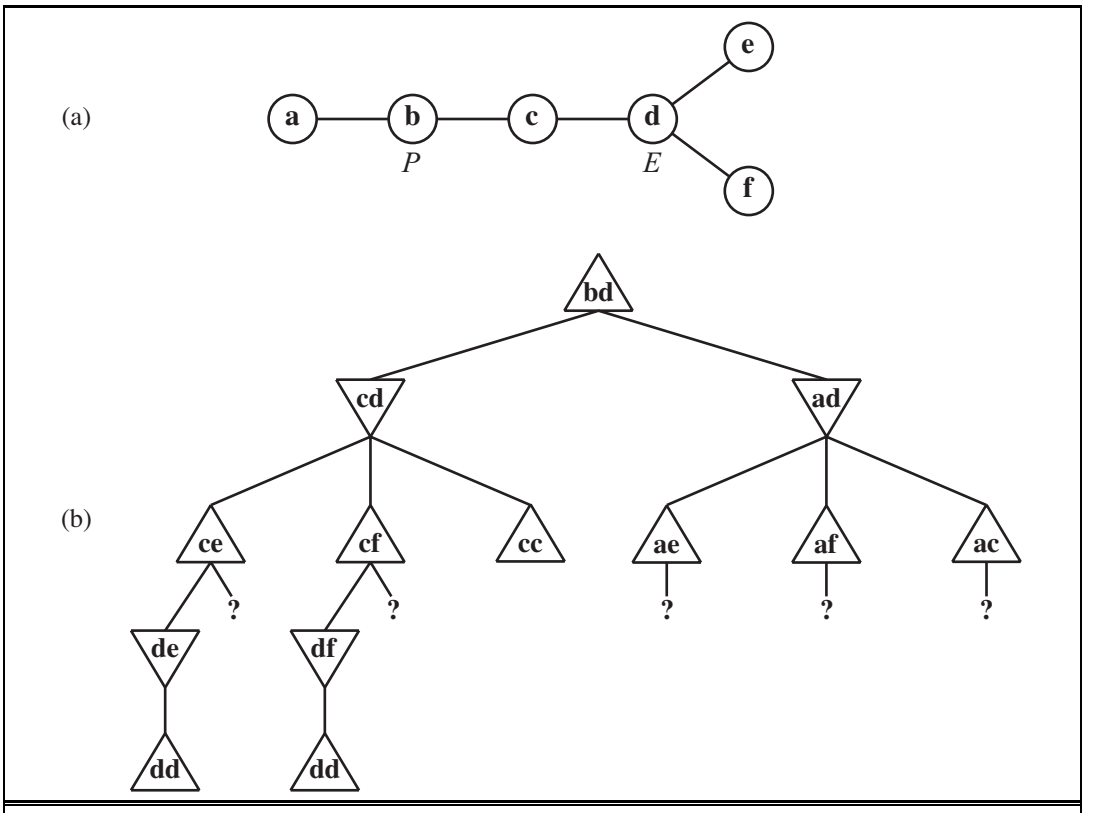


Figure 5.16 (a) A map where the cost of every edge is 1. Initially the pursuer P is at node **b** and the evader E is at node **d**. (b) A partial game tree for this map. Each node is labeled with the P, E positions. P moves first. Branches marked "?" have yet to be explored.

PURSUIT–EVASION

5.3 Imagine that, in Exercise 3.3, one of the friends wants to avoid the other. The problem then becomes a two-player **pursuit–evasion** game. We assume now that the players take turns moving. The game ends only when the players are on the same node; the terminal payoff to the pursuer is minus the total time taken. (The evader "wins" by never losing.) An example is shown in Figure 5.16.

a. Copy the game tree and mark the values of the terminal nodes.

b. Next to each internal node, write the strongest fact you can infer about its value (a number, one or more inequalities such as "≥ 14", or a "?").

