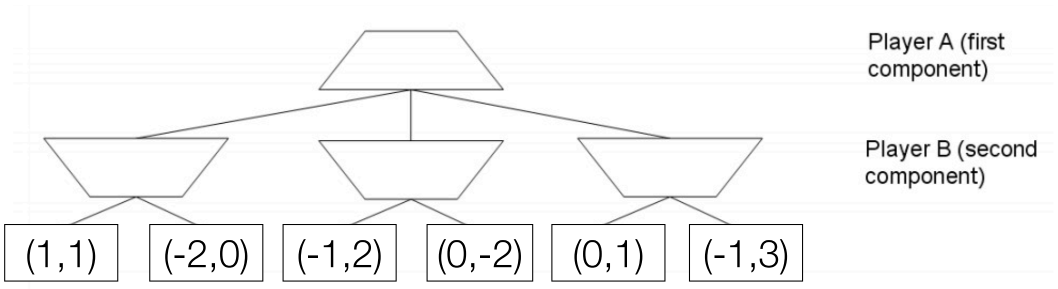
CS188 Fall 2013 Section 3: Games

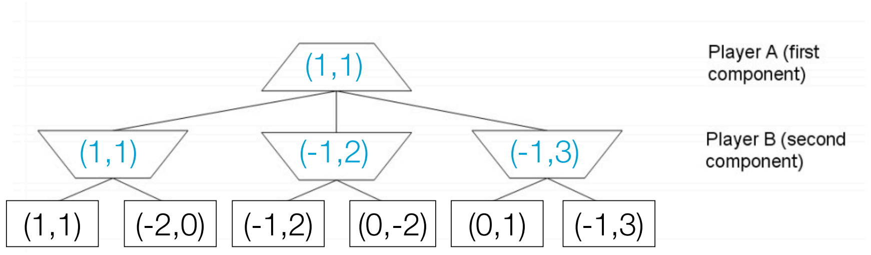
1 Nearly Zero Sum Games

The standard Minimax algorithm calculates worst-case values in a *zero-sum* two player game, i.e. a game in which for all terminal states s, the utilities for players A (MAX) and B (MIN) obey $U_A(s) + U_B(s) = 0$. In the zero sum case, we know that $U_A(s) = -U_B(s)$ and so we can think of player B as simply minimizing $U_A(s)$.

In this problem, you will consider the *non zero-sum* generalization in which the sum of the two players' utilities are not necessarily zero. Because player A's utility no longer determines player B's utility exactly, the leaf utilities are written as pairs $(U_A; U_B)$, with the first and second component indicating the utility of that leaf to A and B respectively. In this generalized setting, A seeks to maximize U_A, the first component, while B seeks to maximize U_B, the second component.



1. Propagate the terminal utility pairs up the tree using the appropriate generalization of the minimax algorithm on this game tree. Fill in the values (as pairs) at each of the internal node. Assume that each player maximizes their own utility.



2. Briefly explain why no alpha-beta style pruning is possible in the general non-zero sum case.
Hint: think first about the case where $U_A(s) = U_B(s)$ for all nodes.

   The values that the first and second player are trying to maximize are independent, so we no longer have situations where we know that one player will never let the other player down a particular branch of the game tree.

   For instance, in the case where $U_A = U_B$, the problem reduces to searching for the max-valued leaf, which could appear anywhere in the tree.

3. For minimax, we know that the value v computed at the root (say for player A = MAX) is a worst-case value. This means that if the opponent MIN doesn't act optimally, the actual outcome v' for MAX can only be better, never worse than v.

   In the general non-zero sum setup, can we say that the value U_A computed at the root for player A is also a worst-case value in this sense, or can A's outcome be worse than the computed U_A if B plays sub-optimally? Briefly justify.

   A's outcome can be worse than the computed v_A. For instance, in the example game, if B chooses $(-2, 0)$ over $(1, 1)$, then A's outcome will decrease from 1 to 0.

4. Now consider the nearly zero sum case, in which $|U_A(s) + U_B(s)| \leq \epsilon$ at all terminal nodes s for some ϵ which is known in advance. For example, the previous game tree is nearly zero sum for $\epsilon = 2$.

   In the nearly zero sum case, pruning is possible. Draw an X in each node in this game tree which could be pruned with the appropriate generalization of alpha-beta pruning. Assume that the exploration is being done in the standard left to right depth-first order and the value of ϵ is known to be 2. Make sure you make use of ϵ in your reasoning.

   We can prune the node $(0, -2)$ and if we allow pruning on equality then we can also prune $(-1, 3)$. See answers to the next two problems for the reasoning.

5. Give a general condition under which a child n of a B node (MIN node) b can be pruned. Your condition should generalize α-pruning and should be stated in terms of quantities such as the utilities $U_A(s)$ and/or $U_B(s)$ of relevant nodes s in the game tree, the bound ϵ, and so on. Do not worry about ties.

   The pruning condition is $\boxed{U_B > \epsilon - \alpha}$.

   Consider the standard minimax algorithm (zero-sum game) written in this more general 2 agent framework. The maximizer agent tries to maximize its utility, U_A, while the second agent (B) tries to minimize player A's value. This is equivalent to saying that player B wants to maximize $-U_A$. Therefore we say that the utility of player B is $U_B = -U_A$ in the standard minimax situation.

