Rating Computer Science Via Chess

In memoriam Daniel Kopec and Hans Berliner

Kenneth W. Regan

Department of CSE, University at Buffalo
Amherst, NY 14260 USA; regan@buffalo.edu

Abstract. Computer chess was originally purposed for insight into the human mind. It became a quest to get the most power out of computer hardware and software. The goal was specialized but the advances spanned multiple areas, from heuristic search to massive parallelism. Success was measured not by standard software or hardware benchmarks, nor theoretical aims like improving the exponents of algorithms, but by victory over the best human players. To gear up for limited human challenge opportunities, designers of chess machines needed to forecast their skill on the human rating scale. Our thesis is that this challenge led to ways of rating computers *on the whole* and also rating the effectiveness of our field at solving *hard problems*. We describe rating systems, the workings of chess programs, advances from computer science, the history of some prominent machines and programs, and ways of rating them.

1 Ratings

Computer chess was already recognized as a field when LNCS began in 1971. Its early history, from seminal papers by Shannon [1] and Turing [2], after earlier work by Zuse and Wiener, has been told in [3–5] among other sources. Its later history, climaxing with humanity's dethronement in the victory by IBM's DEEP BLUE over Garry Kasparov and further dominance even by programs on smartphones, will be subordinated to telling how *rating* the effectiveness of hardware and software components indicates the progress of computing. Whereas computer chess was first viewed as an AI problem, we will note contributions from diverse software and hardware areas that have also graced the volumes of LNCS.

In 1971, David Levy was feeling good about his bet made in 1968 with Alan Newell that no computer would defeat him in a match by 1978 [6]. That year also saw the adoption by the World Chess Federation (FIDE) of the Elo Rating System [7], which had been designed earlier for the United States Chess Federation (USCF). Levy's FIDE rating of 2380, representative of his International Master (IM) title from FIDE, set a level of proficiency that any computer needed to achieve in order to challenge him on equal terms.

The Elo system has aged well. It is employed for physical sports as well as games and has recently been embraced by the statistical website *FiveThirtyEight* [8] for betting-style projections. At its heart is a simple idea:

A difference of x rating points to one's opponent corresponds to an expectation of scoring a p_x portion of the points in a series of games.

This lone axiom already tells much. When $x = 0$, p_x must be 0.5 because the two players are interchangeable. The curve likewise has the symmetry $p_{-x} = 1 - p_x$. When x is large, the value p_x approaches 1 but its rate of change must slow. This makes p_x a sigmoid (that is, roughly S-shaped) curve. Two eminent choices are the cumulant of the normal distribution and the simple logistic curve

$$p_x = \frac{1}{1 + e^{-Bx}}, \tag{1}$$

where B is a scaling factor. Originally the USCF used the former with factors to make $p_{200} = 0.75$, but they switched to the latter with $B = (\ln 10)/400$, which puts the expectation of a 200-points higher-rated player a tad under 76%.

If your rating is R and you use your opponents' ratings to add up your p_x for each of N games, that sum is your expected score s. If your actual score S is higher then you gain rating points, else your new rating R' stays even or goes down. Your *performance rating* over that set of games could be defined as the value R_p whose expectation s_p equals S; in practice other formulas with patches to handle the cases $S = N$ or $S = 0$ are employed. The last issue is how far to move R in the direction of R_p to give R'. The amount of change is governed by a factor called K whose value is elective: FIDE makes K four times as large for young or beginning players as for those who have ever reached a rating of 2400.

Despite issues of rating uncertainty whose skew causes actual scores by 200-points higher rated players to come in *under* 75% (see [9]), unproven suspicions of "rating inflation," proven drift between FIDE ratings and those of the USCF and other national bodies, and alternative systems claiming superiority in *Kaggle* competitions [10], the Elo system is self-stabilizing and reasonably reliable for projections. Hence it is safe to express benchmarks on the FIDE rating scale, whose upper reaches are spoken of as follows:

– 2200 is the colloquial threshold to call a player a "master";
– 2400 is required for granting the IM title, 2500 for grandmaster (GM);
– 2600 and above colloquially distinguishes "Strong GMs";
– 2800+ has been achieved by 11 players; Bobby Fischer's top was 2785.

Kasparov was the first player to pass 2800; current world champion Magnus Carlsen topped Kasparov's peak of 2851 and reached 2882 in May 2014. Computer chess players, however, today range far over 3000. How did they progress through these ranks to get there? Many walks of computer science besides AI contributed to confront a *hard problem*. Just how hard in raw complexity terms, we discuss next.

