ELSEVIER

Artificial Intelligence 134 (2002) 57–83

Artificial Intelligence

www.elsevier.com/locate/artint

Deep Blue

Murray Campbell [a,*], A. Joseph Hoane Jr. [b], Feng-hsiung Hsu [c]

[a] *IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA*
[b] *Sandbridge Technologies, 1 N. Lexington Avenue, White Plains, NY 10601, USA*
[c] *Compaq Computer Corporation, Western Research Laboratory, 250 University Avenue, Palo Alto, CA 94301, USA*

Abstract

Deep Blue is the chess machine that defeated then-reigning World Chess Champion Garry Kasparov in a six-game match in 1997. There were a number of factors that contributed to this success, including:

- a single-chip chess search engine,
- a massively parallel system with multiple levels of parallelism,
- a strong emphasis on search extensions,
- a complex evaluation function, and
- effective use of a Grandmaster game database.

This paper describes the Deep Blue system, and gives some of the rationale that went into the design decisions behind Deep Blue.

Keywords: Computer chess; Game tree search; Parallel search; Selective search; Search extensions; Evaluation function

1. Introduction

This paper describes the Deep Blue® computer chess system, developed at IBM® Research during the mid-1990s. Deep Blue is the culmination of a multi-year effort to build a world-class chess machine. There was a series of machines that led up to Deep Blue, which we describe below. In fact there are two distinct versions of Deep Blue, one which lost to Garry Kasparov in 1996 and the one which defeated him in 1997. This paper will refer primarily to the 1997 version, and comparisons with the 1996 version, which we

* Corresponding author. The ordering of the authors is alphabetic.
E-mail addresses: mcam@us.ibm.com (M. Campbell), Joe.Hoane@SandbridgeTech.com (A.J. Hoane Jr.), Feng-hsiung.Hsu@compaq.com (F.-h. Hsu).

0004-3702/01/$ – see front matter
PII: S0004-3702(01)00129-1

will call Deep Blue I, will be made where appropriate. For clarity, we will sometimes refer to the 1997 version as Deep Blue II. A brief summary of the chess machines described here can be found in Appendix B. A fuller history of the Deep Blue project can be found in [15].

1.1. ChipTest and Deep Thought

Earlier efforts in building a chess machine, ChipTest and Deep Thought, took place at Carnegie Mellon University in the 1980s. In 1988 Deep Thought was the first chess machine to beat a Grandmaster in tournament play. These systems used a single-chip chess move generator [12] to achieve search speeds in the neighborhood of 500,000 positions per second (ChipTest) to 700,000 positions per second (Deep Thought). Deep Thought is described in detail in [16,17].

1.2. Deep Thought 2

In 1989–90, part of the Deep Thought team (Anantharaman, Campbell, Hsu) moved to the IBM T.J. Watson Research Center to continue the effort to build a world-class chess machine. In late 1990, Joe Hoane replaced Thomas Anantharaman in the group. Deep Thought 2, aka Deep Blue prototype, was the first result of this effort. Although the primary purpose of the system was as an intermediate stepping stone to Deep Blue, Deep Thought 2 played in a number of public events from 1991 through 1995.

Although Deep Thought 2 used the same move-generator chip as Deep Thought, it had four improvements:

1. *Medium-scale multiprocessing*. Deep Thought 2 initially had 24 chess engines, although over time that number decreased as processors failed and were not replaced. This compares with Deep Thought, which usually used two processors (although there were four- and six-processor versions that made a few appearances).
2. *Enhanced evaluation hardware*. The Deep Thought 2 evaluation hardware used larger RAMs and was able to include a few additional features in the evaluation function. Nonetheless, the evaluation function was relatively simple. For example, the Deep Thought 2 hardware was not able to recognize “bishops of opposite color”, a feature that chess players know greatly increases the chance for a drawn endgame. In order to address this (and other similar problems), the Deep Thought 2 system implemented a software “band-aid” mechanism. Unfortunately this slowed down the overall system speed and created numerous search anomalies at the hardware/software boundary. In spite of these drawbacks, the missing evaluation function features were sufficiently important to use this evaluation patch.
3. *Improved search software*. The search software was rewritten entirely for Deep Thought 2, and was designed to deal better with the parallel search, as well as implement a number of new search extension ideas. This code would later form the initial basis for the Deep Blue search software.
4. *Extended book* [6]. The extended book (see Section 8.2) allowed Deep Thought 2 to make reasonable opening moves even in the absence of an opening book. This feature was also inherited by Deep Blue.

The major competitive successes of Deep Thought 2 included victories in the 1991 and 1994 ACM Computer Chess Championships, and a 3–1 win against the Danish national team in 1993.

1.3. Deep Blue I

Deep Blue I was based on a single-chip chess search engine, designed over a period of three years. The first chips were received in September of 1995. A number of problems were found with these chips, and a revised version was received in January of 1996.

Deep Blue I ran on a 36-node IBM RS/6000® SP™ computer, and used 216 chess chips. The chips each searched about 1.6–2 million chess positions per second. Overall search speed was 50–100 million chess positions per second.

The full 36-node Deep Blue I played only six games under tournament conditions, all in the February 1996 match with Garry Kasparov. This match was won by Kasparov by a fairly decisive 4–2 score, although the match was tied at 2–2 after the first four games. Three additional tournament-condition matches were played in preparation for the 1996 Kasparov match, all using a single-node version of Deep Blue with 24 chess chips. This system, aka Deep Blue Jr., beat Grandmaster Ilya Gurevich 1.5–0.5, drew Grandmaster Patrick Wolff 1–1, and lost to Grandmaster Joel Benjamin 0–2. [1]

1.4. Deep Blue II

After the 1996 match with Kasparov, it was clear that there were a number of deficiencies in the play of Deep Blue I. A series of changes were made in preparation for the rematch which took place in May of 1997. First, a new, significantly enhanced, chess chip was designed. The new chess chip had a completely redesigned evaluation function, going from around 6400 features to over 8000. A number of the new features were in response to specific problems observed in the 1996 Kasparov games, as well as in test games against Grandmaster Joel Benjamin. The new chip also added hardware repetition detection, a number of specialized move generation modes (e.g., generate all moves that attack the opponent's pieces: see Section 3.1), and some efficiency improvements that increased the per chip search speed to 2–2.5 million positions per second. The second major change was to more than double the number of chess chips in the system, and use the newer generation of SP computer to support the higher processing demands thereby created. A third change was the development of a set of software tools to aid in debugging and match preparation, e.g., evaluation tuning and visualization tools. Finally, we concluded that the searching ability of Deep Blue was acceptable, and we spent the vast majority of our time between the two matches designing, testing, and tuning the new evaluation function.

Deep Blue defeated Garry Kasparov in the 1997 match by a score of 3.5–2.5. For this victory, the Deep Blue team was awarded the Fredkin prize for defeating the human world champion in a regulation match. There were two additional matches played by Deep Blue Jr. in preparation for the Kasparov match. The two matches, against Grandmasters

[1] This last match went a long way to convincing us that Joel Benjamin would be an excellent Grandmaster consultant to the Deep Blue team.

Larry Christiansen and Michael Rohde, were both won by Deep Blue Jr. by a score of 1.5–0.5.

2. System overview

Deep Blue is a massively parallel system designed for carrying out chess game tree searches. The system is composed of a 30-node (30-processor) IBM RS/6000 SP computer and 480 single-chip chess search engines, with 16 chess chips per SP processor. The SP system consists of 28 nodes with 120 MHz P2SC processors, and 2 nodes with 135 MHz P2SC processors. The nodes communicate with each other via a high speed switch. All nodes have 1 GB of RAM, and 4 GB of disk. During the 1997 match with Kasparov, the system ran the AIX® 4.2 operating system. The chess chips in Deep Blue are each capable of searching 2 to 2.5 million chess positions per second, and communicate with their host node via a microchannel bus. The chess chips are described in Section 3.

