

playouts. For example: consider a game with a branching factor of 32, where the average game lasts 100 ply. If we have enough computing power to consider a billion game states before we have to make a move, then minimax can search 6 ply deep, alpha–beta with perfect move ordering can search 12 ply, and Monte Carlo search can do 10 million playouts. Which approach will be better? That depends on the accuracy of the heuristic function versus the selection and playout policies.

The conventional wisdom has been that Monte Carlo search has an advantage over alpha–beta for games like Go where the branching factor is very high (and thus alpha–beta can’t search deep enough), or when it is difficult to define a good evaluation function. What alpha–beta does is choose the path to a node that has the highest achievable evaluation function score, given that the opponent will be trying to minimize the score. Thus, if the evaluation function is inaccurate, alpha–beta will be inaccurate. A miscalculation on a single node can lead alpha–beta to erroneously choose (or avoid) a path to that node. But Monte Carlo search relies on the aggregate of many playouts, and thus is not as vulnerable to a single error. It is possible to combine MCTS and evaluation functions by doing a playout for a certain number of moves, but then truncating the playout and applying an evaluation function.

Early playout termination

It is also possible to combine aspects of alpha–beta and Monte Carlo search. For example, in games that can last many moves, we may want to use **early playout termination**, in which we stop a playout that is taking too many moves, and either evaluate it with a heuristic evaluation function or just declare it a draw.

Monte Carlo search can be applied to brand-new games, in which there is no body of experience to draw upon to define an evaluation function. As long as we know the rules of the game, Monte Carlo search does not need any additional information. The selection and playout policies can make good use of hand-crafted expert knowledge when it is available, but good policies can be learned using neural networks trained by self-play alone.

Monte Carlo search has a disadvantage when it is likely that a single move can change the course of the game, because the stochastic nature of Monte Carlo search means it might fail to consider that move. In other words, Type B pruning in Monte Carlo search means that a vital line of play might not be explored at all. Monte Carlo search also has a disadvantage when there are game states that are “obviously” a win for one side or the other (according to human knowledge and to an evaluation function), but where it will still take many moves in a playout to verify the winner. It was long held that alpha–beta search was better suited for games like chess with low branching factor and good evaluation functions, but recently Monte Carlo approaches have demonstrated success in chess and other games.

The general idea of simulating moves into the future, observing the outcome, and using the outcome to determine which moves are good ones is one kind of **reinforcement learning**, which is covered in Chapter 22.

5.5 Stochastic Games

Stochastic game

Stochastic games bring us a little closer to the unpredictability of real life by including a random element, such as the throwing of dice. Backgammon is a typical stochastic game that combines luck and skill. In the backgammon position of Figure 5.12, Black has rolled a 6–5 and has four possible moves (each of which moves one piece forward (clockwise) 5 positions, and one piece forward 6 positions).