2 Complexity and Endgame Tables

Chess players see all pertinent information. There are no hidden cards as in bridge or poker and no element of chance as in backgammon. Every chess position is well-defined as W, D, or L—that is, winning, drawing, or losing for the player to move. There is a near-universal belief that the starting position is D, as was proved for checkers on an 8×8 board [11]. So how can chess players lose? The answer is that chess is *complex*.

Here is a remarkable fact. Take any program P that runs within n units of memory. We can set up a position P' on an $N \times N$ board—where N and the number of extra pieces are "moderately" bigger than n—such that P' is W if and only if P terminates with a desired answer. Moreover, finding the winning strategy in P' quickly reveals a solution to the problem for which P was coded.

Most remarkably, even if P runs for 2^n steps, such as for solving the Towers of Hanoi puzzle with n rings, individual plays of the game from P' will take far less time. The "Fifty Move Rule" in standard chess allows either side to claim a draw if 50 moves have been played with no capture or pawn advance. Various reasonable ways to extend it to $N \times N$ boards will limit plays to time proportional to N^2 or N^3. The exponential time taken by P is sublimated into the branching of the strategy from P' within these time bounds. For the tower puzzle, the first move frames the middle step of transferring the bottom ring, then play branches into similar but separate combinations for the 'before' and 'after' stages of moving the other $n - 1$ rings.

If we allow P on size-n cases z of the problem to use 2^n memory as well as time, then we must lift the time limit on plays from P', but the size of the board and the time to calculate P' from P and z remain moderate—that is, bounded by a polynomial in n. In terms of computational complexity as represented by Allender's contribution [12], $N \times N$ chess is *complete in polynomial space* with a generalized fifty-move rule [13], and *complete in exponential time* without it [14]. This "double-rail" completeness also hints that the decision problem for chess is relatively hard to parallelize. Checkers, Go, Othello, and similar strategy games extended to $N \times N$ boards enjoy at least one rail of hardness [15–18].

These results as N grows do not dictate high complexity for $N = 8$ but their strong hint manifests quickly in chess. The *Lomonosov tables* [19] give perfect strategies for all positions of up to 7 pieces. They reside only in Moscow and their web-accessible format takes up 140 terabytes. This huge message springs from a small seed because the rules of chess fit on a postcard, yet is *computationally deep* insofar as the effort required to generate it is extreme. The digits of π are as easy as pie by comparison [20]. These tables may be the deepest message we have ever computed.

Even with just 4 pieces, the first item in our history after 1971 shows how computers tapped complexity unsuspected by human players. When defending with king and rook versus king and queen, it was axiomatic that the rook needed to stay in guarding range of the king to avoid getting picked off by a fork from the queen. Such huddling made life easier for the attacker. Computers showed that the rook could often dance away with impunity and harass from the sides to delay up to 31 moves before falling to capture—out of the 50 allotted for the attacker to *convert* by reducing (or changing) the material. Ken Thompson tabulated this endgame for his program BELLE and in 1978 challenged GM Walter Browne to execute the win. Browne failed in his first try, and after extensive study before a second try, squeaked through by capturing the rook on move 50.

Thompson generated perfect tables for 5 pieces with positions tiered by distance-to-conversion (DTC)—that is, the maximum number of moves the defender could delay conversion. In distance-to-mate (DTM), the king and queen versus king and rook endgame can last 35 moves. The 5-piece tables in Eugene Nalimov's popular DTM format occupy 7.1 GB uncompressed. Distance-to-zero (DTZ) is the minimum number of moves to force a capture or pawn move while retaining a W value; if the DTZ is over 50 then its "Z50" flavor flips the position value from W to D in strict accordance with the 50-move draw rule.

Thompson also generated tables for all 6-piece positions without pawns. He found positions requiring up to 243 moves to convert and 262 moves to mate. In many more, the winning strategy is so subtle and painstaking as to be thought beyond human capability to execute. The Lomonosov tables, which are DTM-based, have upped the record to 545 moves to mate—more precisely, 1,088 ply with the loser moving first. Some work on 8-piece tablebases is underway but no estimate of when they may finish seems possible. This goes to indicate that positions with full armies are intractably complex, so that navigating them becomes a heuristic activity. What ingredients allow programs to cope?

