# Artificial Intelligence in Chess: A Survey

**Shawn Sparks**
University of Northern Iowa
sparks13@uni.edu

**Abstract**

This work is to explain some of the big ideas in the field of artificial intelligence as it relates to chess. It starts with a brief history of the development over the centuries starting with the Automaton in the 18th century and following up until the famous match between Deep Blue and Gary Kasparov. Then, the work moves into the techniques in use today. Among the techniques talked about are the minimax algorithm, alpha-beta pruning, parallel processing, iterative-deepening depth-first searching, opening move databases, endgame databases, and evaluation functions. Exact methods for implementing the methods is not discussed, but key concepts of each are presented in a simple manner.

## 1. Introduction

What makes chess such a popular artificial intelligence topic? The easiest answer is that chess is not a simple game. Games like tic-tac-toe are simple enough that even a smart kid can figure out the best moves to make at any given point in the game. For an adult, the game almost loses any fun because it is so easy to force the game to a draw every time. Chess is different in this sense because the pieces each has its own way of moving and the board has sixty-four spaces as opposed to nine. At first glance, that does not seem overwhelmingly important. So the search has a slightly higher branching factor and will likely have a greater depth. In actuality, the branching factor jumps tremendously. It increases so much that the basic search or minimax will no longer work completely on its own. Today's computers simply do not have the processing power to be able compute such a large game tree.

This necessity to move beyond basic minimax is what intrigues so many to study artificial intelligence in the game of chess. Suddenly, heuristics are needed to cut down on the branching factor. Techniques are developed to reduce the number of nodes examined. This means the computer has to really think about the state of the game rather than do a simple analysis what is the final result of the game given a specific set of moves.

The idea of machines playing chess has been thought about for centuries now. Still, the greatest accomplishment computer scientists cling to is Deep Blue's victory over World Chess Champion Gary Kasparov in 1996. An act which many are still trying to replicate on a regular basis.

## 2. History

Chess is believed to have originated in India in the sixth century as a game known as chaturanga. The game finally made its way to Europe approximately one thousand years ago, but the game's popularity would peak in the twelfth and fifteenth centuries. During this period of time, the game had evolved from its original form into the game of chess thought of today (Murray, 1913).

### 2.1. The Automaton

The first well-known notion of a machine that could play chess came around approximate two hundred years ago. In the city of Vienna in1770, Wolgang van Kempelen debuted an invention of his named the Automaton. Kempelen took the Automaton on tour across Europe ranging from Russia to France. It is believed that Benjamin Franklin even inspected the machine at a showing in Paris in 1783. Kempelen sold it to another mechanical genius, Johann N Maelzel, in 1805. Maelzel continued touring with the machine, and supposedly, Napoleon Bonaparte played against it in 1809. During a stay in Paris in the 1820's, the Automaton is said to have had a streak where it won 99% of its games. In 1825, Maelzel travelled to the United States due to financial problems. In the U.S., the Automaton was put on display in all the major cities. In 1837, the Automaton was sold for a mere $400 to pay off some debts of Maelzel's. The Automaton spent the rest of its time in museums until it was destroyed in a fire in Philadelphia on July 5, 1854 (Gilmore, 1997).

The Automaton was described by Gilmore (1997) as follows: John Dickson Carr, the detective story writer, describes the machine in one of his stories. ln appearance it consisted of a human figure sitting down at a board which stood on a wooden chest. This chest had three doors in front. The owner of the affair always opened these doors one at a time to ' prove ' [sic] that the chest was empty of anything save a little mechanism and, anyway, could contain nothing larger than a child. Sadly, the machine did turn out to be a hoax when it was discovered that the trick involved hiding a tiny person in the wooden chest that the chess board sat on. Initially, the man hiding in the lower compartments had lost his legs in a war, but used artificial legs outisde of the machine so nobody suspected him. The individual would change frequently over the years, and was not always a male (Gilmore, 1997).

### 2.2. Real Chess-Playing Machines

In the 1840's, Charles Babbage is said to have considered the notion of his Analytical Machine playing chess, but he never got the opportunity to develop the idea (Hyatt and Nelson, 1990). Then at the Paris World Fair of 1900, a Spanish inventor by the name of Torres y Quevado demonstrated an electro-mechanical device that was capable of playing King and Rook versus King endgames.

While no where near as complicated as a full chess game with sixteen pieces all with their own special movement rules, it was the beginning for the next century's worth of work on Artificial Intelligence in the domain of chess (Bramer, 1982).

It was not until the 1950's though when Alan Turing's program played what is considered the first true computer chess game. In 1957, Henry Simon was so bold as to make the prediction that by 1967 a computer would be the World Chess Champion. It turned out his prediction was wrong, and it would take another thirty years before any results even came close (Hyatt and Nelson, 1990).