Deep Blue is organized in three layers. One of the SP processors is designated as the master, and the remainder as workers. The master searches the top levels of the chess game tree, and then distributes "leaf" positions to the workers for further examination. The workers carry out a few levels of additional search, and then distribute their leaf positions to the chess chips, which search the last few levels of the tree.

Overall system speed varied widely, depending on the specific characteristics of the positions being searched. For tactical positions, where long forcing move sequences exist, Deep Blue would average about 100 million positions per second. For quieter positions, speeds close to 200 million positions per second were typical. In the course of the 1997 match with Kasparov, the overall average system speed observed in searches longer than one minute was 126 million positions per second. The maximum sustained speed observed in this match was 330 million positions per second.

Deep Blue relies on many of the ideas developed in earlier chess programs, including quiescence search, iterative deepening, transposition tables (all described in [24]), and NegaScout [23]. These ideas and others formed a very sound basis for designing and building a chess-playing system. Nonetheless, in creating a system as large and complex as Deep Blue, one naturally runs into relatively unexplored territory. Before describing the components of Deep Blue in detail (in Sections 3 through 8), it is worthwhile to discuss those characteristics of Deep Blue that gave rise to new or unusual challenges.

1. *Large searching capacity*. Previous research in game tree search typically dealt with systems that searched orders of magnitude fewer positions than Deep Blue. The best way to take advantage of this additional searching power is not clear. Our work on the Deep Blue search was guided by two main principles:
   (a) *The search should be* highly *non-uniform.* It is well known that strong human players are able to calculate well beyond the depth reachable by a uniform searcher of any conceivable speed. Our preference for a highly selective search arose from the loss of Deep Thought to Mike Valvo in a correspondence match [22], where it was clear to us that Valvo searched significantly deeper than Deep Thought. The fact that we were hoping to play Garry Kasparov, a chess player known for his complex attacking style, also figured into this choice.

(b) *The search should provide "insurance"* [2] *against simple errors.* We wanted to be sure that all move sequences were explored to some reasonable minimum depth. Early research into pruning algorithms (e.g., null move pruning [3,9]) did not provide us enough evidence to warrant implementation in the hardware search of Deep Thought 2 or Deep Blue. Even without pruning, and using highly selective search, [3] we felt that Deep Blue had sufficient searching power to satisfy our insurance needs. A three minute search on Deep Blue would reach a full-width depth of 12.2 on average. [4]

Sections 4 and 5 describe the Deep Blue search algorithm.

2. *Hardware evaluation.* The Deep Blue evaluation function is implemented in hardware. In a way, this simplifies the task of programming Deep Blue. In a software chess program, one must carefully consider adding new features, always keeping in mind that a "better" evaluation function may take too long to execute, slowing down the program to the extent that it plays more weakly. In Deep Blue, one does not need to constantly re-weigh the worth of a particular evaluation function feature versus its execution time: time to execute the evaluation function is a fixed constant. [5] On the other hand, it is not possible to add new features to the hardware evaluation, [6] and software patches are painful and problematic, as noted above about Deep Thought 2. For the most part, one must learn to either get by without a desired new feature, or manufacture some surrogate out of the features that are already available. Additionally, the extra complexity that is possible in the hardware evaluation function creates an "embarrassment of riches". There are so many features (8000) that tuning the relative values of the features becomes a difficult task. The evaluation function is described in Section 7.
3. *Hybrid software/hardware search.* The Deep Blue search combines a software search, implemented in compiled C code on a general purpose CPU, with a hardware search, encoded in silicon on the chess chip. The software search is extremely flexible, and can be changed as needed. The hardware search is parameterized, but the general *form* of the search is fixed. Thus it suffers to some degree from the difficulties similar to those associated with the hardware evaluation function: new search behaviors cannot be introduced, and the existing parameters require careful tuning. There is also an additional difficulty, namely choosing the best strategy for switching from the software to the hardware search. The very fact that the two searches are different (see Table 1) can lead to horizon effects [4].
4. *Massively parallel search.* Deep Blue is a massively parallel system, with over 500 processors available to participate in the game tree search. Although there has been

[2] This is the terminology used in [18].

[3] Our experiments showed that Deep Blue typically sacrificed two ply of full-width search in order to execute the selective search algorithms.

[4] This estimate is based on a linear least squares fit on all the (iteration, log(time)) data points from the 1997 match with Kasparov.

[5] Actually there is a distinction between slow and fast evaluation, and feature values can have an impact here. See Section 3.2.

[6] There were limits on design time, chip area, and manufacturing cost/time which made it difficult to build new chips.

Table 1
Comparison of hardware and software searches

Software search	Hardware search
Host processor, C code	Chess chip, state machines
Explores tree near root	Explores tree near leaves
No quiescence search	Complex quiescence search
Complex recursive extensions	Mostly local extensions
Transposition table	No transposition table
Uses hardware search as dynamic evaluation function	Uses on-chip static evaluation function
Flexible	Hardwired, limited configurability

previous research in such systems [8,13], integrating a large scale parallel search with the selective search mechanisms in Deep Blue created a new set of challenges. The parallel search and its interaction with the selective search is described in Section 6.

3. The chess chip

The chip used in Deep Blue is described in detail in [14]. This section will give a brief overview of the chip. Details of the functionality that is implemented will be described in the later sections on the hardware search (Section 5) and evaluation function (Section 7).

The chess chip divides into three parts: the move generator, the evaluation function, and the search control. We will examine each of these in turn, followed by a brief description of the on-chip support for external circuitry.

3.1. Move generation

The move generator in the Deep Blue chip was based[7] on the Deep Thought move generator chip [12,13,17], which was in turn based on the move generator of the Belle chess machine [7]. The Deep Blue chip has a number of additional functions, including the generation of checking and check evasion moves, as well as allowing the generation of certain kinds of attacking moves, which permits improved quiescence searching. The chip also supports several search extensions, including singular extensions [2].

The move generator is implemented as an 8×8 array of combinatorial logic, which is effectively a silicon chessboard. A hardwired finite state machine controls move generation. The move generator, although it generates only one move at a time, implicitly computes all the possible moves and selects one via an arbitration network. Computing all the moves simultaneously is one way to get minimum latency while generating moves in a reasonable order.

[7] The Deep Blue move generator is actually a superset of the Deep Thought move generator.

A reasonable move ordering, preferably as close to best-first as possible, is an important consideration for efficient search in game trees. The chess chip uses an ordering that has worked well in practice, first generating captures (ordered from low-valued pieces capturing high-valued pieces to high-valued capturing low-valued), followed by non-capture moves (ordered by centrality). After a move has been examined, a mechanism exists for masking it out and generating the next move in sequence.

3.2. Evaluation function

The evaluation function implemented in the Deep Blue chip is composed of a "fast evaluation" and a "slow evaluation" [7]. This is a standard technique to skip computing an expensive full evaluation when an approximation is good enough. The fast evaluation, which computes a score for a chess position in a single clock cycle, contains all the easily computed major evaluation terms with high values. The most significant part of the fast evaluation is the "piece placement" value, i.e., the sum of the basic piece values with square-based location adjustments. Positional features that can be computed quickly, such as "pawn can run", are also part of the fast evaluation. The slow evaluation scans the chess board one column at a time, computing values for chess concepts such as square control, pins, X-rays, king safety, pawn structure, passed pawns, ray control, outposts, pawn majority, rook on the 7th, blockade, restraint, color complex, trapped pieces, development, and so on. The features recognized in both the slow and fast evaluation functions have programmable weights, allowing their relative importance to be easily adjusted.