3 The Machines: Software to Hardware to Software

Computer chess players began largely as hardware entities but have evolved into software, with enough convergence in basic architecture and interchangeability under APIs that they are now called *engines*. Three main components are identifiable:

1. *Position representation*—by which the rules of chess are encoded and legal moves are generated;
2. *Position evaluation*—by which "knowledge" is converted into numbers; and
3. *Search heuristics*—whose ingenuity marches on through the present.

Generating legal moves is cumbersome especially for the *sliding pieces* bishop, rook, and queen. A software strategy used early on was to maintain and update their limits in each of the compass directions. Limit squares can be off the board, and the trick of situating the board inside a larger array pays a second dividend of disambiguating differences in square indices. For example, the "0x88" layout uses cells 0–7 then 16–23 and so on up to 112–119. Cells pairs with differences in the range [-7,7] must then belong to the same rank (that is, row). The 0x88 layout aligns differences a multiple of 15 southeast-northwest, 16 south-north, and 17 southwest-northeast. Off-board squares are distinguished by having nonzero bitwise-AND with 10001000, which is 0x88 in hexadecimal.

Such tricks go only yea-far, and it became incumbent to implement board operations directly in hardware. As noted by Lucci and Kopec [21], the best computer players from BELLE through DEEP BLUE went this route in the 1980s and 1990s. They avoided the "von Neumann bottleneck" via multiprocessing of both data support and calculation. Chess programs realize less than full benefits of extra processing cores [22], an echo of the parallel hardness mentioned above.

The advent of 64-bit processing decisively favored an alternate representation that had been discussed since the late 1950s: *bitboards*. Instead of storing the position in one 8×8 array, each piece has its own 8×8 binary array—or several—coded as one 64-bit unsigned integer. A rook on the square b2 might be represented by the number 2^9 and its potential moves along the second rank by $r_m = 2^8$ plus the sum of 2^{10} through 2^{15}. If a same-colored piece arrives on a square to its right, coded by $s = 2^i$, then its mobility can be updated by

$$r_m := r_m \mathbin{\&} (s - 1),$$

in just two machine cycles. A similar subtraction trick finds the least bit set to 1 in any position code. Similar operations for files and diagonals, perhaps virtually rotated [23] into horizontal position to avail tricks like this, enable fast move generation and updates. Newer generic hardware instructions, such as population-count (POPCNT) which gives the number of bits set to 1, also speed many operations. All this has lessened the advantage of specialized hardware, exemplified by Robert Hyatt's evolution of CRAY BLITZ into the open-source program CRAFTY.

Evaluation assigns to each position p a numerical value $e_0(p)$. The values are commonly output in discrete units of 0.01 called *centipawns* (cp), figuratively $1/100$ the base value of a pawn. The knight and bishop usually have base values between 300 and 350cp, the rook around 500cp, and the queen somewhere between 850 and 1,000cp. The values are adjusted for positional factors, such as pawns becoming stronger when further advanced and "passed" but weaker when "doubled" or isolated. Greater mobility and attacks on forward and central squares bring higher values. King safety is a third important category, judged by the structure of the king's pawn shield and the proximity of attackers and defenders. The fourth factor emphasized by DEEP BLUE [24] is "tempo," meaning ability to generate threats and moves that enhance the position score. Additional factors face a tradeoff against the need for speedy evaluation, but this is helped by computing them in parallel pipes and by keeping the formula linear. Much human ingenuity goes into choosing and formulating the *factors*, but of late their *weights* have been determined by massive empirical testing (see [25]).

3.1 Search and Soundness

Search has a natural recursive structure. We can replace $e_0(p)$ by the maximum—from the player to move's point of view—of $e_0(p')$ over the set F_1 of positions p' reachable by one legal move, calling this $e_1(p)$. From the other player's point of view those positions have value $e_0'(p') = -e_0(p')$. Now let F_2 be the set of positions p'' reachable by a move from some p' and define $e_1'(p')$ to be the maximum of $e_0'(p'')$ over all p'' reached from p'. From the first player's view this becomes a minimizing update $e_1(p')$; then re-doing the maximization at the root p over these values yields $e_2(p)$. This so-called the *negamax* form of minimax search is often coded as a recursion exactly so. The sequence p', p'' such that $e_2(p) = e_1(p') = e_0(p'')$ (breaking any ties in the order nodes were considered) traces out the *principal variation* (PV) of the search, and the move m_1 leading to p' is the engine's *best-move* (or *first-move*).