### 2.3. Deep Blue vs. Gary Kasparov

On February 10, 1996, IBM's computer, Deep Blue, finally succeeded in beating a World Chess Champion, Gary Kasparov, in a single game of chess. Sadly, the rest of the six-game match resulted in Kasparov winning three of the games and drawing the other two to beat Deep Blue 4-2 in the end.

A rematch between the two was played on May 11, 1997. By this point in time, Deep Blue had obtained the unofficial nickname Deeper Blue after receiving a large number of upgrades in an effort to allow it to search the game tree even deeper, hence the nickname. The upgrades proved their worth when Deep Blue obtained a full match victory with a final score of 3.5-2.5 over Kasparov. This would mark the first time a World Chess Champion had been defeated by a machine under standard tournament rules.

The key to Deep Blue's success was a hardware centric philosophy. It used extensive parallel processing on hardware built for processing chess moves. At the time of the 1996 match, Deep Blue was capable of evaluating approximately 100,000 positions per second. By 1997, the power had been doubled, and Deep Blue was evaluating 200,000 moves per second. This allowed Deep Blue to search to a depth of six to twelve plies on average and sometimes as deep as forty plies. The computer had studied previous games of Gary Kasparov to adjust its play against him. When Kasparov requested information on past games of Deep Blue, IBM refused. Kasparov went on to study other computer-played chess games to study typical behavior. To make matters worse on IBM, they were accused of cheating through means of human intervention during the second game of the 1997 match which was a Deep Blue win. Kasparov thought Deep Blue's play expressed an outstanding level of intelligence and creativity. IBM also refused to give the log files to Kasparov after the match, and also turned down the request for a rematch. Deep Blue was officially retired (Hsu, Campbell, Hoane, 1995).

## 3. Techniques of Chess AI

There is a wide array of techniques used in a chess-playing agent. The base for most of these agents is the minimax algorithm. It is widely understood that the more plies, number of alternating player turns, an agent can evaluate, the better it will perform. Many agents use a variation of the minimax algorithm to truncate the game tree. Accompanying this truncation process is an evaluation function which calculates a minimax value based on the current state of the game rather than the final outcome. Many agents are also employing alpha-beta pruning in attempt to trim nodes out of the game tree. Another technique is iterative-deepening depth-first search which allows the minimax game tree be truncated as little as possible, but also to make the most of the time a chess player is given. To speed up evaluation of the game tree, parallel processing is incorporated. One other strategy used to cut down on the number of nodes needing looked at in the game tree is the concept known as move databases. All of these can be used together and often times are all seen in some form or another in a single agent.

### 3.1. Minimax

Minimax is the simple algorithm that involves using a tree to determine the best next possible move for a player in a two-player game like chess. It is based on the idea that each contestant will make the move that will result in the best result for them. Since the computer is always concerned about which move it will make, it considers itself to be Max while its opponent is Min. In chess, the easiest way to handle the minimax score is by awarding a win a score of one, a draw a score of zero, and a loss a score of negative one. The algorithm goes through the game tree in a depth-first manner until it reaches an checkmate for either player or a draw. The appropriate score is passed back up the tree to the endgame (leaf) node's parent. If this new result is better than the previous best result currently achievable by the parent node, it replaces the old one. Best is defined by whether the parent node is at a level in the tree representing the computer's turn or the opponent's turn.

This procedure is followed until a result has been passed all the way up to the root node of the tree. The agent has now determined the best result it can expect to achieve in the game. The downside is that the algorithm requires the agent to be able to evaluate every possible move from the current state of the game to all possible endings. In chess, this becomes a ridiculously high number of nodes that even with all the computers in the world would be a timely task to complete. Deep Blue could calculate 200,000 positions per second and still only got six to twelve plies deep on average. So how is it possible to incorporate minimax into a chess agent then? The agents cut the search for endgames short and simply evaluate the player's position in the game at a given point. Essentially, the game tree is truncated. However, since higher nodes in a tree rely on results passed up from lower nodes, there must be a way to determine which play is in the stronger position. To do this, computer scientists use evaluation functions (Coppin, 2004).

### 3.2. Evaluation Functions

The purpose of an evaluation function is to consider the state of the board in the middle of a game and determine which player has the upper-hand as well as by how much. This is where the concept of chess strength really comes into play and people might start feeling more

comfortable about relating the agent's methodology with that of intelligent thought. If you went to a bookstore and picked up a book on playing chess, the topics it covered would most likely be used in the evaluation function if anywhere.

Strength is most commonly evaluated in three stages: opening, middle, and endgame. In the opening phase of the game, one of the key aspects is how well developed the player's pieces are. This is a notion of putting each piece in a strong position rather than leaving them in the basic starting squares. Many times an opening move database will be consulted during much of this stage of the game.

