D3: Games

Monday, October 8, 2018 1:50 PM

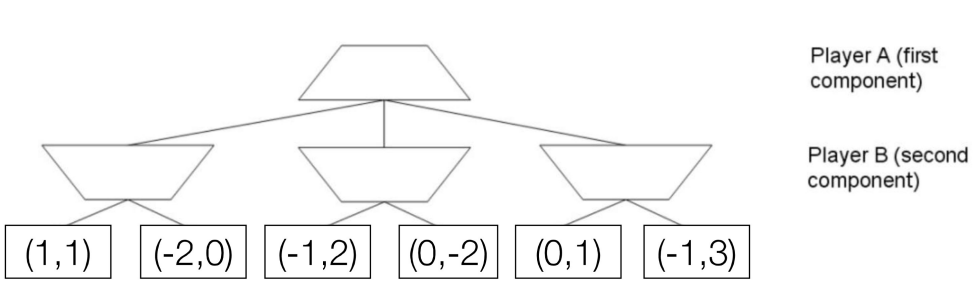
section_3_cNsigDlFSSZgxKANwOsb0axJggk4Uf

CS188 Spring 2014 Section 3: Games

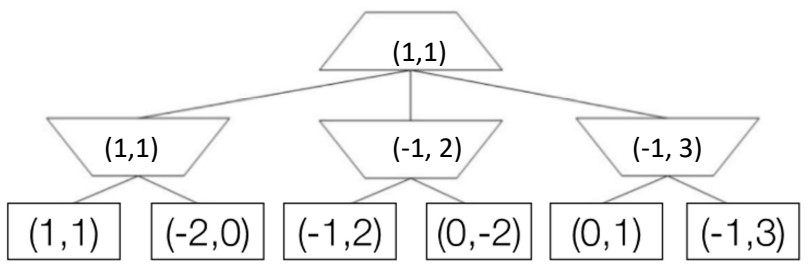
1 Nearly Zero Sum Games

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.

Alpha -beta pruning is when an ancestor node passes down to it's descendants the best current value available to it. Search can be pruned when it is determined that the utility of a node is certain to be less than the best available utility for that ancestor.

What makes AB pruning work is when an antagonistic descendant will generate a utility that is guaranteed to be worse for the parent. However, for non-zero sum games, there is ALWAYS the possibility of a WIN-WIN situation at any node, therefore, we cannot prune or we may miss optimal values

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.

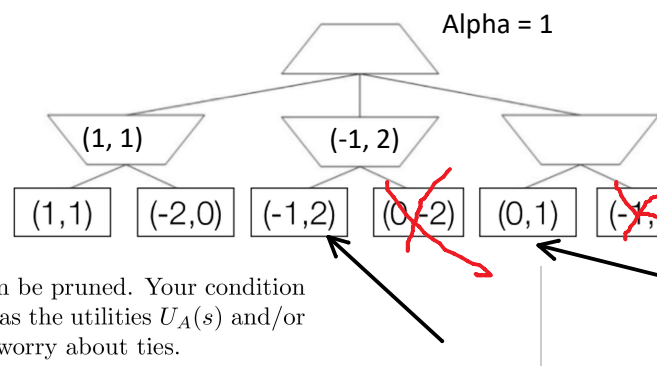
   A's outcome can be worse than predicted in a non-zero sum game if it turns out that Player B plays erratically.

   For example, consider the case where $U_A = U_B$. Which means it is a purely cooperative game. If U_B instead acts erratically, our result will only be worse than predicted

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.

   Suppose net utility is at most epsilon and at worst -epsilon.



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.

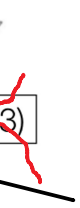
   Suppose we have a max node that knows it can get at least some value Alpha.
   At some point there is a min node descendant that has children to choose utilities from.
   If, for any child, the MIN node b (agent B's choice), see's a child n, where $U_B(n) >= 2 - alpha$.
   Then we know we can prune at b. This is because agent B at b knows it can get at LEAST 2 - alpha, which means at MOST $U_A(n)$ is -(2 -alpha) +2 = alpha. It will only accept nodes that have $U_B(n) > 2-alpha$, in which case $U_A(n)$ must be < alpha.

   Agent B can get at
   The most agent A

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?

   $U_A+U_B <$ epsilon
   $U_A < -U_B +$ epsilon

   If U_B acts suboptimally, U_A is at LEAST $U_B - 4*$epsilon



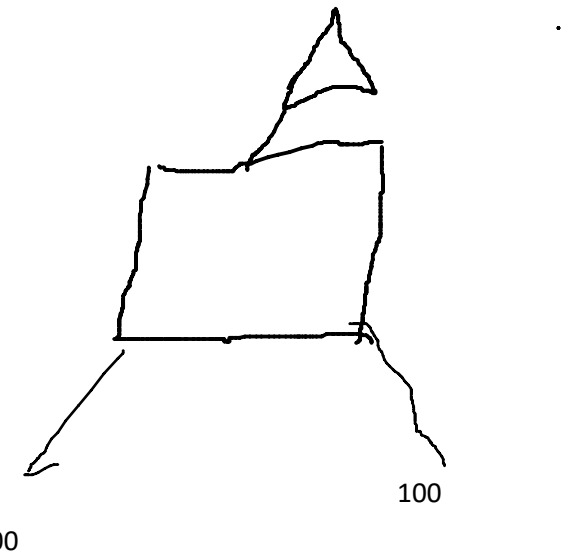
Agent B can get at least 1. So Agent B will never accept anything less than 1. Therefore, Agent A can get at most 1. Agent A already has 1, so, it doesn't care about what goes on in here

least a utility of two.
can get is 0. 0 < alpha, prune

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.



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.

If we replace expectimax with minimax, the value of these nodes
Can ONLY decrease. Therefore it will always be worse.

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

Use minimax in zero-sum games with intelligent adversarial agents

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

Use expectimax when agents aren't intelligent (you might get away with beign greedier)

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.

Player 2 is using expecti-min
Player 1 is a maximizing agent.
Player 1 should use "expecti-min"i-max