Continuing for $d \geq 3$, we define F_d to comprise all positions r reached by one move from a position $q \in F_{d-1}$. Multiple lines of play may go from p to r through different q. Such *transpositions* may also have different lengths so that F_d overlaps F_i for some $i < d$ of the same parity. Given initial evaluations $e_0(r)$ for all $r \in F_d$, minimax well-defines $e_d(p)$ and a PV to a node $r \in F_d$ so that all nodes in the PV have value $e_d(p) = e(r)$. In case of overlap at a node u in F_i the value with higher *generation* subscript—namely j in $e_j(u)$—is preferred. The *simple depth-d search* has $e(r) = e_0(r)$ for all $r \in F_d$, but we may get other values $e(r)$ by *search extension* beyond the *base depth* d, possibly counting them as having higher generation and extending the PV.

The 50-move rule ensures that $e_d(p)$ converges to the true value $+M$, 0, or $-M$ of p, where a big number M is used as the *mate value*. Convergence is aided by the rule that the side bringing the third occurrence of any position in a game can claim a draw. Engines avoid cycles in search by the sharper policy of giving any node q repeating a position earlier in the line of search (or game) a fixed value $e(q) = 0$ of highest generation. The goal of search is to visit a subset E of nodes within a feasible time budget so that minimax from values

$e_0(r)$ over sufficiently many "floor nodes" r in E well-defines a value $v_d(p)$ so that for $c \leq d \leq D$ with d and D as high as possible:

- E includes enough of F_c that no value $e_0(q)$ for an unvisited node $q \in F_c \backslash E$ affects $v_d(p)$ by minimax;
- most of the time this is true for F_d in place of F_c; and
- $v_d(p)$ approximates $e_D(p)$.

The first clause is solidly defined and says that the search is *sound for depth* c. The second clause aspires to soundness for a stipulated depth d and motivates our first considering search strategies that alone cannot violate such soundness. The third clause is about trying to *extend* the search to depths $D > d$ without reference to soundness.

Nearly all chess programs use a structure of *iterative deepening* in successive *rounds* $d = 1, 2, 3, \ldots$ of search. The sizes of the sets $E = E_d$ of nodes visited in round d nearly always follow a geometric series so that the *effective branching factor* (*ebf*) of the search—variously reckoned as $|E_d|/|E_{d-1}|$ or as $|E_d|^{1/d}$ for high enough d—is bounded by a constant. This constant should be significantly less than the "basic" branching factor $|F_d|/|F_{d-1}|$. Similar remarks apply for the overall time T_d to produce $v_d(p)$ and the number N_d of node visits (counting multiple visits to the same node) in place of $|E_d|$.

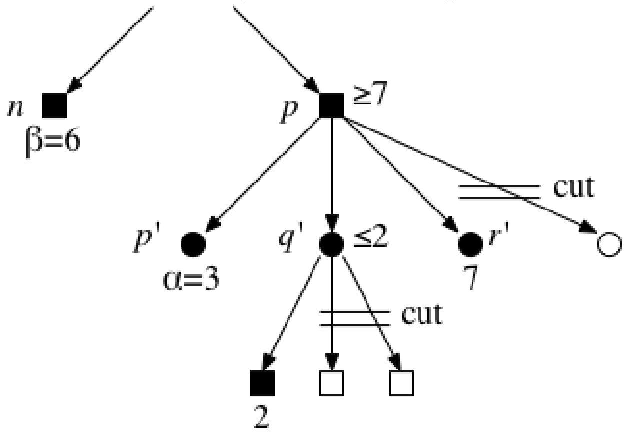
3.2 Alpha-Beta

The first search strategy involves guessing α and β such that our ultimate $v_d = v_d(p)$ will belong to a *window* (α, β) with $\beta - \alpha$ as small as we dare. One motive for iterative deepening is to compute v_{d-1} on which to center the window for round d. Values outside the window are reckoned as "$\geq \beta$" or "$\leq \alpha$" and these endpoint-values work fine in minimax—if $e_d(p)$ crosses one of them then we *fail high* or *fail low*, respectively. After a fail-low we can double the lower window width by taking $\alpha' = 2\alpha - v_{d-1}$ and try again, doing similar for a fail-high, and possibly winding back to an earlier round $d' < d$. Using endpoints relieves the burden of being precise about values away from v_d. This translates into search savings via *cutoffs* described next.