Then comes the middle phase. Here is where the most controversy and differing of opinions comes. The two foci at this stage are pawn development and the setup of the other pieces. The goal is to advance pawns, but in a protected manner so that they might be used to convert to different pieces on the far side of the board. They can also be used to block the advancement of the opponent's pawns and pieces. The other pieces are used in a process of capturing pieces, baiting the opponent, or putting pressure on the king.

In the endgame phase, enough pieces have been removed from the board that it is time to start thinking about checkmates. This may be in checkmating the opponent, but it could also be about avoiding a checkmate and fighting for a draw. Obviously, the key focus is on how to attack or defend kings. It is also works frequently in conjunction with an endgame database (Berliner, Kopec, and Northam, 1990).

These are the base concepts of evaluation functions as well as artificial intelligence in chess as a whole. Further detail will have to be overlooked due to the vastness of the topic. Literally, entire books have been written about these ideas, and the strategies of chess at the root of it all has been debated among chess players since its beginning. Still, nothing is as good as simply evaluating moves as deep as possible into the game. One method for looking at more moves is to throw out the moves that are clearly not worth the time to evaluate. This easiest method for pulling this off is alpha-beta pruning.

### 3.3. Alpha-Beta Pruning

The key idea with alpha-beta pruning is that there is no need to look at moves that are known to be not worth taking. The trick is determining which moves are irrelevant so they can be skipped. This is accomplished by looking a couple moves down the tree. Say it is the computer's turn. This means the first ply of the game tree represents all the possible moves for it. Each of these moves will have a final expected strategic evaluation score passed up to it from the subsequent moves and the evaluation function. The next ply down would represent the opponent's possible moves. This would also be the second ply of the game tree. Say the agent determines that the best score it can expect to obtain by making the first move examined results in a zero minimax score. Then the agent begins to examine its second possible move. If the first move of the opponent evaluates to a -1 minimax score, the agent knows already that his second move will at best result in a -1 because its oppoent will choose the move resulting in the -1, or a lower value if there is one. Therefore, the agent knows right then the first move is better than the second move. In this case, there is no reason to waste computing time inspecting any of the other moves following the agent's second move.

So what is the importance of saving a little computing time here and there? The deeper in a game tree an agent can search, the more informed of a decision it can make. The more informed a decision is, the smarter, or better, move it will make. By reducing the number of nodes being examined, the processing time is reduced. Because tournament chess rules require a move to be chosen within a specified amount of time, the time saved is vital for the agent's success. This excess time can be used to search deeper into the game tree as it makes the most of the time it has (Coppen, 2004).

### 3.4. Iterative-Deepening Depth-First Search

The technique known as iterative-deepening depth-first search is a great way to make sure an agent really is making the most of its time saved by alpha-beta pruning. Due to the vast differences in branching factor of a game tree in chess caused by number of pieces still on the board and their placement along with alpha-beta pruning allowing us to ignore nodes in the tree, it is hard for an agent to know how many plies it will be able to search across the breadth of the tree while not running out of time. This search technique makes a very quick, shallow search of possible moves. It is not anything special, but at least the agent now has a move to take if it does not get the time to find a better move. Then, the agent searches the game tree again, but to a depth one ply deeper. It continues to repeat the search process, increasing the depth of the search with each successive iteration until the time has run out and the computer must make a move.

Initially, iterative-deepening depth-first search seems like a great waste of time because of how many moves are evaluated multiple times. However, in actuality, the search does not waste as much time as might be thought. Due to the nature of a chess tree, the deepest plies contain an exponentially greater number of moves compared to the first few plies. If an agent were to such to a set depth on each move, it might risk trying to search one ply too many, resulting in it searching one thousand nodes without returning a move to be taken in time. Compared to iterative-deepening searching where only nine hundred nodes including one hundred repeat nodes might be searched, but at least a move was chosen. A fixed-depth search might also waste a great deal of its time sitting idle due to making a quick search. Iterative-deepening depth-first searches are great for utilizing as much of the alloted time as possible. Parallel processing simply performs those searches faster (Abramson, 1989).

### 3.5. Parallel Processing

Parallel Processing is simply a matter of dividing the workload to multiple processors. It does not directly effect the big ideas behind all the techniques previously discussed. All it does is do the work involved in those techniques on different processors so that it is not held up or constrained by the speed of a single processor. One of

the driving developments in Artificial Intelligence in chess is the application of the latest hardware to improve performance as was seen in the case of Deep Blue. With such a focus on time, it is just natural that solutions are sought after in the form of hardware as much as in the software.