3.3. Search control

The search control portion of the chip uses a number of state machines to implement null-window alpha-beta search. The advantage of null-window search is that it eliminates the need for a value stack, simplifying the hardware design. [8] The disadvantage is that in some cases it is necessary to do multiple searches, for example when an exact score is needed.

Another limitation on the hardware search is the lack of a transposition table, which is known to improve search efficiency significantly in many cases. The effect of this limitation is lessened by the fact that the upper levels of the Deep Blue search are in software and have access to a transposition table.

The search requires a move stack to keep track of moves that have been explored so far at each level of the search tree. The move stack in Deep Blue II includes a repetition detector, which was not included in Deep Blue I. This detector contained a 32-entry circular buffer of the last 32 ply. Using a content-addressable memory algorithm [13], the repetition detector maintains the numbers of pieces displaced in each of the last 32 positions with respect to the current board position. When the number of pieces displaced equals zero, we have a repeated position. If the number of pieces displaced equals one, the hardware can recognize the presence of a legal move that would lead to repetition, and bound the score

[8] The alpha-beta algorithm normally maintains two values, alpha and beta, on a stack.

appropriately. A displaced count of one also can trigger the "no progress" condition: see Section 4.3.

3.4. Extendability

The chess chips optionally support the use of an external FPGA (Field Programmable Gate Array) to provide access to an external transposition table, more complicated search control, and additional terms for the evaluation function. In theory this mechanism would have allowed the hardware search to approach the efficiency and complexity of the software search. Null move search was also explicitly supported by this method. Due to time constraints, this capability was never used in Deep Blue.

4. Software search

Based on the experiences with Deep Thought 1, a new selective search was built for Deep Thought 2 (which would later form the basis for the Deep Blue selective search). This search, which we call "dual credit with delayed extensions" was designed based on a number of principles:

1. *Extend forcing/forced pairs of moves.* An important part of tactics in chess concerns forcing/forced pairs (ffp's) of moves.[9] These show up in various contexts, e.g.,
   (a) White has an unstoppable winning threat. Black has numerous delaying moves (e.g., checks, mate threats, attacks on high valued pieces, etc.) which demand precise responses by White. Eventually the delaying moves run out and the White win is discovered.
   (b) White has a series of sacrifices and immediate threats, which eventually result in checkmate or the win of material greater than that sacrificed.

   Ideally one would extend the search two ply for each ffp. This almost always leads to a "search explosion", i.e., an effectively non-terminating search. The following techniques are intended to address this problem.
2. *Forced moves are expectation dependent.* A move may be forced for one level of expectation and not forced for another. A move that is "fail low", i.e., below the current level of expectation, is never considered forced in the Deep Blue search. This restriction is also described in [2].
3. *Fractional extensions* [20]. It is not feasible to fully extend all the ffp's without the search exploding. First of all, there may be more than one reasonable response to a forcing move, though by the definition of forcing there should only be a small number of reasonable responses. Second, even forced moves usually have legal alternatives which must be refuted. One method of addressing this problem is to allow fractional extensions, where an ffp does not get a full 2-ply extension, but rather some smaller amount, say 1.75 ply. The less forcing the ffp, the less the extension.

[9] A move that is forced must have a backed up score significantly better (by more than some threshold) than the backed up score of all the available alternatives. This can be generalized to the case where there are a very small number of moves better than all the alternatives.

4. *Delayed extensions.* Often an isolated ffp is meaningless, and in any case it is not that hard to "search through". Things usually become interesting (and more difficult) when there is a series of ffp's. One response to this observation is to allow ffp's to accumulate "credit", and only when sufficient credit is available can it be "cashed in" for an extension. By setting this threshold appropriately, extensions are delayed until multiple ffp's occur in a given path.
5. *Dual credit.* An immediate and serious problem that arises in the above is on the "principal variation" (PV). The PV represents current best play for both sides, i.e., both sides are at their level of expectation. In this case, both sides may be forced to some degree, and both sides are not fail low, which allows credit to be accumulated. Clearly it is not feasible to accumulate more than 1 ply of credit for several PV moves in a row: the search will explode. One solution to this problem is to separate and accumulate the credit for the two sides separately.[10] If either side accumulates sufficient credit to cash in for an extension, the other side must give up an equal amount of credit.
6. *Preserve the search envelope.* As observed in [2], it is essential to preserve the search envelope to avoid an oscillating search.

Fig. 1 illustrates some of the basic ideas in the dual credit with delayed extensions algorithm. The pseudo-code is based on a depth-limited version of alpha-beta using the negamax formulation [19]. Lines added to the basic alpha-beta code are marked with a "*".

The first difference we note is in line 5, where the two credits are passed recursively as parameters in the call to DC. The next significant point is lines 14 through 19. Here is where an extension may take place. In line 14, CREDIT_LIMIT is the "cash-in" threshold. Line 15 calculates the number of plies of extension to be performed, which is the integer number of plies needed to bring hisCredit below CREDIT_LIMIT. Given a CREDIT_LIMIT of 2, as is used in Deep Blue, a value of hisCredit of 2.5 gives a 1-ply extension, and a hisCredit value of 3.25 gives a 2-ply extension. Note that as the extension is performed (line 18), both hisCredit and myCredit are reduced by the corresponding amount (though not below zero).

Until line 26, the code is similar to alpha-beta. Line 26 swaps hisCredit and myCredit as is required by the negamax framework, and recursively calls the search on the current successor position. Line 28 is reached if the current move has exceeded the previous bestScore. This is the prerequisite for credit to be given. GenerateCredit() hides a wealth of details, including full or reduced depth offset searches and null move threat tests. If credit is generated for the current move,[11] the search on the successor position is reinvoked (line 30) with the new credit added in. This time, if the score exceeds bestScore, it becomes the new bestScore (line 32).

To simplify this description, we have skipped over issues related to preserving the search envelope. In an actual implementation, it would be essential to know the amount of credit

[10] McAllester and Yuret [21] suggested separating the depth computation for the two sides, though not in the context of a credit system.

[11] Note that the generated credit can be fractional. Deep Blue has a granularity of 1/4 ply.

```
 1 int DC(
 2   position p,
 3   int alpha, int beta,
 4   int depthToGo,
 5*  float myCredit, float hisCredit)
 6 {
 7   int numOfSuccessors;
 8   int bestScore;
 9   int i;
10   int sc;
11*  float newCredit;
12*  int extensionAmount;
13
14*  if (hisCredit >= CREDIT_LIMIT) {
15*    extensionAmount = ceiling(hisCredit - CREDIT_LIMIT);
16*    hisCredit = hisCredit - extensionAmount;
17*    myCredit = max(myCredit - extensionAmount, 0);
18*    depthToGo = depthToGo + extensionAmount;
19*  }
20
21   if (depthToGo == 0) { return Evaluate(p); }
22
23   bestScore = alpha;
24   numOfSuccessors = GenerateSuccessors(p);
25   for (i = 1; i <= numOfSuccessors; i++) {
26*    sc = -DC(p.succ[i], -beta, -alpha, depthToGo - 1,
               hisCredit, myCredit);
27     if (sc > bestScore) {
28*      newCredit = GenerateCredit();
29*      if (newCredit > 0)
30*        sc = -DC(p.succ[i], -beta, -alpha, depthToGo - 1,
                    hisCredit, myCredit + newCredit);
31*      if (sc > bestScore)
32         bestScore = sc;
33     }
34     if (bestScore >= beta) { return bestScore; }
35   }
36   return bestScore;
37 }
```

Fig. 1. The dual credit algorithm.

generated in earlier appearances of this position, and search with at least that amount of credit. This avoids oscillating searches, as well as significantly reducing the number of re-searches (line 29). Details such as checkmate and stalemate have also been glossed over.

