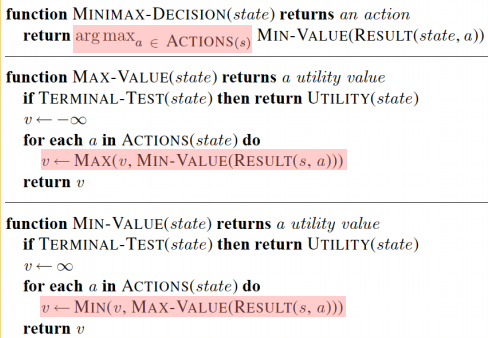
**Homework assignment#3 (Chap5)**

106971001 林上人

1. **Pseudo codes documentation**

Pages: 19

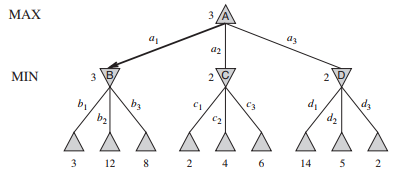


③

②

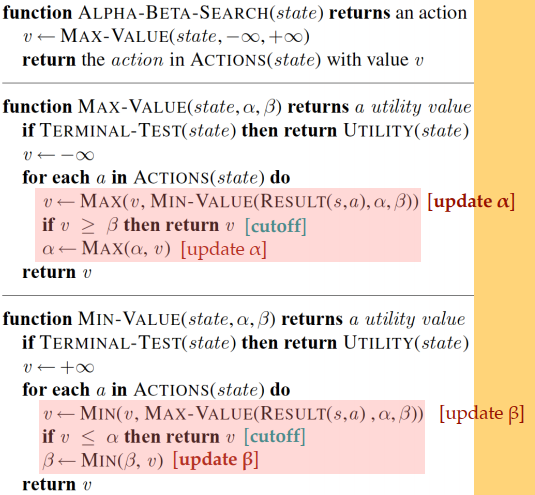
①

Minimax 演算法 – 在最壞情況中謀求最大利益

此演算法假設雙人對局的情形，且對方一定會選擇讓我方情況最差的動作，因此通常情形會如下圖所示，一層為MAX一層為MIN交替產生，MAX表

示我們要從後面取得最大值，MIN表示對方要從後面取得最小值的情形，➀的部分為root表示我們要得到能夠從MIN-VALUE函數產生max值的動作參數，後續➁、➂先檢查狀態是否結束，例如遊戲以分出勝負或是其他狀況，然後根據所在的層，➁MAX-VALUE函數就從後續的MIN-VALUE回傳的值回傳最大值，➂MIN-VALUE函數就從後續的MAX-VALUE回傳的值回傳最小值。

