” in a circle.
   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?

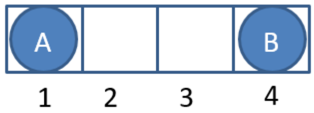
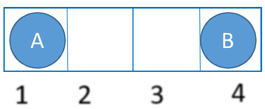


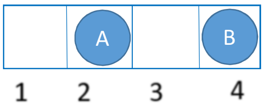
Figure 7.
The starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in either direction. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space if 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.

a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



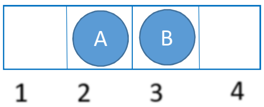
1,4 A

a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



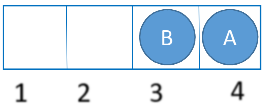
1,4   A
|
2,4   B

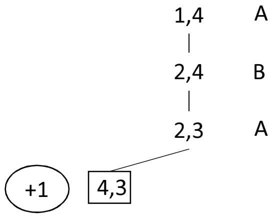
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



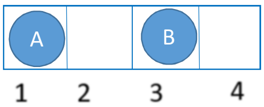
1,4   A
|
2,4   B
|
2,3   A

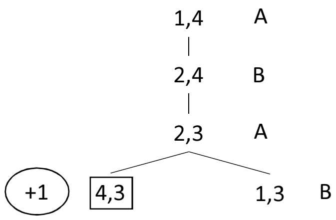
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



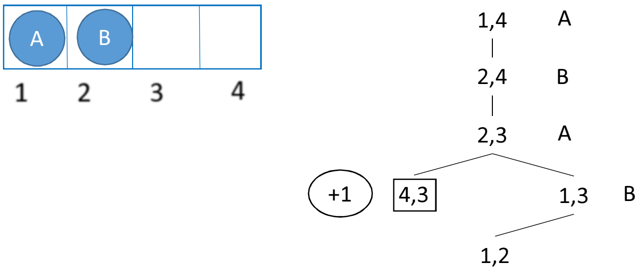


a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.

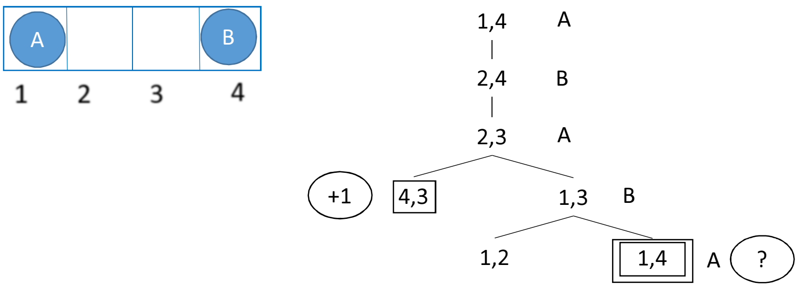




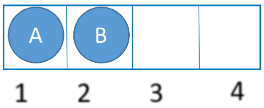
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.

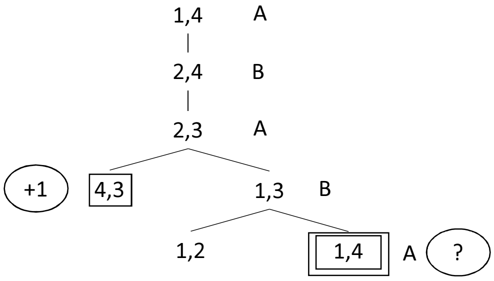


a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



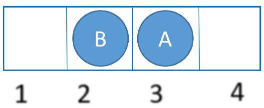
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.

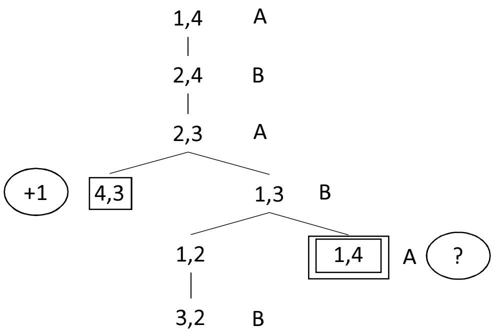




9

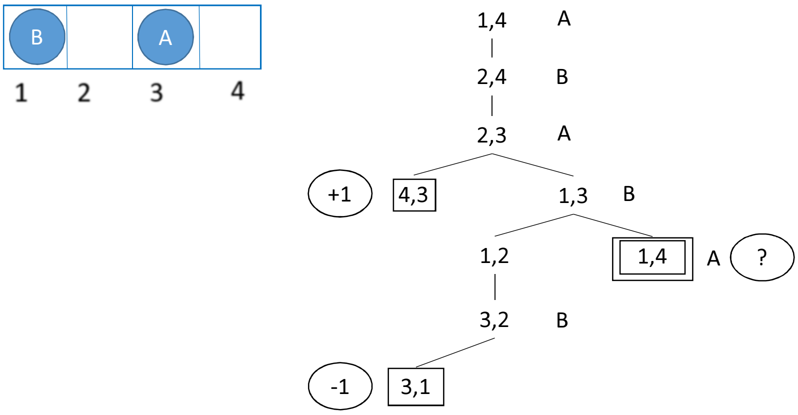
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