4.1. Credit generation mechanisms

There is a large set of mechanisms to identify nodes that should receive credit.

1. *Singular, binary, trinary, etc.* [12]
   A singular move is one that is significantly better than all the alternatives [2]. One can generalize this to the case where there are two, three or more good moves. Of course the more reasonable moves that are available, the less credit that should be given. It is clear that a large part of what a strong human chess player would define as forcing is covered by singularity. It is in just these kinds of positions that Grandmasters are able to calculate very deeply.
2. *Absolute singular.*
   When there is only one legal move a large credit can be given with very little risk. The reason is that, if the absolute singular move ends up failing low, there are no alternatives to examine so the cost is contained. It is reasonable in many cases to give a full two ply extension. This type of move is usually a check evasion move.
3. *Threat, mate threat.*
   It is relatively simple using a null move search to detect if there is a threat in the current position [1]. A null move search is a search conducted after one side passes. The intuition here is that if one side passes, then loses quickly, that side is deemed to have a pending threat against it which the other side is close to exploiting. Positions where a threat exists tend to be constrained in the number of reasonable alternatives. If a large threat exists, such as a threat to checkmate, a higher level of credit can be given. The Deep Blue implementation required that recent ancestors of a given position have forcing characteristics before a threat credit is given.
4. *Influence.*
   This mechanism gives credit for moves which are enabled by previous moves. For example, credit may be given for an apparently good white response to a black move which was not available the previous time black moved. The idea here is that we assume black is developing a combination even if we don't quite see what the combination is.
5. *Domain dependent.*
   Traditional search extension schemes, such as check evasion and passed pawn pushes, can also be incorporated into this method. For example, a passed pawn push can be considered a forcing move, and the response, if it does not fail low, can generate credit.

Many of these methods require auxiliary computation in order to gather the information necessary to make extension decisions. This was in line with our philosophy of using the tremendous raw searching power of Deep Blue to enable a more selective search.

The credit assigned for various conditions is depth dependent, with positions near the root of the tree generally receiving more credit than positions far from the root. This choice allowed quicker resolution of moderately deep forcing lines without allowing the search to explode.

[12] This terminology is borrowed from stellar astronomy, where it is used to count the number of stars in a system.

Table 2
Search characteristics, Position 1

Iteration	Minimum software depth	Maximum software depth	Estimated maximum combined depth
6	2	5	11–21
7	3	6	12–22
8	4	11	17–27
9	5	15	21–31
10	6	17	23–33
11	7	20	26–36
12	8	23	29–39

4.2. Sample behavior

The following gives a sample of the behavior of Deep Blue in two quite different positions. The first position [13] is before White's move 37 in the game Deep Blue–Garry Kasparov, Match game 2, New York, 1997, and contains some forcing tactical lines (see Table 2). The second position [14] is before Black's move 11 in the fifth game of the same match, and would not normally be considered a tactical position (see Table 3). For better observability, this experiment was run on Deep Blue Jr., a version of Deep Blue that runs on a single node of an IBM RS/6000 SP computer. For a given iteration i, the software is assigned $i - 4$ ply, which represents the minimum depth search in software. The maximum depth reached in software is greater than the minimum due to search extensions, and this value is given in the third column. In these two positions, the maximum software depth is approximately three times the minimum depth. The last column estimates the maximum depth reached in hardware and software combined. It is not possible to directly measure this number, but the estimate is based on results of simulating the hardware search. When hardware search extensions and quiescence search are taken into account, we typically see searches of 6 to 16 ply. Thus we can see iteration 12 searches can reach as deep as 40 ply in positions of this type, which suggests that position 2 is rather tactical after all. This shows that a superficial analysis of a position does not always assess the forcingness of the key lines of play.

4.3. Miscellaneous

The Deep Blue scores are composed of two 16-bit signed integers. The regular search score is in one integer, and the tie breaker score is in the other. Therefore, for a draw, the regular search score is zero and the tie breaker contains either the static evaluation of a theoretically drawn position or the count of moves until repetition, which is also useful choosing draws in the midgame. The count of moves to repetition will be positive if the

[13] r1r1q1k1/6p1/3b1p1p/1p1PpP2/1Pp5/2P4P/R1B2QP1/R5K1 w.
[14] r2qk2r/pp3ppp/2p1pn2/4n3/1b6/3P2PP/PPPN1PB1/R1BQK2R b.

Table 3
Search characteristics, Position 2

Iteration	Minimum software depth	Maximum software depth	Estimated maximum combined depth
6	2	5	11–21
7	3	6	12–22
8	4	10	16–26
9	5	16	22–32
10	6	19	25–35
11	7	20	26–36
12	8	24	30–40

machine is striving for a draw or the count will be negative if the machine is trying to avoid a draw.

Another idea in Deep Blue, implemented in both hardware and software, is a pruning mechanism we call "no progress". It is based on the assumption that if a move is good for a given side, it is best to play it earlier rather than later. "No progress" is implemented by detecting if the current position could have been reached by playing an alternate move at some earlier position on the search path. If so, the search is terminated with a fail low. Although this algorithm has only limited effect in most positions, situations which are somewhat blocked and have few pieces present can observe noticeable benefits.

5. Hardware search

The hardware search is that part of the Deep Blue search that takes place on the chess chip. A chess chip carries out a fixed-depth null-window search, which includes a quiescence search. There are also various types of search extension heuristics, both for the full-width and the quiescence portions of the search, which are described below.

The hardware search is fast, but is relatively simple in the Deep Blue system configuration. To strike a balance between the speed of the hardware search and the efficiency and complexity of the software search, we limit the chess chips to carry out only shallow searches. This typically results in 4- or 5-ply searches plus quiescence in middlegame positions and somewhat deeper searches in endgames.

Once a hardware search is initiated, the host processor controlling that chip is free to do other work, including performing the software search and initiating hardware searches on other chips. The host polls the chips to determine when a hardware search has completed. The host can abort a hardware search if needed, e.g., if the search is taking too long, or is no longer relevant.

The fact that the hardware search uses a null window requires special handling in the case where an exact value within a range is needed, rather than a bound. The host can carry out a binary search, initiating a series of null-window searches to determine the value. In many cases it is possible to use multiple chess chips simultaneously to speed this operation.

The main parameters of the hardware search are described below:

1. Depth of search, which controls the depth of the full-width portion of the hardware search. This is the primary parameter for controlling the size of search.
2. Depth of offset searches, to detect singular, binary, trinary conditions at the root node of the hardware search tree.
3. Endgame rules assertions off or on (always on in Deep Blue software code). The switch is mainly for debugging purposes.
4. Endgame ROM assertions off or on (always on in Deep Blue software code). The switch is mainly for debugging purposes. The endgame ROMs on the chess chip had the goal of improving the evaluation of common endgames where the natural evaluation was inaccurate. The particular endgames included were king and pawn vs. king, rook and pawn vs. rook, queen vs. pawn, and rook vs. pawn. Each endgame has certain characteristic patterns which are drawn, and some of these patterns are encoded in the ROMs.
5. Number of "mating" checks allowed for each side in the quiescence search. A mating check is a checking move which allows zero escape squares for the king or any checking move which is a "contact" [15] check by the queen. This parameter is used to control the size of the quiescence search.
6. Number of singular checking moves allowed in the quiescence search (king has one escape square, or queen or rook contact check, or any check given while the checked side has a hung [16] piece). This parameter is used to control the size of the quiescence search.
7. Flags to enable testing of singular, binary, or trinary conditions at the search root. These extensions are only implemented at the root of the hardware search. Without access to a transposition table, more general implementations could suffer from non-terminating searches.
8. Flag to ignore stalemate at one ply above quiescence.
9. Flag to allow a one-ply extension in the quiescence search after a pawn moves to the 7th rank or, in some cases, pawn moves to the 6th rank.
10. Flag to allow a one-ply extension in the quiescence search when the side to move has multiple hung pieces or a piece that is pinned and hung.
11. Flag to allow a one-ply extension in the quiescence search when the side to move has one hung piece for consecutive plies.
12. Flag to allow a one-ply extension in the quiescence search if opponent has any hung pieces.