Suppose we enter node p as shown in Figure 1 with window $(1, 6)$ and the first child p' yields value 3 along the current PV. This lets us search the next child q' with the narrower window $(3, 6)$. Now suppose this fails because its first child q'' gives value 2. It returns the value "≤ 2" for q' without needing to consider any more of its children, so search is *cut off* there and we pop back up to p to consider its next child, r'. Next suppose r' yields value 7. This breaks β for p and all further children of p are considered *beta-cutoffs*. If p is the root then this fail-high re-starts the search until we find a bound β' that holds up when $v_d(p)$ is returned. If not—say if the $\beta = 6$ value came from a sibling n of p as shown in the figure—then p gets the value "≥ 6" and pops up to its parent. A value $v_{d-1}(r') = 4$, however, would move the PV to go through r' and keep the search going with exact values in telescoping windows between α and β.

One further note is that if we had advance confidence that the adversary's first reply at q' would show its inferiority to going to p', then we could call search at q' with the *null window* $(3, 3)$ there instead, propagating it downward as needed. If we were wrong then we'd have to undo any ersatz cutoffs from $\beta'' = 3$ along the way, but if we're right then we've pocketed their time savings.

Fig. 1. Alpha-beta search example



Returning to the beta-cutoff from $v(r') = 7$, consider what happened along the new PV in nodes below r'. Every defensive move m' at r' needed to be tried in order to show that none kept the lid under $\beta = 6$; there were no *alpha*-cutoffs on these moves. This situation propagates downward so we've searched all children of half the nodes on the PV. If there are always ℓ such children then we've done about $\ell^{d/2} = (\sqrt{\ell})^d$ work. This is the general best-case for alpha-beta search when soundness is kept at depth d, and it is often approachable. A further move-ordering idea that helps is to try "killer moves" that achieved cutoffs in sibling positions first, even recalling them from searches at previous moves in the game. But with ℓ between 30 and 40 in typical chess positions, optimizing cutoffs alone brings the *ebf* down only to about 6.

Further savings come from storing values $e_j(q)$ at hashed locations $h(q)$ in the *transposition table*. The most common scheme assigns a "random"-but-fixed 64-bit code to each combination of 12 kinds of piece and square. This makes $12 \times 64 = 768$ codes, plus one for the side to move, four for White and Black castling rights, and eight for the files of possible *en-passant* captures. The *primary key* $H(q)$ is the bitwise-XOR of the basic codes that apply to q. Then the *secondary key* $h(q)$ can be defined by $H(q)$ modulo the size N of the hash table, or when $N = 2^k$ for some k, by taking k bits off one end of $H(q)$. Getting $H(r)$ for the next or previous position r merely requires XOR-ing the codes for the destination and source squares of the piece moved, any piece captured, the side-to-move code, and any other applicable codes. Besides storing $e_j(q)$ we store $H(q)$ and j (and/or other "age" information), the former to confirm sameness with the position probed and the latter to tell whether $e_j(q)$ went as deep as we need. If so, we save searching an entire subtree of the current parent of q. We may ignore the possibility of primary-key collisions $H(q) = H(r)$ for distinct positions q, r in the same search. Collisions of secondary keys $h(q) = h(r)$ are frequent but errors from them are often "minimaxed away" (see also [26]).

3.3 Extensions and Heuristics

We can get more mileage by extending D beyond d. Shannon [1] already noted that many depth-d floor nodes come after a capture or check or flight from check and have moves that continue in that vein. Those may be further expanded until a position identified as *quiescent* is reached. Human players tend to calculate such forced sequences as a unit. Thus the game-logical floor for round d may be deeper along important branches than the nominal depth-d floor.

Furthermore, the PV may accrue many nodes q whose value hangs strongly on one move m to a position q', so that a large change to $e_i(q')$ would change $e_{i+1}(q)$ by the same amount. The move m is called *singular* and warrants a better fix on its value by expanding it deeper. Such *singular extensions* could be reserved for cases of delaying moves by a defender on the ropes or moves known to affect positions already seen in the search, or liberalized to consider groups of two or more move options as "singular" [27, 28].