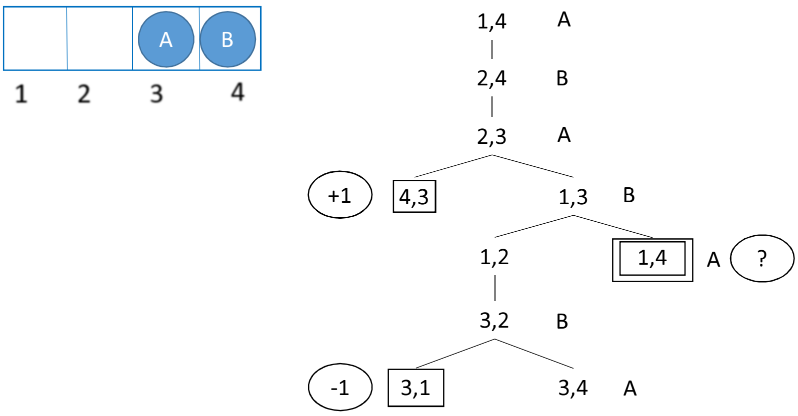


10

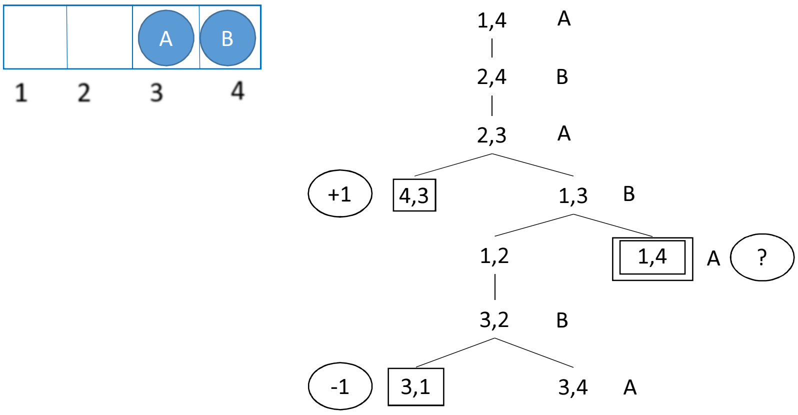
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



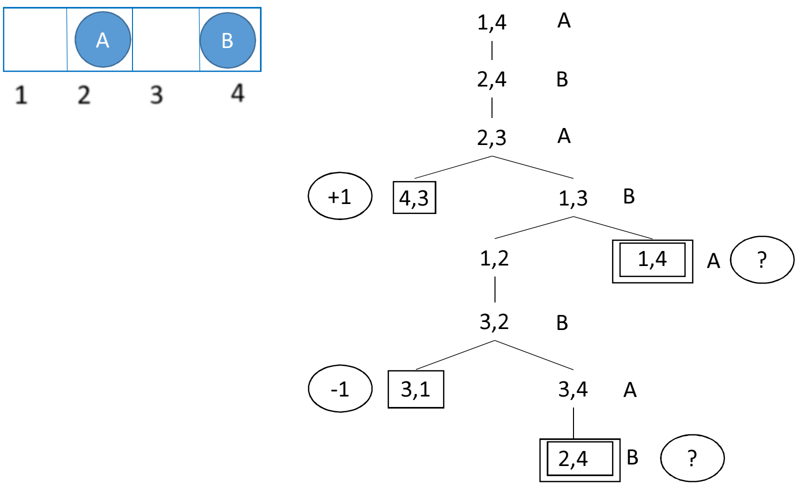
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a ”?” in a circle.