6. Parallel search

Deep Blue is composed of a 30-node RS/6000 SP computer and 480 chess chips, with 16 chips per node. The SP nodes communicate with each other using the MPI (Message Passing Interface) standard [10]. Communication is via a high-speed switch. The chess

[15] A contact check is a checking move to a square immediately adjacent to the opposing king.

[16] A hung piece can be captured by the opponent with apparent safety.

chips communicate with their host node via a Micro Channel® bus. This heterogeneous architecture has a strong influence on the parallel search algorithm used in Deep Blue, as discussed below.

6.1. Parallel search algorithm

To characterize the parallel search algorithm used in Deep Blue, we will use the taxonomy given in [5].

1. *Processor hierarchy*. Deep Blue uses a *static* processor tree, with one SP node controlling the other 29 nodes, which in turn control 16 chess chips each. The static nature of the hierarchy is in part determined by the fact that the chess chips are not general purpose processors and can only act as slaves. In addition, the chess chips can only communicate directly with their host node.
2. *Control distribution*. Deep Blue uses a *centralized control* of the parallel search. The control is managed on the SP nodes, since the chess chips do not have this functionality.
3. *Parallelism possible*. Deep Blue permits parallelism under the following conditions: [17]
   (a) *Type* 1 (*PV*) *nodes*. After the first move has been examined at a PV node, all the alternatives may be examined in parallel (with an offset window to enable the selective search algorithm). Null move searches, used to generate threat information for the selective search, can also be carried out in parallel. Finally, PV nodes at the hardware/software boundary can be searched in parallel. Because the hardware search allows only null-window searches, a number of searches are required to determine an exact score within a window.
   (b) *Good type* 2 *nodes* (nodes where the first move "fails high", or exceeds expectations). Most parallel algorithms do not allow or need parallelism in this case. Deep Blue executes reduced depth offset searches as well as the null move searches in parallel after the first move fails high. As above, these searches generate information for use by the selective search.
   (c) *Bad type* 2 *nodes* (nodes where the fail high move is not searched first). The moves after the first are searched in parallel. After a fail high move is found, all the alternatives are searched in parallel with a reduced depth offset search. Null move searches also can execute in parallel at this point.
   (d) *Type* 3 *nodes* (nodes where all the moves fail low). These moves can all be searched in parallel.
4. *Synchronization*. Type 1 and type 2 nodes are synchronization points for Deep Blue. The first move must be evaluated before parallelism is allowed. There are also global synchronization points at the end of iterations.

[17] The terminology of node types used in this section was originally defined in [19].

6.2. Parallel search implementation

The early iterations of the Deep Blue parallel search are carried out on the master node. There is not much parallelism in the first few iterations, and the master is fast enough (it has 16 chess chips) that there is little to be gained by attempting to further parallelize the search.

As the search gets deeper, jobs get allocated throughout the system. There are three major issues that need to be addressed:

- *Load balancing*. The search extensions algorithm used in Deep Blue leads to widely varying tree sizes for a given search depth. This extends all the way to the hardware, where the complex quiescence search can cause a search to "blow up". This can lead to severe load balancing problems. The solution used in Deep Blue was to abort long-running hardware searches (more than 8000 nodes) and push more of the search into software. This gives additional opportunities for parallelism. Similarly, jobs on the worker nodes can abort and return their job to the master for further splitting.
- *Master overload*. The performance bottleneck of the Deep Blue system was the master processor. One method of alleviating this was to ensure that the workers always had a job "on deck", ready to execute when it completes its active job. This reduces the effect of the communication latency between master and workers.
- *Sharing between nodes*. Workers do not directly communicate with each other. This decision was made in order to simplify the implementation. Workers will generally pass their "expensive" transposition table results up to the master.

As a final point, it should be noted that the Deep Blue parallel search is non-deterministic. Various factors can influence timing and processor job assignments. Although this was not a major concern, it makes debugging the system much more difficult.

6.3. Parallel search performance

We have limited experimental results to assess the efficiency of the Deep Blue parallel search. The most accurate numbers were derived on a single-node version of Deep Blue with 24 chess chips. We compared the 24 chip system with a single chip system on a variety of positions. The results varied widely depending on the tactical complexity of the position searched. For positions with many deep forcing sequences speedups averaged about 7, for an observed efficiency of about 30%. For quieter positions, speedups averaged 18, for an observed efficiency of 75%. The non-deterministic nature of the search, particularly in tactical positions, makes it more difficult to conduct these measurements.

It is difficult to assess the efficiency of the full 30-node Deep Blue system. Indirect evidence suggests an overall observed efficiency of about 8% in tactical positions and about 12% in quiet positions. It is clear that there is room for improvement here. However it was a conscious design decision of the Deep Blue team to focus on improving the evaluation function following the 1996 Kasparov match, and the parallel search code was largely untouched between the 1996 and 1997 matches.

7. Evaluation function

7.1. Overview

The Deep Blue evaluation function is essentially a sum of feature values. The chess chip recognizes roughly 8000 different "patterns", and each is assigned a value. Features range from very simple, such as a particular piece on a particular square, to very complex, as will be described below in Section 7.2. A feature value can be either static or dynamic. Static values are set once at the beginning of a search. Dynamic values are also initialized at the beginning of a search, but during the search they are scaled, via table lookup, based on the value and type of pieces on the board at evaluation time. For example, king safety, passed pawns, and pawn structure defects are sensitive to the amount of material on the board.

The initialization of the feature values is done by the "evaluation function generator", a sub-program which was run on the master node of SP system. The Deep Blue evaluation function generator is run only at the root of the search tree. It would likely be of great benefit to run it at other nodes near the root of the tree after a large positional change has occurred, such as trading queens. Although the dynamic evaluation terms can handle this type of transition, some static feature values may be left with less than ideal values. Unfortunately the full evaluation function generator takes measurable wall clock time to run and download, and partial downloading was considered too complex to implement.

The evaluation function generator has a second role beyond simply adjusting feature values based on the context of the root position. The large number of distinct feature values dictate that some sort of abstraction be imposed on the values in order to keep the task manageable. The evaluation generator makes these abstractions, dictating relationships between groups of related feature values rather than setting them independently.

There are 54 registers (see Table 4) and 8096 table entries (see Table 5) for a total of 8150 parameters that can be set in the Deep Blue evaluation function. [18] Some of the parameters correspond to chess situations that are not physically realizable (e.g., pawns on the first rank), and others are used for control purposes rather than corresponding to a particular combination of chess features. There are about 8000 actual features that can be detected in the chess hardware.

It is impossible in a paper like this to describe all the details of the evaluation function. We will present a detailed example of one table that gives a feel for the nature of the evaluation function. A brief description of all the registers and tables is given in Appendix A.