So how exactly should the work be split up? There are a couple different common ways of doing this. The first is that all of the initial possible moves are put into a queue. Then each processor is assigned one move. The processor is dedicated to searching through the entirety of this move's subtree. Then as the processor finishes and becomes free, it is assigned another move if there are still any on the queue. This process continues until all of the moves have been examined. Normally in this case, a processor is dedicated to handling the control of the other processors.

Another means of distributing the workload is to actually queue every node in the game tree. Then, each processor will pull a node from the queue, evaluate it, and move to the next node. Often times in this case there is only a processor needed to start the process rather than a dedicated processor to handle the others as they free up. This is because each processor pulls from the same queue as they become free. With processors grabbing new nodes so frequently, it saves a lot of time in this manner. However, there is one more technique frequently used to speed up searches. This technique is known as a move database (Beal, 1986).

### 3.6. Move Databases

Move databases are large databases with common board states and a set of the best possible moves from that state. The idea here being that certain situations occur in a large number of chess games. Rather than searching through the tree to a limited depth each time a specific situation is encountered, a move database lets the agent instantly know a handful of the best possible moves from that position.

There are two common types of move databases, opening move databases and endgame databases. Endgame databases are the easier and more straight forward to develop. With this type, the goal is to set a board with few enough pieces that it is physically possible for an extended search to calculate how to force the best possible outcome. These searches are done well in advance of an actual game being played. Then, the initial state of the board is stored in the database along with the best move to make from that situation. When a game is actually played, the agent simply needs to get the game to a point where the state of the board matches one of the states in the endgame database. It then proceeds to follow the directions in the database until the end of the game. To truly be useful, these databases need to be highly comprehensive for several different possible endgames. A database like this would start with a King-Rook vs. King endgame. Then it might add a King-Knight vs. King endgame. It would keeping changing and adding piece combinations. The more variations and possible game states in the database, the more useful it will be during game time.

Opening move databases prove to be more difficult in that they are way too early in the game for a comprehensive search to be done on the game tree. This prevents a search being used to determine the official outcome. If it were possible, chess would be solved and the whole game would be played by simply referencing a single move database until the final outcome of the game. This is not the case, however. Instead a move database needs to be filled with opening strategies that are commonly understood to be good ways to open a game. These strategies are determined by the experts in the game of chess, even if they are not fully agreed upon. Unfortunately, this makes the chess agent predictable because the best chess players will be able to recognize the strategy in use and attack its weaknesses. In an attempt to be less predictable, an agent will need to vary the tactics it is using to throw off its opponents. This means that it cannot even necessarily choose the strongest move every single time. Its opponents will get used to this and be prepared to take advantage of it. The complicated part in all of this is how does an agent know when to pick the best move, the second best move, or so on down the line. How this is determined varies from system to system and is too in-depth for the focus of this work (Bramer, 1982).

## 4. Conclusion

The origins of chess trace back over a millenium, but it was not until the 20th century chess-playing computers were created. In that time, computer scientists have developed several highly useful methods for creating better chess agents. Most start with the minimax algorithm before altering it for performance gains. Among these alterations are the use of evaluation functions, alpha-beta pruning, parallel processing, and move databases. The overall search pattern is also modified through the use of iterative-deepening depth-first searches. These modifications can really be added in to an agent's implementation as desired. They can even all be put into the same agent. Ultimately, they can be used to create a fairly competitive chess agent.

## References


Abramson, Bruce. Control Strategies for Two- Player Games. *ACM Computing Surveys*. Vol. 21 (1989): 137-161.

Beal, D. F. *Advances in Computer Chess 4*. Macmillan Pub Co: New York, NY, 1986.

Berliner, Hans, Danny Kopec, and Ed Northam. A Taxonomy of Concepts for Evaluationg Chess Strength. *Conference on High Performance Networking and Computing*. 1990: 336-343.

Bramer, M. A., Game-Playing Programs: Theory and Practice. *ACM SIGART Bulletin*. 1982: 62-168.

Coppin, Ben. *Artificial Intelligence Illuminated*. Jones and Bartlett Publishers: Sudbury, Massachusetts, 2004.

Gilmore, C. Chess Automaton. *Chess Graphics*. May 1997. Nov. 1, 2007. http://www.chessgraphics.net/ac.htm

Hsu, Feng-hsiung, Murray S. Campbell, and A. Joseph Hoane, Jr. Deep Blue System Overview. *International*

*Conference on Supercomputing*. 1995: 240-244.
Hyatt, Robert M., and Harry L. Nelson Chess and Supercomputers: Details About Optimizing Cray Blitz. *Conference on High Performance Networking and Computing*. 1990: 354-363.
Murray, H. J. R., *A History of Chess*, Oxford University Press: Oxford, 1913.