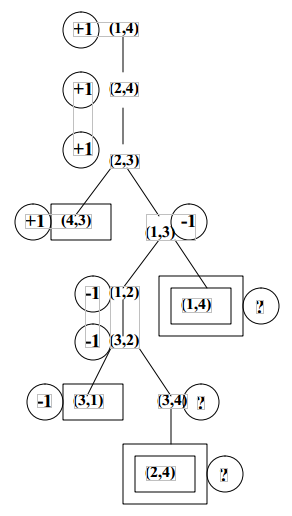
5.8



(a)

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

(b)

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

(c)

Standard minimax 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. In this example, both (1,4) and (2,4) repeat in the tree but they are won positions, If the game tree has cycles, then a dynamic programming method must be used. These algorithms can determine whether each node has a well-determined value or is really an infinite loop in that both players prefer to stay in the loop (or have no choice). In such a case, the rules of the game will need to define the value (otherwise the game will never end).

(d)

The base case n=3 is a loss for A and the base case n=4 is a win for A. For any n > 4, the initial moves are the same: A and B both move one step towards each other.

Now, we can see that they are engaged in a subgame of size n − 2 on the squares [2, . . . , n − 1], except that there is an extra choice of moves on squares 2 and n − 1. Ignoring this for amoment, it is clear that if the “n − 2” is won for A, then A gets to the square n − 1 before B gets to square 2 (by the definition of winning) and therefore gets to n before B gets to 1, hence the “n” game is won for A. By the same line of reasoning, if “n − 2” is won for B then “n” is won for B. Now, the presence of the extra moves complicates the issue, but not too much. First, the player who is slated to win the subgame [2, . . . , n − 1] never moves back to his home square. If the player slated to lose the subgame does so, then it is easy to show that he is bound to lose the game itself—the other player simply moves forward and a subgame of size n − 2k is played one step closer to the loser’s home square.

5.10

(a)

This count doesn’t take into account transpositions. An upper bound on the number of distinct game states is 3N, as each square is either empty or filled by one of the two players. Note that we can determine who is to play just from looking at the board.

(b)

At the end of the game the squares are divided between the two players: ⌈N/2⌉ to the first player and ⌊N/2⌋ to the second. Thus, a good lower bound on the number of distinct states is../Desktop/螢幕快照%202018-12-20%2019.14.32.png ,the number of distinct terminal states.

(c)

For a state s, let X(s) be the number of winning positions containing no O’s and O(s) the number of winning positions containing no X’s. One evaluation function is then Eval(s) = X(s) − O(S). Notice that empty winning positions cancel out in the evaluation function. Alternatively, we might weight potential winning positions by how close they are to completion.

(d) Using the upper bound of N! from (a), and observing that it takes 100NN! instructions. At 2GHz we have 2 billion instructions per second (roughly speaking), so solve for the largest N using at most this many instructions. For one second we get N = 9, for one minute N = 11, and for one hour N = 12.

5.13

(a)

n2 = max(n3, n31, . . . , n3b3), giving

n1 = min(max(n3, n31, . . . , n3b3), n21, . . . , n2b2)

Then n3 can be similarly replaced, until we have an expression containing nj itself.

(b)

n1 = min(l2,max(l3, n3, r3), r2)

Again, n3 can be expanded out down to nj. The most deeply nested term will be min(lj, nj, rj).

(c)

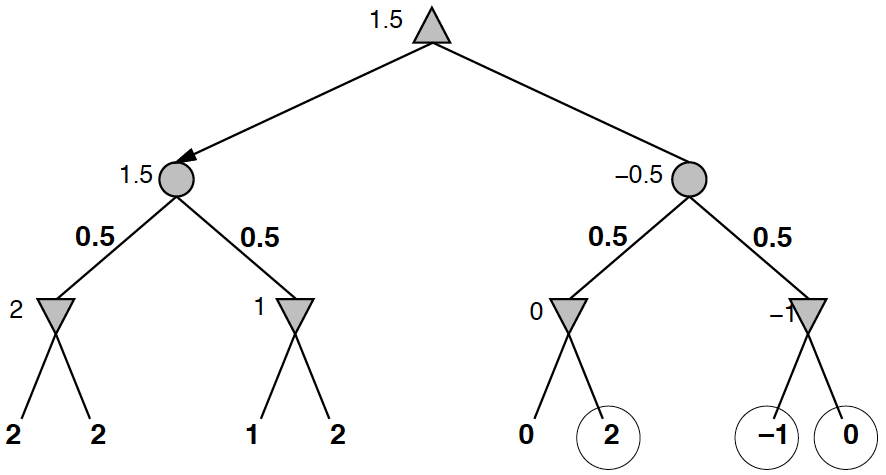
If nj is a max node, then the lower bound on its value only increases as its successors are evaluated. Clearly, if it exceeds lj it will have no further effect on n1. By extension, if it exceeds min(l2, l4, . . . , lj) it will have no effect. Thus, by keeping track of this value we can decide when to prune nj. This is exactly what α-β does.

(d)

The corresponding bound form in nodes nk is max(l3, l5, . . . , lk).

5.16

(a) & (d)



(b)

Given nodes 1–6, we would need to look at 7 and 8: if they were both +∞ then the values of the min node and chance node above would also be +∞ and the best move would change. Given nodes 1–7, we do not need to look at 8. Even if it is +∞, them in node cannot be worth more than −1, so the chance node above cannot be worth more than −0.5, so the best move won’t change.

(c)

The worst case is if either of the third and fourth leaves is −2, in which case the chance node above is 0. The best case is where they are both 2, then the chance node has value 2. So it must lie between 0 and 2.

5.20

(a)

No pruning. In a max tree, the value of the root is the value of the best leaf. Any unseen leaf might be the best, so we have to see them all.

(b)

No pruning. An unseen leaf might have a value arbitrarily higher or lower than any other leaf, which (assuming non-zero outcome probabilities) means that there is no bound on the value of any incompletely expanded chance or max node.

(c)

same as (a)

(d)

No pruning. Nonnegative values allow lower bounds on the values of chance nodes, but a lower bound does not allow any pruning.

(e)

Yes. If the first successor has value 1, the root has value 1 and all remaining successors can be pruned.

(f)

Yes. Suppose the first action at the root has value 0.6, and the first outcome of the second action has probability 0.5 and value 0; then all other outcomes of the second action can be pruned.

(g)

(ii) Highest probability first. This gives the strongest bound on the value of the node, all other things being equal.

5.21

(a)

True. The second player will play optimally, and so is perfectly predictable up to ties. Knowing which of two equally good moves the opponent will make does not change the value of the game to the first player.

(b)

False. By knowing the second player’s strategy still can increase the first player’s chances of winning.

(c)

False. Backgammon is game of chance, rolling dice will produce random results, which can not be predicted, and makes backgammon a stochastic game.