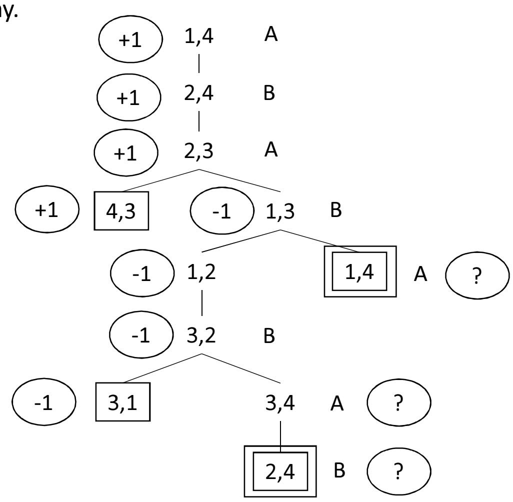
a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a "?" in a circle.



a. Draw the complete game tree, using the following conventions:
   i. Write each state as (s_A, s_B) where s_A and s_B denote the token locations.
   ii. Put each terminal state in a square boxes and write its game value in a circle,
   iii. Put loop states (states that already appear on the path to the root) in double square boxes. Since it is not clear how to assign values to loop states, annotate each with a "?" in a circle.



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



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. 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?

Standard minimax is depth-first and would go into an infinite loop. It can be fixed by comparing the current state against the stack; and if the state is repeated, then return a “?” value. Propagation of “?” values is handled as above.

2. The minimax algorithm assumes that players take turns moving, but in card games such as bridge, the winner of the previous trick plays first on the next trick.
   Modify the algorithm to work properly for these games.

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

Modified MAX-VALUE
…
for each a **in** Actions(state) **do**
 s = RESULT(a, state)
 if(WINNER(s)==MAX)
 then v ← MAX(v, MAX-VALUE(s))
 else v ← MAX(v, MIN-VALUE(s))
…

3. Proof for a positive linear transformation of leaf values (transforming value x to ax+b with a>0), the choice of moves remains unchanged in a game tree with chance nodes.

Suppose that the values of the descendants of a node are $x_1 \ldots x_n$, and that the transformation is ax+b.

min(y+b, z+b)= min (y,z)+b

min(ay, az)= a min (y,z)

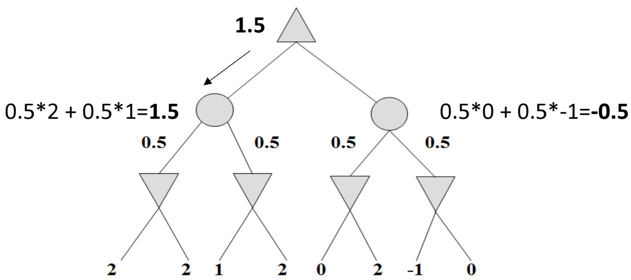
$\min(ax_1+b, ax_2+b, \ldots, ax_n+b) = a \min(x_1, \ldots, x_n)+b$

$\max(ax_1+b, ax_2+b, \ldots, ax_n+b) = a \max(x_1, \ldots, x_n)+b$

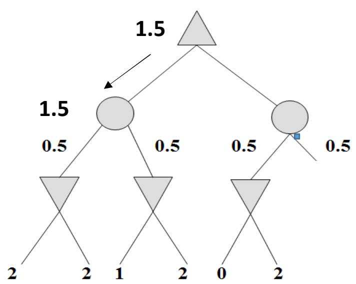
$p_1(ax_1+b)+ \ldots +p_n(ax_n+b) = a\,(p_1x_1+ \ldots +p_nx_n) + \sum_{i=1}^{n} p_i b$

$p_1(ax_1+b)+ \ldots +p_n(ax_n+b) = a\,(p_1x_1+ \ldots +p_nx_n) + b$

4. This question considers pruning in games with chance nodes. The following figure shows the complete game tree for a trivial 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 ∞.
   a. Compute the value of all nodes, and indicate the best move at the root with an arrow.
   b. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaves? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.
   c. 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?
   d. Circle all the leaves that need not be evaluated under the assumption in (c).

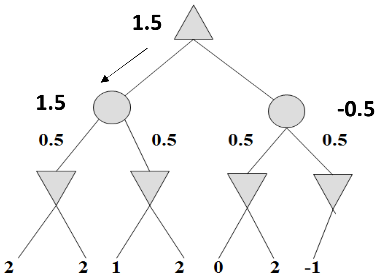


4. This question considers pruning in games with chance nodes. The following figure shows the complete game tree for a trivial game. Assume that the leaf nodes are to be evaluated in left- to-right order, and that before a leaf node is evaluated, we know nothing about its value—the range of possible values is -∞ to ∞.
   b. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaves? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.
   c. 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?
   d. Circle all the leaves that need not be evaluated under the assumption in (c).



Given nodes 1–6, we would need to look at 7 and 8.