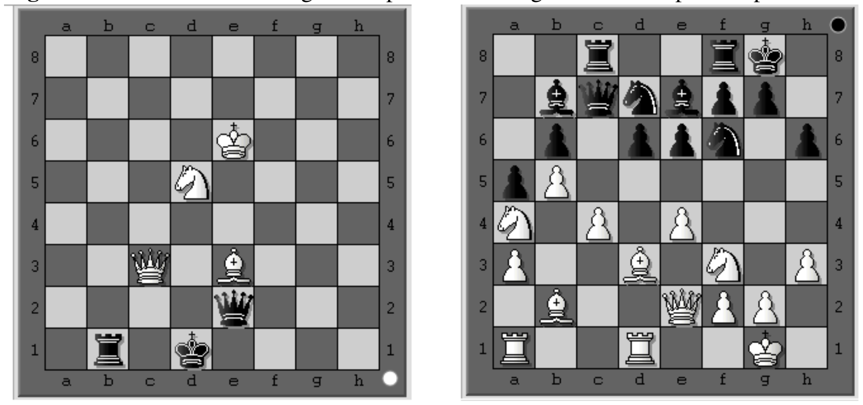
Other extensions have been tried. Search depths are commonly noted as "d/D" where d is the nominal depth and D is the maximum extended depth. Their values $e(r)$ for $r \in F_d$ may differ widely from $e_0(r)$ but this does not violate our notion of depth-d soundness which takes those values $e(r)$ as given. We have added more nodes beyond F_d but not saved any more inside it than we had from cutoffs. Further progress needs compromise on soundness.

From numerous heuristics we mention two credited with much of the software side of performance gain. The idea of *late move reductions* (LMR) is simply to do only the first yea-many moves from the previous round's rank order to nominal depth d, the rest to lower depths c. If $d/c = 2$, say, this can prevent a subtle mate-in-n-ply from being seen until the search has reached round $2n$.

Even $c = d - 4$ or $d - 3$ can make terms in $(\sqrt{\ell})^c$ minor enough to replace $(\sqrt{\ell})^d$ by $(\sqrt{a})^d$ for $a < 4$, which is enough to bring the *ebf* under 2.

The second idea compresses search "vertically" rather than "horizontally" in situations where we are trying to prove a cutoff value v after a "killer" but might not know how to order our subsequent moves to cut off lower down too. If the defender is really bad off then allowing two moves in a row might not improve the score beyond v or much at all. Inserting *null moves* for our turns can cement the search-depth halving on our side and also branch on fewer defensive sequences than using two alternating levels of search would bring. To be sure, there are so-called *Zugzwang* situations where letting the opponent move twice gives *us* an unfair advantage—propagating the illusion of "killer moves" when there really are none. However, these situations tend to occur in endgames where they are recognizable in advance and errors especially for nodes away from the PV may be stopped by minimax from propagating to the root.

Fig. 2. Left: Position illustrating search phenomena. Right: Bratko-Kopec test position 22.



The position at left in Figure 2 illustrates many of the above concepts. The Lomonosov 7-piece tables show it a draw with best play. Evaluation gives White a 100–200cp edge on material with bishop and knight versus rook, but engines may differ on other factors such as which king is more exposed. After 1. Qd4+ Kc2 2. Qc5+, Black's king cannot return to d1 because of the fork 3. Nc3+, so Black steps out with 2...Kb3. Then White has the option 3. Qb6+ Kc2 4. Qxb1+ Kxb1 5. Nc3+ Kc2 6. Nxe2. Since Black is not in check and has no captures, this position may be deemed *quiescent* and given a +600 to +700 value or even higher since the extra bishop plus knight is generally a winning advantage. However, Black has the quiet 6...Kd3 which forks the bishop and knight and wins one of them, leaving a completely drawn game. What makes this harder to see is that White can delay the reckoning over longer *horizons* by giving more checks: 4. Qc7+ Kb3 5. Qb8+ Kc2 6. Qc8+ Kb3 7. Qb7+ Kc2 8. Qc6+ Kb3. White has not repeated any position and now has three further moves 9. Qc3+ Ka2 (if Black rejects ...Ka4) 10. Qa5+ Kb3 11. Qb4+ Kc2 before needing to decide whether to take the plunge with 12.Qxb1+. Pushing things even further is that White can preface this with 1. Ke7 threatening 2. Nb4 with Black's queen unable to give check. Black must answer by 1...Rb7+ and after 2. Kd6 must meekly return by 2...Rb1. Especially in the position after 1. Ke7 Rb7+, values can differ widely between engines and between depths for the same engine and be isolated to changes in the size of the hash table. Evidently the high degree of singularity raises the chance of a rogue $e(r)$ value propagating to the root.

