**Search Trees**

The problem of chess has been tackled extremely often in A.I., and the earliest work dates back from the period when classical A.I. used to be more popular (back in the 40s and 50s). Since everything was done symbolically then, the approach to computer chess was symbolic also: basically, each move is modelled as a branch in a tree. Alternate levels in the tree indicate alternate players' moves, and as they make choices the game progresses down the tree. For chess, each node in the tree represents a specific board configuration.

This tree is a very small part of the huge *search space* we discussed in the previous page. It's called a *search tree*. Basically, the computer needs to look through that to find the best possible option. But how does it know the best branch to take?

**Evaluation**

The search would be perfect if the computer could scan the entire tree: you go all the way to the end of the game, and find out what's going to happen in every case. Then you work backwards, and pick the solution which increases its chances of winning the most. That's all very nice, but in chess there's no way of getting to the end of the tree. And that's where evaluation comes in.

Instead of going all the way to the leafs of the tree, we stop at a specific node, and try to estimate how good this situation is. This is usually enough to indicate if the branch should be chosen or not, but all depends on the quality of the evaluation function. Typically, an evaluation function is hand-crafted by an expert. All the experience and knowledge is crammed into a small equation that tells the search how good a node is.

So how is this evaluation used in practice by the search?

**Minimax**

A minimax search algorithm can do this. The evaluation is used at each node in the tree to obtain a reward value for each player (although one value can be enough in zero sum games: a win on one side implies a loss on the other). The *minimax* algorithm alternatively tries to maximise the advantage of one player, and minimise the reward of the other player at consecutive levels of the tree. You can obvisouly do this the other way around if you want the computer to play as the other player, or if you want it to loose :P

**Enhancements**

Without going into too much detail, there are many improvements that can be made to such tree searches. One of them involves clipping branches that are obviously not going to be any better than solutions already found. This is known as alpha-beta pruning, and some of the best computer chess players use this approach.

Additionally to this, it is often beneficial to look into specific branches of the tree before others. This is done by using a heuristic function to guide the search. This allows other worse branches to be discarded more often, as the best will have been visited first. Also, among other things, this implies that the solution found for a specific number of evaluations is more optimal: if you stopped a heuristic search after a specific amount of time, the results will be better than without a heuristic.

**Problems!**

As we've shown, you can't search everything. The search needs to be stopped at a specific depth in the tree, this is called the **search ply**. This implies that the computer has a smaller view of the game. This limited scope has many obvious pitfalls... the computer can commit to an branch simply because the nasty consequences of its actions are beyond its horizon!

Controversially, recent research shows that in many cases, deeper ply searches can in fact be detrimental to the quality of the search, for the exact same reason. The key is then to come up with a good evaluation function. Obviously, there's room for machine learning here.

**Learning**

A lot of progress has been made. In other games such as Back-Gammon, some implementations have shown it possible to learn the best set of features to use in the evaluation function, and how to weight them. This is done by extensive self-play, which has allowed the program to beat even the best human players! It has even managed to introduce a new strategy to the game, which has revealed itself very successful.

Machine learning has not been as successful at learning to evaluate chess positions. Many argue this is due to back-gammon being more stochastic due to the roll of the die. Additionally, the board configuration in chess are more complex, and require more 'expert features' than back-gammon (an expert feature is a compound feature created by an expert combining complex properties of the board layout).

**Alternatives**

There is a lot of debate whether this symbolic approach is the *Right Way* do 'solve' chess... The symbolic approach is even doubted generally, and not only for chess! I recommend you read a discussion by a famous researcher called Rodney Brooks: [Elephants Can't Play Chess](http://www.ai.mit.edu/people/brooks/papers/elephants.pdf). He argues that a less symbolic approach is the way forward (not only in chess, but A.I. in general). At the other end of the scale, you get those die-hard minimax fans that claim that the entire field of A.I. can be solved with this kind of search... yes, but how exactly can you solve "AI"?

Meanwhile, back in the real world. More flexible approaches to minimax involving learning the evaluation function, as well as heuristics to guide the search. But nothing drastically new to chess has been used successfully. Neural Network based Othello players are spreading (not those using NN based evaluation functions), and can do fairly well. They still do not match the symbolic approach