4. This question considers pruning in games with chance nodes. The following figure shows the complete game tree for a trivial 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 ∞.
   b. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaves? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.
   c. 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?
   d. Circle all the leaves that need not be evaluated under the assumption in (c).



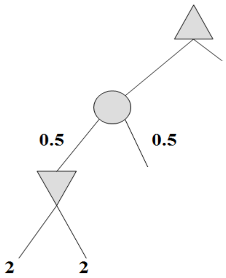
Given nodes 1–6, we would need to look at 7 and 8.

Given nodes 1–7, we do not need to look at 8.

22

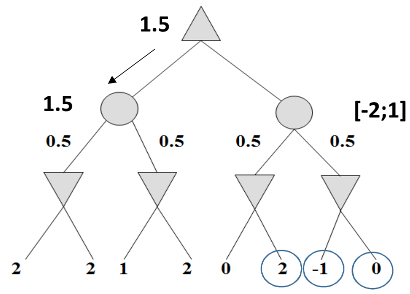
4. This question considers pruning in games with chance nodes. The following figure shows the complete game tree for a trivial game. Assume that the leaf nodes are to be evaluated in left- to-right order, and that before a leaf node is evaluated, we know nothing about its value—the range of possible values is -∞ to ∞.
   c. 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?
   d. Circle all the leaves that need not be evaluated under the assumption in (c).

It must lie between 0 and 2



23

4. This question considers pruning in games with chance nodes. The following figure shows the complete game tree for a trivial 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 ∞.
    d. Circle all the leaves that need not be evaluated under the assumption in (c).



5. Which of the following are true and which are false? Give brief explanations.
    a. In a fully observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what strategy the second player is using— that is, what move the second player will make, given the first player's move.
    b. In a partially observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what move the second player will make, given the first player's move.
    c. A perfectly rational backgammon agent never loses.

a. True

b. False

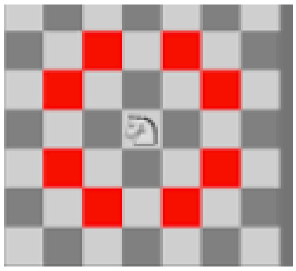
c. False. It's a game of chance

6. Consider the problem of placing k knights on an n×n chessboard such that no two knights are attacking each other, where k is given and k ≤ n
   a. Choose a CSP formulation. In your formulation, what are the variables?
   b. What are the possible values of each variable?
   c. What sets of variables are constrained, and how?.

a. A variable for every board position (n^2)

b. Each of the variables have the domain {occupied, empty}

c. Every pair of squares separated by a knight's move cannot be occupied.
The total number of occupied squares is k.



26

7. Explain the difference of depth first search for non CSP problems and basic backtracking search for CSP problems.

Assignments are commutative. Do not have to think about the order.
Only consider one variable at a node. Which is not true for standard depth first (getAllSuccessors).
Depth-first search for CSPs with single-variable assignments is called backtracking search.

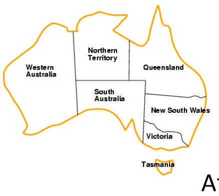
8. Explain what can be done to reduce the state space exploration of standard backtracking search.

Use heuristics for variable selection: MRV, Degree heuristic.
Use heuristics for value selection
Inference

```
function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, value)
        if inferences ≠ failure then
          add inferences to assignment
          result ← BACKTRACK(assignment, csp)
          if result ≠ failure then
             return result
     remove {var = value} and inferences from assignment
  return failure
```

27

9. How many solutions are there for the map-colouring problem from the lecture?



SA has 3 options

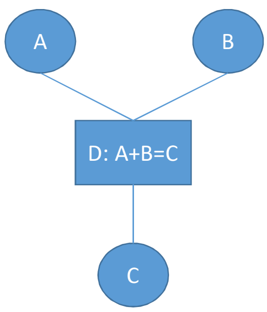
Afterwards, WA has only 2 options

Afterwards all others have only 1 choice → 3*2=6

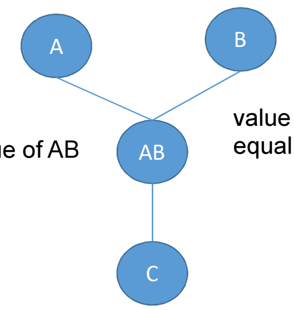
Tasmania has always 3 options, because of independence

→ Total number of solutions = 18

10. Show how a single ternary constraint such as "A + B = C" can be turned into three binary constraints by using an auxiliary variable.



value of A must be equal to the first value of AB



value of B must be equal to the second value of AB

sum of AB must be equal to C