been worked on for so long, the standard approach dominates other methods in tournament play. Some believe that this has caused game playing to become divorced from the mainstream of AI research: the standard approach no longer provides much room for new insight into general questions of decision making. In this section, we look at the alternatives.

First, let us consider heuristic minimax. It selects an optimal move in a given search tree provided that the leaf node evaluations are exactly correct. In reality, evaluations are usually crude estimates of the value of a position and can be considered to have large errors associated with them. Figure 5.14 shows a two-ply game tree for which minimax suggests taking the right-hand branch because $100 > 99$. That is the correct move if the evaluations are all correct. But of course the evaluation function is only approximate. Suppose that the evaluation of each node has an error that is independent of other nodes and is randomly distributed with mean zero and standard deviation of $\alpha$. Then when $\alpha = 5$, the left-hand branch is actually better 71% of the time, and 58% of the time when $\alpha = 2$. The intuition behind this is that the right-hand branch has four nodes that are close to 99; if an error in the evaluation of any one of the four makes the right-hand branch slip below 99, then the left-hand branch is better.

In reality, circumstances are actually worse than this because the error in the evaluation function is not independent. If we get one node wrong, the chances are high that nearby nodes in the tree will also be wrong. The fact that the node labeled 99 has siblings labeled 1000 suggests that in fact it might have a higher true value. We can use an evaluation function that returns a probability distribution over possible values, but it is difficult to combine these distributions properly, because we won’t have a good model of the very strong dependencies that exist between the values of sibling nodes.

Next, we consider the search algorithm that generates the tree. The aim of an algorithm designer is to specify a computation that runs quickly and yields a good move. The alpha-beta algorithm is designed not just to select a good move but also to calculate bounds on the values of all the legal moves. To see why this extra information is unnecessary, consider a position in which there is only one legal move. Alpha-beta search still will generate and evaluate a large search tree, telling us that the only move is the best move and assigning it a value. But since we have to make the move anyway, knowing the move’s value is useless. Similarly, if there is one obviously good move and several moves that are legal but lead to a quick loss, we
Section 5.9. Summary

We have looked at a variety of games to understand what optimal play means and to understand how to play well in practice. The most important ideas are as follows:

- A game can be defined by the initial state (how the board is set up), the legal actions in each state, the result of each action, a terminal test (which says when the game is over), and a utility function that applies to terminal states.
- In two-player zero-sum games with perfect information, the minimax algorithm can select optimal moves by a depth-first enumeration of the game tree.
- The alpha—beta search algorithm computes the same optimal move as minimax, but achieves much greater efficiency by eliminating subtrees that are provably irrelevant.
- Usually, it is not feasible to consider the whole game tree (even with alpha—beta), so we would not want alpha—beta to waste time determining a precise value for the lone good move. Better to just make the move quickly and save the time for later. This leads to the idea of the utility of a node expansion. A good search algorithm should select node expansions of high utility—that is, ones that are likely to lead to the discovery of a significantly better move. If there are no node expansions whose utility is higher than their cost (in terms of time), then the algorithm should stop searching and make a move. Notice that this works not only for clear-favorite situations but also for the case of symmetrical moves, for which no amount of search will show that one move is better than another.

This kind of reasoning about what computations to do is called metareasoning (reasoning about reasoning). It applies not just to game playing but to any kind of reasoning at all. All computations are done in the service of trying to reach better decisions, all have costs, and all have some likelihood of resulting in a certain improvement in decision quality. Alpha—beta incorporates the simplest kind of metareasoning, namely, a theorem to the effect that certain branches of the tree can be ignored without loss. It is possible to do much better. In Chapter 16, we see how these ideas can be made precise and implementable.

Finally, let us reexamine the nature of search itself. Algorithms for heuristic search and for game playing generate sequences of concrete states, starting from the initial state and then applying an evaluation function. Clearly, this is not how humans play games. In chess, one often has a particular goal in mind—for example, trapping the opponent's queen—and can use this goal to selectively generate plausible plans for achieving it. This kind of goal-directed reasoning or planning sometimes eliminates combinatorial search altogether. David Wilkins' (1980) PARADISE is the only program to have used goal-directed reasoning successfully in chess: it was capable of solving some chess problems requiring an 18-move combination. As yet there is no good understanding of how to combine the two kinds of algorithms into a robust and efficient system, although Bridge Baron might be a step in the right direction A fully integrated system would be a significant achievement not just for game-playing research but also for AI research in general, because it would be a good basis for a general intelligent agent.