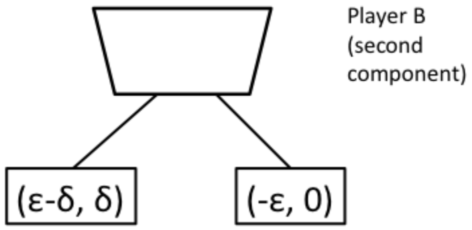
   Recall from lecture that in standard $\alpha - \beta$ pruning we allow a pruning action to occur under a minimizer (player B) node if $v < \alpha$. Under our more general 2 agent framework this condition is equivalent to saying you can prune under player B if $U_A = -U_B < \alpha \Rightarrow U_B > -\alpha$.

   For this question we have an ϵ-sum game so we need to add an additional requirement of ϵ on U_B before pruning can occur. In particular, we know that $|U_A + U_B| \leq \epsilon \Rightarrow U_A \leq \epsilon - U_B$. We want to prune if this upper bound is less than α because then we guarantee that max has an better alternative elsewhere in the tree. Therefore, in order to prune we must satisfy $\epsilon - U_B < \alpha \Rightarrow U_B > \epsilon - \alpha$.

6. In the nearly zero sum case with bound ϵ, what guarantee, if any, can we make for the actual outcome u' for player A (in terms of the value U_A of the root) in the case where player B acts sub-optimally?

$\boxed{u' \geq U_A - 2\epsilon}$

To get intuition about this problem we will first think about the worst case scenario that can occur for player A. Consider the small game tree below for $\delta > 0$:



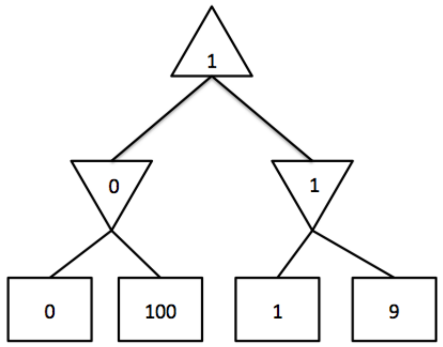
The optimal action for player B to take would be $(\epsilon - \delta, \delta)$. If player B acts optimally then player A will end up with a value of $U_A = \epsilon - \delta$. Now, consider what would happen if player B acted suboptimally, namely if player B chose $(-\epsilon, 0)$. Then player A would receive an actual outcome of $u' = -\epsilon$. So, we see that $u' \geq U_A - 2\epsilon + \delta$. Now let δ be arbitrarily small and you converge to the bound boxed above.

Thus far we have just shown (by example) that we cannot hope for a better guarantee than $u' \geq U_A - 2\epsilon$ (if someone claimed a better guarantee, the above would be a counter-example to that (faulty) claim). We are left with showing that this bound actually holds true. To do so, consider what happens when Player B plays suboptimally. By definition of suboptimality, that means the outcome of the game for player B is not the optimal U_B but some lower value $U'_B = U_B x$ with $x > 0$. This will have the maximum effect on player A's pay-off when for the optimal outcome we had $U_A + U_B = \epsilon$, but for the suboptimal outcome we have $U'_A + U'_B = U'_A + U_B x = \epsilon$. From the first equation we have $U_B = \epsilon U_A$, substituting into the second equation gives us: $U'_A = \epsilon\epsilon + U_A = U_A 2\epsilon$ as the worst-case outcome for player A.

2 Minimax and Expectimax

In this problem, you will investigate the relationship between expectimax trees and minimax trees for zero-sum two player games. Imagine you have a game which alternates between player 1 (max) and player 2. The game begins in state s_0, with player 1 to move. Player 1 can either choose a move using minimax search, or expectimax search, where player 2's nodes are chance rather than min nodes.

1. Draw a (small) game tree in which the root node has a larger value if expectimax search is used than if minimax is used, or argue why it is not possible.



   We can see here that the above game tree has a root value of 1 for the minimax strategy. If we instead switch to expectimax and replace the min nodes with chance nodes, the root of the tree takes on a value of 50 and the optimal action changes for MAX.

2. Draw a (small) game tree in which the root node has a larger value if minimax search is used than if expectimax is used, or argue why it is not possible.

   Optimal play for MIN, by definition, means the best moves for MIN to obtain the lowest value possible. Random play includes moves that are not optimal. Assuming there are no ties (no two leaves have the same value), expectimax will always average in suboptimal moves. Averaging a suboptimal move (for MIN) against an optimal move (for MIN) will always increase the expected outcome.

   With this in mind, we can see how there is no game tree where the value of the root for expectimax is lower than the value of the root for minimax. One is optimal play – the other is suboptimal play averaged with optimal play, which by definiton leads to a higher value for MIN.

3. Under what assumptions about player 2 should player 1 use minimax search rather than expectimax search to select a move?

   Player 1 should use minimax search if he/she expects player 2 to move optimally.

4. Under what assumptions about player 2 should player 1 use expectimax search rather than minimax search?

   If player 1 expects player 2 to move randomly, he/she should use expectimax search. This will optimize for the maximum expected value.

5. Imagine that player 1 wishes to act optimally (rationally), and player 1 knows that player 2 also intends to act optimally. However, player 1 also knows that player 2 (mistakenly) believes that player 1 is moving uniformly at random rather than optimally. Explain how player 1 should use this knowledge to select a move. Your answer should be a precise algorithm involving a game tree search, and should include a sketch of an appropriate game tree with player 1's move at the root. Be clear what type of nodes are at each ply and whose turn each ply represents.

   Use two games trees:

   Game tree 1: max is replaced by a chance node. Solve this tree to find the policy of MIN.

   Game tree 2: the original tree, but MIN doesn't have any choices now, instead is constrained to follow the policy found from Game Tree 1.