7.2. Extended example: Rooks on files

We now describe in detail the "Rooks on files" table. We will begin by describing the features that are detected, then discuss the values assigned to each combination of features, and show how the values are incorporated into the overall position evaluation.

[18] In actual fact, the number is even higher than this. Many of the registers and table entries have two, three, or even four separate values. The extended example in Section 7.2 illustrates one method of using multiple values.

Table 4
Deep Blue evaluation registers

Function	Number of registers	Data bits
Rooks on seventh rank	12	8
Bias	1	8
Opposing rook behind passed	1	9
Mpin and hung	1	7
Pinned and simple hung	1	8
Hung	4	7
Xraying	2	6
Pinned and hung	1	7
Permanent pin and simple hung	1	8
Knight trap	6	8
Rook trap	8	8
Queen trap	2	8
Wasted pawns	2	6
Bishop pair	2	7
Separated passed	2	8
Missing wing	2	10
Bishops of opposite colors	2	6
Evaluation control	2	32
Side to move	2	4

The chessboard is scanned, one file at a time, and a pair of values is looked up, one from the white rook table and one from the black rook table, for each file. The index bits for these tables are as follows:

- "unopposed" is a 1-bit subindex, with 0 indicating an enemy pawn on the file, and 1 indicating no enemy pawn on the file.
- "blockage" is a 2-bit subindex with two interpretations, depending on whether or not there is an enemy pawn on the file. If there are no enemy pawns, 0 indicates that my rook could safely move to the 7th or 8th ranks, 1 indicates that there is a minor piece guarded by a pawn that blocks the file, 2 indicates that enemy minor pieces guard the 7th and 8th ranks, and 3 indicates that the 7th and 8th are guarded by the enemy, but not by minor pieces. If there is an enemy pawn on the file, 0 indicates that the pawn is unprotected, 1 indicates that there is a minor piece guarded by a pawn that shields the pawn, 2 indicates that the pawn is protected by a minor piece, and 3 indicates "granite", i.e., the pawn is protected by another pawn.
- "semi_open" is a 1-bit subindex, with 0 indicating the presence of a pawn of mine on the file, and 1 indicating no pawns of mine on the file.

Table 5
Deep Blue evaluation tables

Function	Number of tables	Table entries	Data bits
Multiple pawns	2	80	8
Minor on weak	2	192	12
Self block	2	320	5
Opponent block	2	128	4
Back block	2	160	5
Pinned	2	128	8
Mobility	8	128	9
Pawn structure	2	160	32
Passed pawns	2	320	32
Joint signature	1	256	8
Rooks on files	2	192	10
Bishops	4	128	11
Pawn storm	2	128	18
Pawn shelter	2	384	14
Development	1	256	9
Trapped bishop	1	128	8
Signature	2	128	20
Contempt	1	256	8
Piece placement	1	1024	10

- "rook_count" is a 2-bit subindex, with only the values 0, 1 and 2 being legal. rook_count indicates the number of rooks for my side that are not behind my own pawns.
- "centrality" is a 2-bit subindex, with files "a" and "h" receiving value 0, files "b" and "g" receiving 1, files "c" and "f" receiving 2, and files "d" and "e" receiving 3.

Each table entry is 10 bits, which is divided into three fields:

- "kmodOpp" is a 2-bit field which causes extra points to be added to the king safety if the sides have castled on opposite sites. The field chooses a multiplier, which is 2 for 0, 1.5 for 1, 1 for 2, and 0.5 for 3. The base value (see below) is multiplied by the appropriate value and then included in the king safety calculation if the file is adjacent to the enemy king. As a special case, the rook file is considered adjacent to the bishop file.
- "kmod" is a 2-bit field, similar to kmodOpp, used when the kings have castled on the same side.
- "base" is a 6-bit field which gets summed into the overall evaluation. This is the "value" of the given formation, independent of king safety considerations.

There is an additional factor to consider for rooks on files. Under some circumstances, pawns can be semi-transparent to rooks. For example, if a pawn is "levering", it is considered semi-transparent to rooks. For this purpose, levering is defined to be having the possibility of capturing an enemy pawn. Under such circumstances, rooks get about half the value of the unblocked file. This feature was of critical importance in Game 2 of the 1997 match between Garry Kasparov and Deep Blue.

The king-safety component of rooks on files is not directly added to the evaluation of a position, but is first scaled by the amount of material on the board (via a table lookup). Positions with most of the pieces still on the board may retain the full king-safety value, while endgames will have the value scaled to close to zero. This, for example, encourages Deep Blue to trade pieces in positions where its king is less safe than the opponent's king. The king safety term itself is non-linear, and is quite complex, particularly before castling has taken place.

7.3. Automated evaluation function analysis

Although the large majority of the features and weights in the Deep Blue evaluation function were created/tuned by hand, there were two instances where automated analysis tools aided in this process.

The first tool had the goal of identifying features in the Deep Blue I evaluation function that were "noisy", i.e., relatively insensitive to the particular weights chosen. The hypothesis was that noisy features may require additional context in order to be useful. A hill-climbing approach was used to explore selected features (or feature subsets), and those that did not converge were candidates for further hand examination. A number of features in the Deep Blue I evaluation were identified, and significantly modified in Deep Blue II hardware, including piece mobility, king safety, and rooks on files.

A second tool was developed with the goal of tuning evaluation function weights. This tool used a comparison training methodology [25] to analyze weights related to pawn shelter. Training results showed that the hand-tuned weights were systematically too low [26], and they were increased prior to the 1997 match. There is some evidence that this change led to improved play [26].

8. Miscellaneous

8.1. Opening book

The opening book in Deep Blue was created by hand, primarily by Grandmaster Joel Benjamin, with assistance from Grandmasters Nick De Firmian, John Fedorowicz, and Miguel Illescas. The book consisted of about 4000 positions,[19] and every position had been checked by Deep Blue in overnight runs. The openings were chosen to emphasize positions that Deep Blue played well. In general this included tactically complex openings,

[19] This may seem surprisingly small. In fact, numerous openings did have minimal preparation, due to our confidence in the extended book (Section 8.2).

but also included more positional openings that Deep Blue handled well in practice. Opening preparation was most extensive in those openings expected to arise in match play against Kasparov. In fact, none of the Kasparov-specific preparation arose in the 1997 match.

Prior to a game, a particular repertoire was chosen for Deep Blue. There were a number of possible repertoires to choose from, and the choice would be made on the basis of the match situation and the previous experience playing with the same color. Last minute changes or corrections were made in a small "override" book.

8.2. Extended book

The extended book [6] in Deep Blue is a mechanism that allows a large Grandmaster game database to influence and direct Deep Blue's play in the absence of opening book information. The basic idea is to summarize the information available at each position of a 700,000 game database, and use the summary information to nudge Deep Blue in the consensus direction of chess opening theory.

The specific mechanism used in Deep Blue was to assign bonuses (or penalties) to those moves in a given position that had been played in the Grandmaster game database. For example, suppose that in the opening position of a chess game the move d4 is given a 10 point bonus. Deep Blue would carry out its regular search, but offset the alpha-beta search window for d4 by 10 points. Thus d4 would be preferred if it was no more than than 10 points worse the best of the other moves.

A number of factors go into the extended book evaluation function, including:

- The number of times a move has been played. A move frequently played by Grandmasters is likely to be good.
- Relative number of times a move has been played. If move A has been played many more times than move B, then A is likely to be better.
- Strength of the players that play the moves. A move played by Kasparov is more likely to be good than a move played by a low-ranked master.
- Recentness of the move. A recently played move is likely to be good, an effect that can in some cases dominate other factors.
- Results of the move. Successful moves are likely to be good.
- Commentary on the move. Chess games are frequently annotated, with the annotators marking strong moves (with "!") and weak moves (with "?"). Moves marked as strong are likely to be good; moves marked as weak are likely to be bad.
- Game moves versus commentary moves. Annotators of chess games frequently suggest alternative moves. In general, game moves are considered more reliable than commentary moves, and are thus likely to be better.