How often is the quality of play compromised? It is one thing to try these heuristics against human players, but surely a "sounder" engine is best equipped to punish any lapses. Silver [29] reports an experiment where a current engine running on a smartphone trounced one from ten years ago that was given hardware fifty times faster. Although asking for depth d really gives a *mélange* of c and D with envelope E lopsidedly bunched along the PV, it all works.

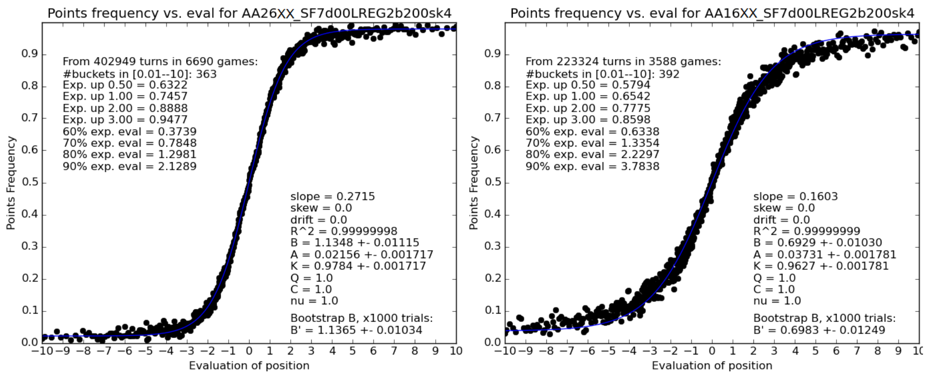
We have glossed over many variants and ideas, including Hans Berliner's B^* search [30] which uses endpoints exclusively. Many have been studied and debated in the journal and symposia of the International Computer Chess Association, now evolved into the International Computer *Games* Association (ICGA), including LNCS conference proceedings. We argue that their sum achievement is most neatly expressed by plotting the engines' position values v against the portion p_v of points that human players of a given rating went on to score from positions of value v with either side to move. Figure 3 plots this from all standard-time games recorded in [31] between players rated within 10 points of a "milepost" 2600, 2625, 2650, or 2675, and likewise for levels in the 1600s range. Both sets give a near-perfect fit to a two-parameter logistic curve:

$$p_v = A + \frac{1 - 2A}{1 + e^{-Bv}}. \tag{2}$$

Here A represents the frequency of losing or drawing a "completely won" game and is small enough that we can focus on B. The one parameter B does double-duty: it is the scaling conversion from engine values to expectation and also scales with the skill of the players. The y-axis and B are the same as in our equation (1) for expectation based on rating difference. This suggests that skill is largely the sharpness of perceptions of value. If a chess program were to value a queen at 15 rather than 9 and so on for other terms in its evaluation function, we would have to scale B down by $3/5$ to preserve the correspondence to scoring frequency. The figures have about the same ratio of B, which suggests that values are 60% more vivid to 2600s-rated players than to 1600s-rated players.

Their simplicity gives such curves the force of natural law. Amir Ban, co-creator of the (DEEP) JUNIOR chess program, argued [32] that the logistic relationship optimizes both the predictive accuracy and playing skill of the programs. In a skin-deep way this is *false*: the programs can post-process their values in any way that preserves the rank order of moves without affecting their play. In order to rule out this possibility, we have used the *open-source* STOCKFISH program (official version 7 release) to analyze the human games for the plots. That the evaluation terms, search heuristics, and minimax dynamics conform to the logistic relationship shows their *natural* acuity.

Fig. 3. Points expectation for 2600s-rated and 1600s-rated players from STOCKFISH 7 values.



4 Benchmarking Progress

All the notable human-computer matchups under standard tournament conditions over the past 40 years total in the low hundreds of games. A dozen such games at most are available for major iterations of any one machine or program. Games in computer-computer play do not connect into the human rating system. With ratings based only on a few bits of information—the outcomes of games and opponents' ratings—the sample size is too small to get a fix. Ratings based on 25 or fewer games are labeled "provisional" by the USCF. However much we feel the lack in retrospect, it applied all the more years ago looking forward.

Various internal ways were used to project skill. Programs could be played against themselves with different depth or time limits of search. The scoring rate of the stronger over the weaker translates into an Elo difference by the curve (1). Thompson [33] carried this out with BELLE at single-digit search depths, finding a steady gain of about 200 Elo per extra ply, but a followup experiment joined by Condon [34] found *diminishing returns* beyond depth 7.