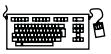
c. Beneath each question mark, write the name of the node reached by that branch.

d. Explain how a bound on the value of the nodes in (c) can be derived from consideration of shortest-path lengths on the map, and derive such bounds for these nodes. Remember the cost to get to each leaf as well as the cost to solve it.

e. Now suppose that the tree as given, with the leaf bounds from (d), is evaluated from left to right. Circle those "?" nodes that would *not* need to be expanded further, given the bounds from part (d), and cross out those that need not be considered at all.

f. Can you prove anything in general about who wins the game on a map that is a tree?

5.4 Describe and implement state descriptions, move generators, terminal tests, utility functions, and evaluation functions for one or more of the following stochastic games: Monopoly, Scrabble, bridge play with a given contract, or Texas hold'em poker.



5.5 Describe and implement a *real-time*, *multiplayer* game-playing environment, where time is part of the environment state and players are given fixed time allocations.

5.6 Discuss how well the standard approach to game playing would apply to games such as tennis, pool, and croquet, which take place in a continuous physical state space.

5.7 Prove the following assertion: For every game tree, the utility obtained by MAX using minimax decisions against a suboptimal MIN will be never be lower than the utility obtained playing against an optimal MIN. Can you come up with a game tree in which MAX can do still better using a *suboptimal* strategy against a suboptimal MIN?

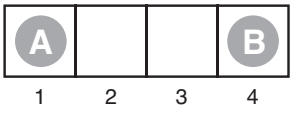


Figure 5.17 The starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in either direction. If the opponent occupies an adjacent space, then a player may jump over the opponent to the next open space if 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.

5.8 Consider the two-player game described in Figure 5.17.

a. Draw the complete game tree, using the following conventions:

- Write each state as (s_A, s_B), where s_A and s_B denote the token locations.
- Put each terminal state in a square box and write its game value in a circle.
- Put *loop states* (states that already appear on the path to the root) in double square boxes. Since their value is unclear, 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?

d. This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

5.9 This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals

with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. The utility function assigns $+1$ to any position with $X_3 = 1$ and -1 to any position with $O_3 = 1$. All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as $Eval(s) = 3X_2(s)+X_1(s)-(3O_2(s)+O_1(s))$.

a. Approximately how many possible games of tic-tac-toe are there?

b. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.

c. Mark on your tree the evaluations of all the positions at depth 2.

d. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.

e. Circle the nodes at depth 2 that would *not* be evaluated if alpha–beta pruning were applied, assuming the nodes are generated in the optimal order for alpha–beta pruning.

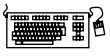
5.10 Consider the family of generalized tic-tac-toe games, defined as follows. Each particular game is specified by a set $\mathcal{S}$ of *squares* and a collection $\mathcal{W}$ of *winning positions*. Each winning position is a subset of $\mathcal{S}$. For example, in standard tic-tac-toe, $\mathcal{S}$ is a set of 9 squares and $\mathcal{W}$ is a collection of 8 subsets of $\mathcal{W}$: the three rows, the three columns, and the two diagonals. In other respects, the game is identical to standard tic-tac-toe. Starting from an empty board, players alternate placing their marks on an empty square. A player who marks every square in a winning position wins the game. It is a tie if all squares are marked and neither player has won.

a. Let $N = |\mathcal{S}|$, the number of squares. Give an upper bound on the number of nodes in the complete game tree for generalized tic-tac-toe as a function of N.

b. Give a lower bound on the size of the game tree for the worst case, where $\mathcal{W} = \{\}$.

c. Propose a plausible evaluation function that can be used for any instance of generalized tic-tac-toe. The function may depend on $\mathcal{S}$ and $\mathcal{W}$.

d. Assume that it is possible to generate a new board and check whether it is a winning position in $100N$ machine instructions and assume a 2 gigahertz processor. Ignore memory limitations. Using your estimate in (a), roughly how large a game tree can be completely solved by alpha–beta in a second of CPU time? a minute? an hour?