Pages: 33



①

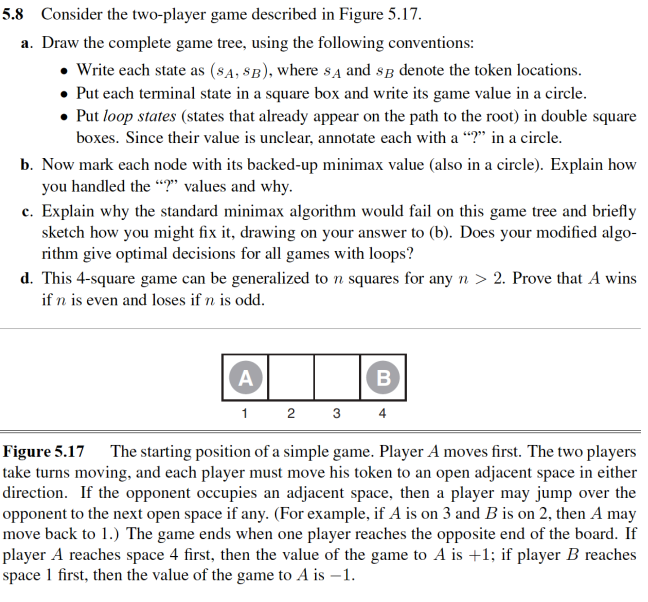
③

②

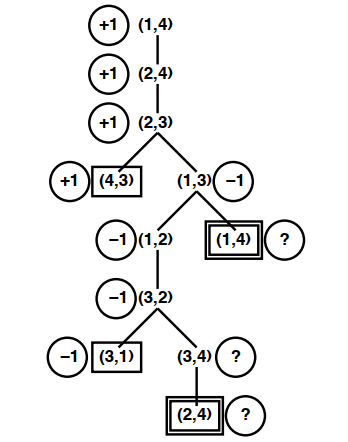
ALPHA-BETA-SEARCH

此處是改善原本的MINMAX演算法，從而不需將每一個節點都拓展，以達到優化的目的，主要是在MINIMAX中加入α、β做紀錄，作為是否pruning的標準，其中α代表到目前為止極大值的最佳解，β代表到目前為止極小值的最佳解，以MAX層來看➀因為在MAX層時會從下一層的MIN-VALUE回傳的值中取最大值，所以➁下一層的MIN-VALUE在計算每一個MAX-VALUE(RESULT(s,a))時，➂若有出現比α值低的就表示在這一層(MIN層)的這一個分支回傳值會小於α，而這個分支回到MAX層時就不會被選中，因此可以不用繼續檢查其他MAX-VALUE(RESULT(s,a))直接回傳υ，以MIN層來看道理相同，只是改成檢查β值，並且在MAX節點更新α值，在MIN節點更新β值。

1. **Exercise**



(a)



MAX

MAX

MAX

MAX

MIN

MIN

MIN

MIN

(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.

In this example, both (1,4) and (2,4) repeat in the tree but they are won positions. Although it works in this case, it does not always work because it is not clear how to compare “?” with a drawn position; nor is it clear how to handle the comparison when there are wins of different degrees.

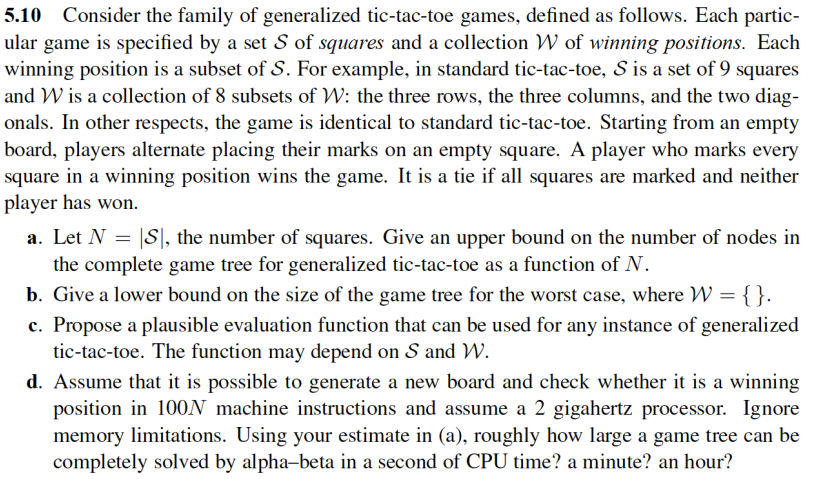
(d)

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.

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.



(a)

Upper bound : N!

One for each ordering of squares, so an upper bound on the total number of nodes is . This is not much bigger than N! itself as the factorial function grows super-exponentiallly. This is an overestimate because some games will end early when a winning position is filled.

(b)

In this case no games terminate early, and there are N! different games ending in a draw.

So ignoring repeated states, we have nodes.

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 ,the number of distinct terminal states.

