One might well wonder about the phrase "chances of winning." After all, chess is not a game of chance: we know the current state with certainty, and no dice are involved. But if the search must be cut off at nonterminal states, then the algorithm will necessarily be uncertain about the final outcomes of those states. This type of uncertainty is induced by computational, rather than informational, limitations. Given the limited amount of computation that the evaluation function is allowed to do for a given state, the best it can do is make a guess about the final outcome.

Let us make this idea more concrete. Most evaluation functions work by calculating various features of the state—for example, in chess, we would have features for the number of white pawns, black pawns, white queens, black queens, and so on. The features, taken together, define various categories or equivalence classes of states: the states in each category have the same values for all the features. For example, one category contains all two-pawn vs. one-pawn endgames. Any given category, generally speaking, will contain some states that lead to wins, some that lead to draws, and some that lead to losses. The evaluation function cannot know which states are which, but it can return a single value that reflects the proportion of states with each outcome. For example suppose our experience suggests that 72% of the states encountered in the two-pawns vs. one-pawn category lead to a win (utility +1); 20% to a loss (0), and 8% to a draw (1/2). Then a reasonable evaluation for states in the category is the expected value:

\[
E_{\text{VAL}} = (0.72 \times +1) + (0.20 \times 0) + (0.08 \times 1/2) = 0.76
\]

In principle, the expected value can be determined for each category, resulting in an evaluation function that works for any state. As with terminal states, the evaluation function need not return actual expected values as long as the ordering of the states is the same.

In practice, this kind of analysis requires too many categories and hence too much experience to estimate all the probabilities of winning. Instead, most evaluation functions compute separate numerical contributions from each feature and then combine them to find the total value. For example, introductory chess books give an approximate material value for each piece: each pawn is worth 1, a knight or bishop is worth 3, a rook 5, and the queen 9. Other features such as "good pawn structure" and "king safety" might be worth half a pawn, say. These feature values are then simply added up to obtain the evaluation of the position.

A secure advantage equivalent to a pawn gives a substantial likelihood of winning, and a secure advantage equivalent to three pawns should give almost certain victory, as illustrated in Figure 5.8(a). Mathematically, this kind of evaluation function is called a weighted linear function because it can be expressed as

\[
E_{\text{VAL}} = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)
\]

where each \(w_i\) is a weight and each \(f_i\) is a feature of the position. For chess, the \(f_i\) could be the numbers of each kind of piece on the board. and the \(w_i\) could be the values of the pieces (1 for pawn, 3 for bishop, etc.).

Adding up the values of features seems like a reasonable thing to do, but in fact it involves a strong assumption: that the contribution of each feature is independent of the values of the other features. For example, assigning the value 3 to a bishop ignores the fact that bishops are more powerful in the endgame, when they have a lot of space to maneuver.
Figure 5.8 Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

For this reason, current programs for chess and other games also use *nonlinear* combinations of features. For example, a pair of bishops might be worth slightly more than twice the value of a single bishop, and a bishop is worth more in the endgame (that is, when the *move number* feature is high or the *number of remaining pieces* feature is low).

The astute reader will have noticed that the features and weights are not part of the rules of chess! They come from centuries of human chess-playing experience. In games where this kind of experience is not available, the weights of the evaluation function can be estimated by the machine learning techniques of Chapter 18. Reassuringly, applying these techniques to chess has confirmed that a bishop is indeed worth about three pawns.

5.9.2 Cutting off search

The next step is to modify Alpha-Beta-Search so that it will call the heuristic EVAL function when it is appropriate to cut off the search. We replace the two lines in Figure 5.7 that mention TERMINAL-TEST with the following line:

```
if CUTOFF-TEST(state, depth) then return EVAL(state)
```

We also must arrange for some bookkeeping so that the current depth is incremented on each recursive call. The most straightforward approach to controlling the amount of search is to set a fixed depth limit so that CUTOFF-TEST(state, depth) returns true for all depth greater than some fixed depth d. (It must also return true for all terminal states, just as TERMINAL-TEST did.) The depth d is chosen so that a move is selected within the allocated time. A more robust approach is to apply iterative deepening. (See Chapter 3.) When time runs out, the program returns the move selected by the deepest completed search. As a bonus, iterative deepening also helps with move ordering.