5.11 Develop a general game-playing program, capable of playing a variety of games.

a. Implement move generators and evaluation functions for one or more of the following games: Kalah, Othello, checkers, and chess.

b. Construct a general alpha–beta game-playing agent.

c. Compare the effect of increasing search depth, improving move ordering, and improving the evaluation function. How close does your effective branching factor come to the ideal case of perfect move ordering?

d. Implement a selective search algorithm, such as B* (Berliner, 1979), conspiracy number search (McAllester, 1988), or MGSS* (Russell and Wefald, 1989) and compare its performance to A*.

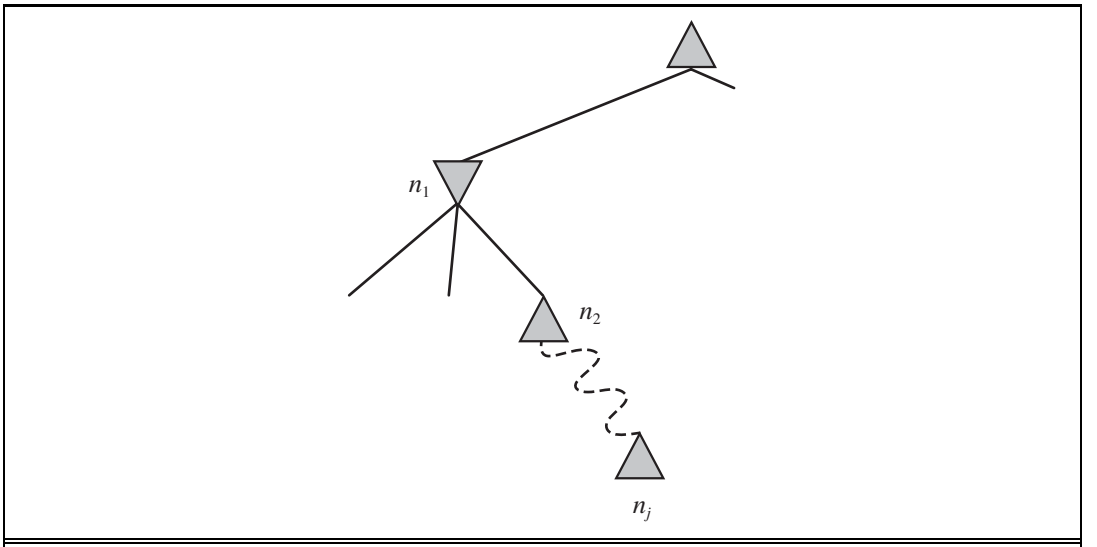


Figure 5.18 Situation when considering whether to prune node n_j.

5.12 Describe how the minimax and alpha–beta algorithms change for two-player, non-zero-sum games in which each player has a distinct utility function and both utility functions are known to both players. If there are no constraints on the two terminal utilities, is it possible for any node to be pruned by alpha–beta? What if the player's utility functions on any state differ by at most a constant k, making the game almost cooperative?

5.13 Develop a formal proof of correctness for alpha–beta pruning. To do this, consider the situation shown in Figure 5.18. The question is whether to prune node n_j, which is a max-node and a descendant of node n_1. The basic idea is to prune it if and only if the minimax value of n_1 can be shown to be independent of the value of n_j.

a. Mode n_1 takes on the minimum value among its children: $n_1 = \min(n_2, n_{21}, \ldots, n_{2b_2})$. Find a similar expression for n_2 and hence an expression for n_1 in terms of n_j.

b. Let l_i be the minimum (or maximum) value of the nodes to the *left* of node n_i at depth i, whose minimax value is already known. Similarly, let r_i be the minimum (or maximum) value of the unexplored nodes to the right of n_i at depth i. Rewrite your expression for n_1 in terms of the l_i and r_i values.

c. Now reformulate the expression to show that in order to affect n_1, n_j must not exceed a certain bound derived from the l_i values.

d. Repeat the process for the case where n_j is a min-node.

5.14 Prove that alpha–beta pruning takes time $O(2^{m/2})$ with optimal move ordering, where m is the maximum depth of the game tree.