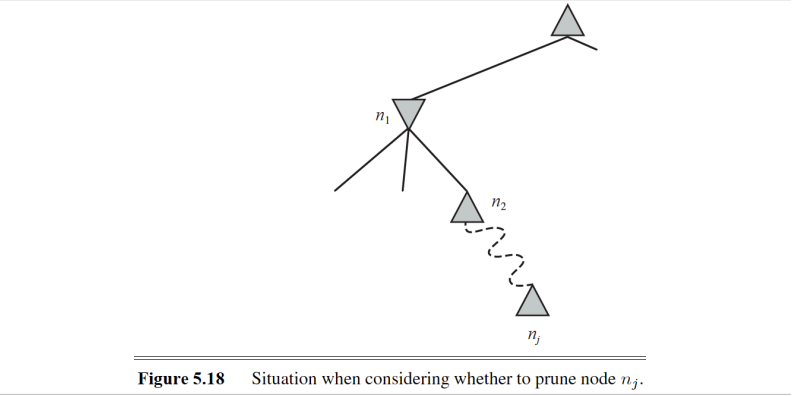
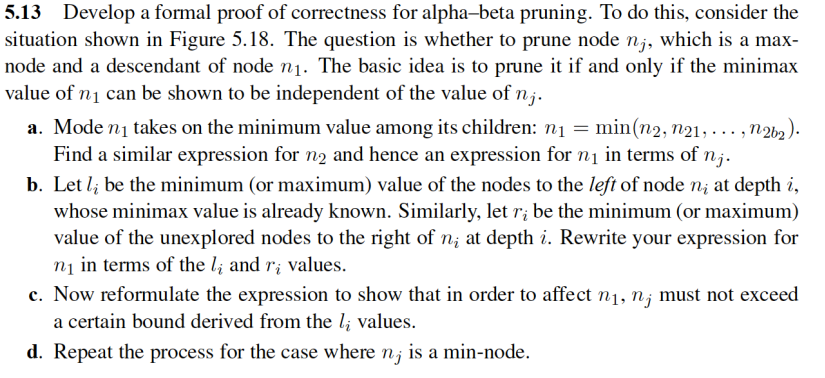
(c)

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, 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.



(a)

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

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

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

(b)

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

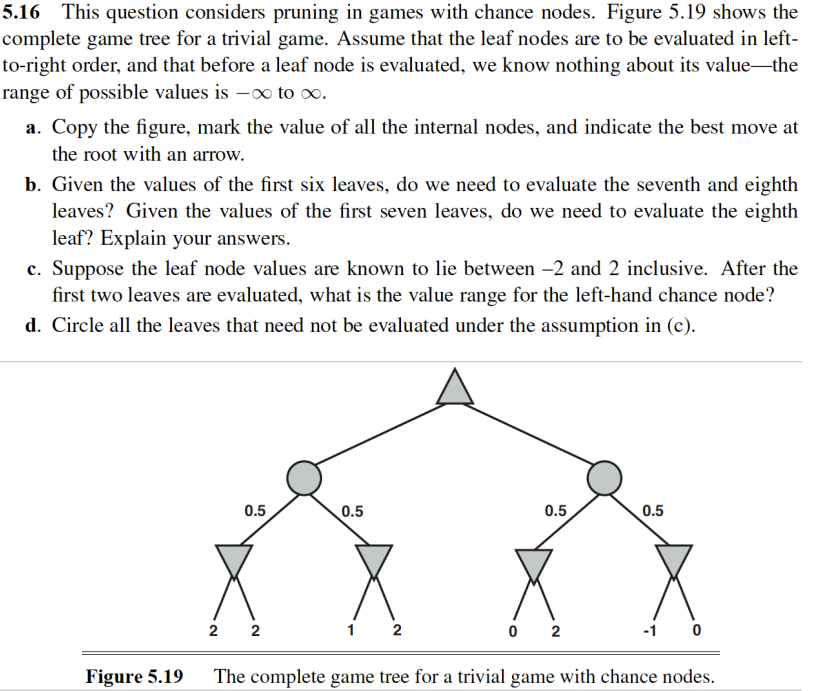
n3 can be expanded out down to nj. The most deeply nested term will be min(lj, nj, rj).