We developed an ad hoc function that combined these factors in a nonlinear way to produce a scalar value as output. The value of the bonus can be as high as half a pawn in favorable situations. In some situations, where the bonus for one move is very large and other move bonuses are much smaller, Deep Blue has the option of playing a move immediately, without first carrying out a search.

The extended book was introduced into Deep Thought 2 in 1991, and was used with good success through the matches with Kasparov. In [6], an example is given of how the extended book worked in game 2 of the 1997 Kasparov–Deep Blue match.

8.3. Endgame databases

The endgame databases in Deep Blue includes all chess positions with five or fewer pieces[20] on the board, as well as selected positions with six pieces that included a pair of blocked pawns. The primary sources for these databases were the Ken Thompson CD-ROMs [27] and additional databases supplied by Lewis Stiller.

Endgames databases were used both off-line and on-line. The off-line usage was during the design of the chess chips. Each chip had a ROM which stored patterns to help evaluate certain frequently occurring chess endgames. The databases were used to verify and evaluate these patterns. See Section 5 for more details.

The software search used the databases in on-line mode. Each of the 30 general purpose processors in the Deep Blue system replicated the 4-piece and important 5-piece databases on their local disk. The remaining databases, including those with 6 pieces, were duplicated on two 20-GB RAID disk arrays, and were available to all the general purpose processors through the SP switch.

Endgames were stored in the databases with one bit per position (indicating lose or does-not-lose). If a position is reached during the search that had a known value, it received a score composed of two parts: a high-order, game theoretic value, and a low-order, tie-breaker value. The tiebreaker value is simply the value produced by the evaluation function on the position in question. If this score is sufficient to cause a cutoff, the search immediately backs up this score.

For example, suppose Deep Blue had to choose between various possible continuations that resulted in it playing the weak side of a rook and pawn versus rook endgame. Deep Blue would, of course, prefer drawn positions over lost ones. In addition, given the choice between different drawn positions, it would choose the one with the best evaluation function value.

The endgame databases did not play a critical role in the matches against Kasparov. In the 1997 match, only game 4 approached an endgame that required access to the databases, but the ROMs on the chess chips had sufficient knowledge to recognize the rook and pawn versus rook draws that could have arisen.

8.4. Time control

Chess games typically have a set of requirements on the speed of play, termed the "time control". For example, the Deep Blue–Kasparov games initially required 40 moves to be played in two hours. Failure to make the specified number of moves leads to forfeiting the game.

The time control mechanism in Deep Blue is relatively straightforward. Before each search, two time targets are set. The first is the normal time target, set to be approximately

[20] We use the term "piece" to also include pawns.

the time remaining to the next time control divided by the moves remaining. In practice, a considerable time buffer is reserved, which allows for sufficient time to handle technical difficulties, as well as saving time for a possible "sudden-death" phase. The second time target is the panic time target, which is roughly one third of the remaining time.

If, at the normal time target, the situation is "normal", a time-out occurs and the current best move is played. There are a few conditions under which Deep Blue will go beyond its normal target into "panic time".

- The current best move has dropped 15 points or more from the score of the previous iteration. In this case, the search continues until either a new move is found within the 15 point margin, the iteration is completed, or the panic time target is reached.
- The best move from the previous iteration is in a potential "fail-low" state. Continue until this state is resolved, or the panic time target is reached. If this move ends up dropping 15 points or more from its prior value, continue as in the previous case.
- A new move is in a potential "fail-high" state. Continue until this state is resolved, or the panic time target is reached.

These conditions are triggered frequently during a game, but it is quite rare to actually go all the way to the panic time target. In the 1997 match with Kasparov, this happened only once.

9. Conclusion

The success of Deep Blue in the 1997 match was not the result of any one factor. The large searching capability, non-uniform search, and complex evaluation function were all critical. However other factors also played a role, e.g., endgame databases, the extended book, and evaluation function tuning.

It is clear, however, that there were many areas where improvements could have been made. With additional effort, the parallel search efficiency could have been increased. The hardware search and evaluation could have been made more efficient and flexible with the addition of an external FPGA. Current research (e.g., [11]) suggests that the addition of pruning mechanisms to Deep Blue might have significantly improved the search. Evaluation function tuning, both automatic and manual, was far from complete.

In the course of the development of Deep Blue, there were many design decisions that had to be made. We made particular choices, but there were many alternatives that were left unexplored. We hope this paper encourages further exploration of this space.

Acknowledgements

We would like to gratefully acknowledge the help of the following people: C.J. Tan, Jerry Brody, Joel Benjamin, John Fedorowicz, Nick De Firmian, Miguel Illescas, Lewis Stiller, Don Maddox, Gerald Tesauro, Joefon Jann, Thomas Anantharaman, Andreas Nowatzyk, Peter Jansen, and Mike Browne. We would also like to thank the editor and the anonymous referees for their helpful comments.

Appendix A. Evaluation tables and registers

- Rooks on seventh: There are 12 8-bit registers, for the various combinations of {White, Black} × {single, doubled} × {regular, absolute-king-trapped, absolute-king-not-trapped}.
- Bias: This register was designed for use with external (off-chip) hardware.
- Rook behind passed: A bonus is awarded for a rook behind a passed pawn of the same color.
- Mpin and hung: A bonus is awarded when the opponent has a piece that is hung and pinned against a piece equal in value to the pinner.
- Pinned and simple hung: A bonus is awarded when the opponent has a piece that is pinned and hung.
- Hung: A bonus is awarded if the opponent has a piece that is hung.
- Xraying: A bonus is awarded for having an "xray attack", i.e., an attack masked by one's own piece.
- Pinned and hung: A bonus is awarded when the opponent has a piece that is both pinned and hung. "Hung" is distinguished from "simple hung" by a more detailed analysis of the capture sequence.
- Permanent pin and simple hung. A bonus is awarded for a permanent pin of a piece.
- Knight trap: These 6 registers (3 for each side) provide penalties for some frequently occurring situations where knights can get trapped.
- Rook trap: These 8 registers (4 for each side) provide penalties for some frequently occurring situations where rooks can get trapped.
- Queen trap: These 2 registers (1 for each side) provide penalties for when there is no safe queen mobility.
- Wasted pawns: A penalty is assessed for pawn groups that have "wasted" pawns. A pawn group has a wasted pawn if it is unable to create a passed pawn against a smaller group of enemy pawns. There are two values, one each for White and Black, and the penalties are dynamically scaled by the amount of material on the board at the time of evaluation.
- Bishop pair: A bonus may be awarded for having a pair of bishops. There are two values, on each for White and Black.
- Separated passed: Widely separated passed pawns are advantageous if trying to win in a bishops of opposite color endgame.
- Missing wing: Provides a penalty for the side that is ahead if it allows itself to be left with pawns on only one side.
- BOC: A set of reductions on the evaluation for pure bishops of opposite color, and various combinations of major pieces and bishops of opposite color.
- Side to move: A bonus may be given for being on move.
- Multiple pawns: There are two 80-entry tables, for the various combinations of {White, Black} × {backward, not backward} × {opposed, unopposed} × {doubled, tripled+} × {ah, bg, cf, de} × {isolated, pseudo-isolated, supported, duo}. Some combinations are not legal. There is an 8-bit penalty value associated with each realizable combination.