5.15 Suppose you have a chess program that can evaluate 10 million nodes per second. Decide on a compact representation of a game state for storage in a transposition table. About how many entries can you fit in a 2-gigabyte in-memory table? Will that be enough for the

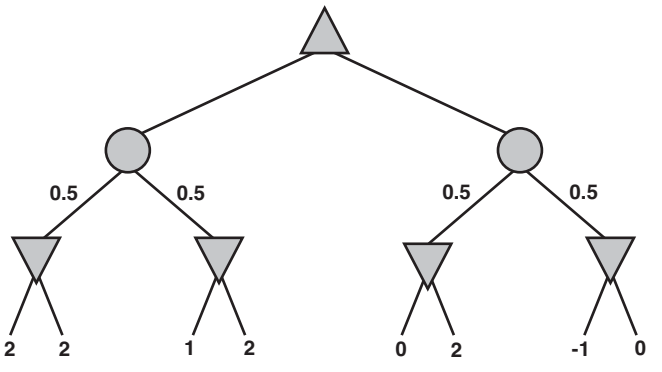


Figure 5.19 The complete game tree for a trivial game with chance nodes.

three minutes of search allocated for one move? How many table lookups can you do in the time it would take to do one evaluation? Now suppose the transposition table is stored on disk. About how many evaluations could you do in the time it takes to do one disk seek with standard disk hardware?

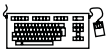
5.16 This question considers pruning in games with chance nodes. Figure 5.19 shows the complete game tree for a trivial game. Assume that the leaf nodes are to be evaluated in left-to-right order, and that before a leaf node is evaluated, we know nothing about its value—the range of possible values is $-\infty$ to ∞.

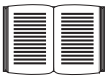
a. Copy the figure, mark the value of all the internal nodes, and indicate the best move at the root with an arrow.

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).





5.17 Implement the expectiminimax algorithm and the *-alpha–beta algorithm, which is described by Ballard (1983), for pruning game trees with chance nodes. Try them on a game such as backgammon and measure the pruning effectiveness of *-alpha–beta.

5.18 Prove that with a positive linear transformation of leaf values (i.e., transforming a value x to $ax + b$ where $a > 0$), the choice of move remains unchanged in a game tree, even when there are chance nodes.

5.19 Consider the following procedure for choosing moves in games with chance nodes:

- Generate some dice-roll sequences (say, 50) down to a suitable depth (say, 8).
- With known dice rolls, the game tree becomes deterministic. For each dice-roll sequence, solve the resulting deterministic game tree using alpha–beta.

- Use the results to estimate the value of each move and to choose the best.

Will this procedure work well? Why (or why not)?

5.20 In the following, a "max" tree consists only of max nodes, whereas an "expectimax" tree consists of a max node at the root with alternating layers of chance and max nodes. At chance nodes, all outcome probabilities are nonzero. The goal is to *find the value of the root* with a bounded-depth search. For each of (a)–(f), either give an example or explain why this is impossible.

a. Assuming that leaf values are finite but unbounded, is pruning (as in alpha–beta) ever possible in a max tree?

b. Is pruning ever possible in an expectimax tree under the same conditions?

c. If leaf values are all nonnegative, is pruning ever possible in a max tree? Give an example, or explain why not.

d. If leaf values are all nonnegative, is pruning ever possible in an expectimax tree? Give an example, or explain why not.

e. If leaf values are all in the range $[0, 1]$, is pruning ever possible in a max tree? Give an example, or explain why not.

f. If leaf values are all in the range $[0, 1]$, is pruning ever possible in an expectimax tree?

g. Consider the outcomes of a chance node in an expectimax tree. Which of the following evaluation orders is most likely to yield pruning opportunities?

(i) Lowest probability first
(ii) Highest probability first
(iii) Doesn't make any difference

5.21 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.

5.22 Consider carefully the interplay of chance events and partial information in each of the games in Exercise 5.4.

a. For which is the standard expectiminimax model appropriate? Implement the algorithm and run it in your game-playing agent, with appropriate modifications to the game-playing environment.

b. For which would the scheme described in Exercise 5.19 be appropriate?

c. Discuss how you might deal with the fact that in some of the games, the players do not have the same knowledge of the current state.