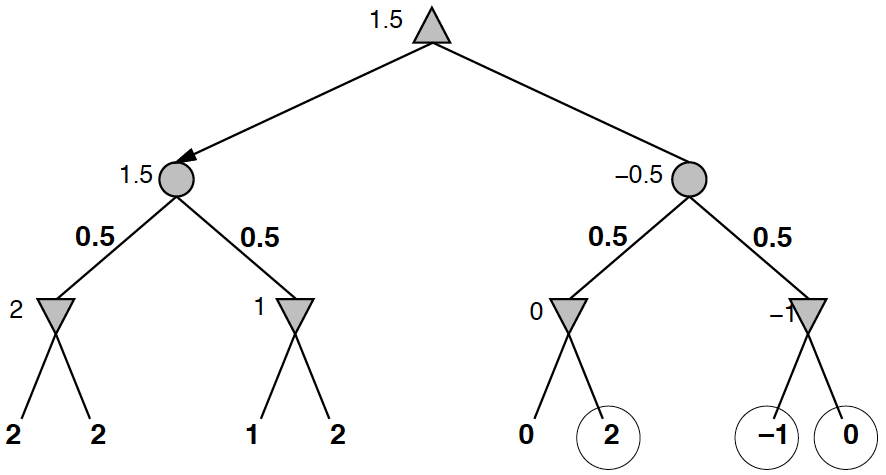
(c)

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.

(d)

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

 (a) & (d)

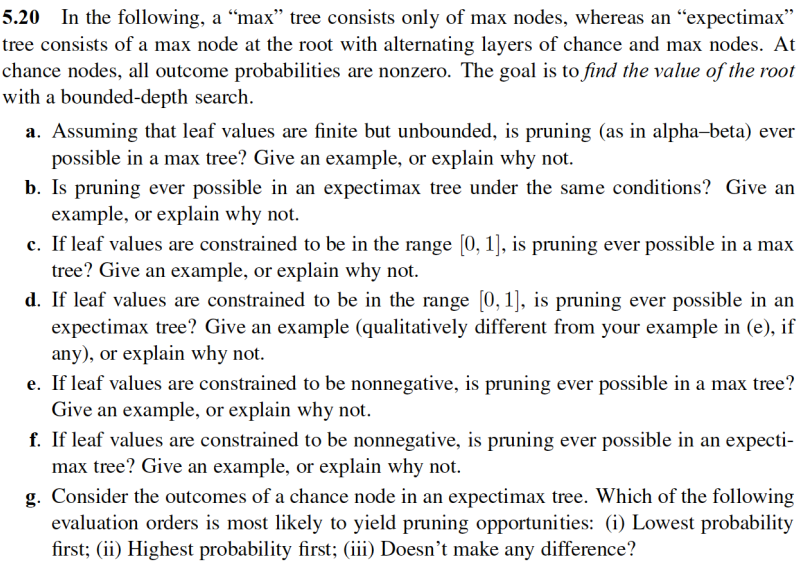


(b)

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.

 (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)

No pruning.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)

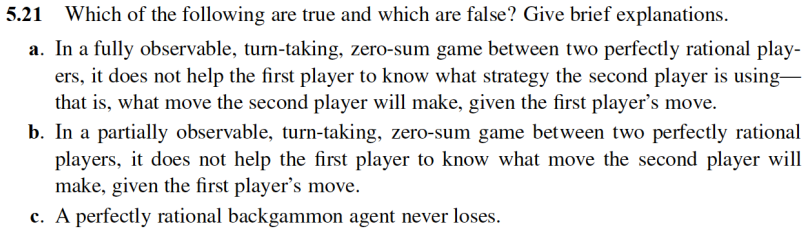
Yes. If the first successor has value 1, the root has value 1 and all remaining successors can be pruned.

(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.

 (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, and the opponent may consistently roll much

better dice.