- Minor on weak: This pair of tables gives bonuses for minor pieces on weak squares, taking into account such factors as advancement, centrality, pawn support, minor piece type, challengability, and screened status.
- Self block: This table assesses how each side's bishops are restrained by their own pawn.
- Opp block: This table assesses how each sides's bishops are restrained by the opponents's pawns.
- Back block: This table assesses how doubled pawns can restrain a bishop's mobility.
- Pinned: Pins are awarded bonuses depending on the strength of the pin and the amount of pressure on the pinned piece.
- Mobility: The mobility to each square on the board is summed to produce on overall mobility score. Each square has 16 possible levels of mobility, from maximum Black control through maximum White control. The control level and the square location determine each square's contribution.
- Pawn structure: This table assesses various features of pawn structure not handled elsewhere.
- Passed pawns: This table assesses the worth of passed pawns.
- Joint signature: This table allows adjustments to be made for particular piece matchups.
- Rooks on files: There are two 192-entry tables, for the various combinations of {White, Black} $\times$ {opposed, unopposed} $\times$ {4 blockage types} $\times$ {semi-open, not semi-open} $\times$ {ah, bg, cf, de} $\times$ {0, 1, or 2 rooks}. There is an 6-bit value associated with each combination, and two 2-bit flags indicating how the value is to be used in the king-safety calculation.
- Bishops: Bishops are awarded bonuses for the value of the diagonals they control. The diagonals are individually assessed, and include factors such as transparency (ability to open at will), king safety, and target of attack.
- Pawn storm: This table helps to assess pawns being used to attack the opponent's king.
- Pawn shelter: This table evaluates the pawn shelter that protects or may protect one's own king.
- Development: This table measures the differences in development between the two sides, factors in the king situation, and gives a bonus to the side that is ahead in development/king safety.
- Trapped bishop: This table is used to detect situations where bishops can become trapped.
- Signature: These tables, one for each side, allow adjustments to be made for particular piece combinations that work well or poorly together. For example, queen and knight are thought to cooperate better than queen and bishop.
- Contempt: The table gives the adjustment to the draw score. It is used to either prefer or avoid draws, depending on the opponent and the match situation. The adjustment is material dependent.
- Piece placement: This table, in common use in most chess programs, has one entry for each piece type on each square of the board.

Table B.1
Summary of systems

System name	First played	Processors	Nodes/second	Feature groups
ChipTest	1986	1	50K	1
ChipTest-m	1987	1	400K	1
Deep Thought	1988	2–6	700K–2M	4
Deep Thought 2	1991	14–24	4M–7M	4
Deep Blue I	1996	216	50M–100M	32
Deep Blue II	1997	480	100M–200M	38
Deep Blue Jr.	1997	24	20M–30M	38
Deep Blue Jr. demo	1997	1	2M	38

Appendix B. Summary of Deep Blue and predecessors

Table B.1 gives some characteristics of the various systems in the Deep Blue lineage. There are a number of qualifications on the values in this table.

- The ChipTest and Deep Thought systems used software adjustments to account for features not recognized by the evaluation hardware.
- The Deep Blue systems allowed the evaluation features to interact in ways that were not possible in earlier systems.
- The demo version of Deep Blue Jr. was restricted to one second of computation time on one processor, did not detect repetition with game history, and had a fixed evaluation function that did not vary with game stage.

References

[1] T.S. Anantharaman, Extension heuristics, ICCA J. 14 (2) (1991) 135–143.
[2] T.S. Anantharman, M.S. Campbell, F.-h. Hsu, Singular extensions: Adding selectivity to brute-force searching, Artificial Intelligence 43 (1) (1990) 99–110. Also published in: ICCA J. 11 (4) (1988) 135–143.
[3] D.F. Beal, Experiments with the null move, in: D.F. Beal (Ed.), Advances in Computer Chess 5, Elsevier Science, Amsterdam, 1989, pp. 65–79.
[4] H. Berliner, Chess as problem solving: The development of a tactics analyzer, Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1974.
[5] M.G. Brockington, A taxonomy of parallel game-tree search algorithms, ICCA J. 19 (3) (1996) 162–174.
[6] M. Campbell, Knowledge discovery in Deep Blue, Comm. ACM 42 (11) (November 1999) 65–67.
[7] J.H. Condon, K. Thompson, Belle chess hardware, in: Advances in Computer Chess 3, Pergamon Press, Oxford, 1982, pp. 45–54.
[8] R. Feldmann, Spielbaumsuche mit massiv parallelen Systemen, Ph.D. Thesis, Universität-Gesamt-hochschule Paderborn, Germany, 1993.
[9] G. Goetsch, M.S. Campbell, Experiments with the null-move heuristic, in: T.A. Marsland, J. Schaeffer (Eds.), Computers, Chess, and Cognition, Springer, Berlin, 1990, pp. 159–168.
[10] W. Gropp, E. Lusk, A. Skjellum, Using MPI, 2nd Edition, MIT Press, Cambridge, MA, 1999.
[11] E.A. Heinz, Scalable Search in Computer Chess, Friedrick Vieweg & Son, 2000.
[12] F.-h. Hsu, A two-million moves/s CMOS single-chip chess move generator, IEEE J. Solid State Circuits 22 (5) (1987) 841–846.

[13] F.-h. Hsu, Large-scale parallelization of alpha-beta search: An algorithmic and architectural study, Ph.D. Thesis, Carnegie Mellon, Pittsburgh, PA, 1990.
[14] F.-h. Hsu, IBM's Deep Blue chess grandmaster chips, IEEE Micro (March–April 1999) 70–81.
[15] F.-h. Hsu, Behind Deep Blue, Princeton University Press, Princeton, NJ, 2002.
[16] F.-h. Hsu, T. Anantharman, M. Campbell, A. Nowatzyk, A Grandmaster chess machine, Scientific American (October 1990) 44–50.
[17] F.-h. Hsu, T.S. Anantharman, M.S. Campbell, A. Nowatzyk, Deep Thought, in: T.A. Marsland, J. Schaeffer (Eds.), Computers, Chess, and Cognition, Springer, Berlin, 1990, pp. 55–78.
[18] A. Junghanns, Are there practical alternatives to alpha-beta in computer chess?, ICCA J. 21 (1) (1998) 14–32.
[19] D.E. Knuth, R.W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6 (4) (1975) 293–326.
[20] D.N.L. Levy, D. Broughton, M. Taylor, The SEX algorithm in computer chess, ICCA J. 12 (1) (1989) 10–21.
[21] D.A. McAllester, D. Yuret, Alpha-beta-conspiracy search, 1993. http://www.research.att.com/~dmac/abc.ps.
[22] M. Newborn, Kasparov Versus Deep Blue: Computer Chess Comes of Age, Springer, Berlin, 1997.
[23] A. Reinefeld, An improvement to the scout tree-search algorithm, ICCA J. 6 (4) (1983) 4–14.
[24] D.J. Slate, L.R. Atkin, Chess 4.5—The Northwestern University chess program, in: P.W. Frey (Ed.), Chess Skill in Man and Machine, Springer, New York, 1977, pp. 82–118.
[25] G. Tesauro, Connectionist learning of expert preferences by comparison training, in: D. Touretzky (Ed.), Advances in Neural Information Processing Systems 1 (NIPS-88), Morgan Kaufmann, San Mateo, CA, 1989, pp. 99–106.
[26] G. Tesauro, Comparison training of chess evaluation functions, in: J. Furnkranz, M. Kumbat (Eds.), Machines that Learn to Play Games, Nova Science Publishers, 2001, pp. 117–130.
[27] K. Thompson, Retrograde analysis of certain endgames, ICCA J. 9 (3) (1986) 594–597.