The two prior versions of CHESS 4.7 triumphed in amateur and regional tournaments before its match with Levy, but the first provisional ratings above 2200 were earned by CRAY BLITZ and BELLE in the early 1980s. Berliner integrated his B^* search and high-tech parallel hardware to make his HITECH machine the first recognized as surpassing 2400 in 1988. Feng-hsiung Hsu, Thomas Anantharaman, and Murray Campbell, working apart from Berliner at Carnegie Mellon, developed CHIPTEST. Mike Browne and Andreas Nowatzyk joined them for DEEP THOUGHT, which was the first to beat a GM (Bent Larsen) in regulation play and gain a GM-level rating (2552). A flurry of activity followed in 1989 but with no clear forecast of further progress. Berliner et al. [35] conducted extensive self-play experiments and were led to state in their abstract:

> Projections of potential gain have time and again been found to overestimate the actual gain. [Our work] suggests that once a certain knowledge gap has been opened up, it cannot be overcome by small increments in searching depth. The conclusion . . . is that extending the depth of search without increasing the present level of knowledge will not in any foreseeable time lead to World Championship level chess.

Hsu et al. [36] reached the opposite conclusion regarding DEEP THOUGHT, projecting that a 14 or 15-ply basic search with extensions beyond 30 ply would achieve a 3400 rating. The Thoresen engine competition site today shows no rating above 3230 [37]. One can say that its evolution into DEEP BLUE landed between the two projections. A chart from 1998 by Moravec [38] seems to justify the extrapolation to 3400 by its notably linear plot of ascribed engine ratings up to DEEP THOUGHT II near 2700 and 11 ply in 1991 and 1994, but it plots DEEP BLUE well under the line at 13 ply and only a 2700–2750 rating.

Already in the late 1970s, Ivan Bratko and Daniel Kopec conceived that an *external* test applicable to both human and computer players and less taxing than fully staged games could provide a reliable metric. The published form [39, 40] was a suite of twenty-four positions, twelve on tactics and twelve emphasizing strategy of pawn structure in particular. The former are instantly solved by today's computers but the latter retain their challenge, especially position 22 pictured in Fig. 2 which they deemed "hardest." The official STOCKFISH 8 version with 256MB hash on one core thread in its "Single-PV" playing mode takes until depth 26 to settle on the key move—yet this happens within 20 seconds on an eight-year-old PC. Writing in 1990, Marsland [41] opined:

> Although one may disagree with the choice of test set, question its adequacy and completeness, and so on, the fact remains that the designers of computer chess programs still do not have an acceptable

means of estimating the performance of chess programs, without resorting to time-consuming and expensive "matches" against other subjects. Clearly there is considerable scope for such test sets, as successes in related areas like pattern recognition attest.

What further distinguished the Bratko-Kopec work were tests on human subjects rated below-1600, 1600–1799, 1800–1999, 2000–2199, 2200–2399, and 2400+. The results filled the whole range from only two correct out of 24 to 21-of-24, showing a clear correspondence to rating. The Elo rating chart in [40] assigned 2150 to BELLE, 2050 to CHESS 4.9, and ratings 1900 and under to DUCHESS and other tested programs. Their results were broadly in accord with those ratings. But all these results were from small data.

Guy Haworth [42] proposed using endgame tables to benchmark humans—and computers not equipped with them. The DTM, DTC, and/or DTZ metrics furnish numerical scores that are indisputable and objective, and the 6- and later 7-piece tables expand the range of realistic test positions. Humans of a given rating class could be benchmarked from games in actual competition that entered these endgames.

Matej Guid led Bratko back into benchmarking with a scheme using depth 12 of CRAFTY as authority to judge all moves (after the first twelve turns) in all games from world championship matches by intrinsic quality [43]. This was repeated with other engines as judges [44] including then-champion RYBKA 3 to reported depth 10, which arguably compares best to depth 13 or 14 on other engines since RYBKA treats its bottom four search levels as a unit [45]. Coming back to Haworth and company joined by this writer, two innovations of [46, 47] were doing un-pruned full-depth analysis of multiple move options besides the best and played moves, and judging prior likelihoods of those moves by *fallible agents* modeling player skill profiles in a Bayesian setting. This led in [48] to using RYBKA 3 to analyze essentially all legal moves to reported depth 13, training a frequentist model on thousands of games over all rating classes from 1600 to 2700, and conditioning noise from the observed greater magnitude of errors in positions where one side has a non-negligible advantage. The model supplies not only metrics and projections but also error bars for various statistical tests of concordance with the judging engine(s) and an "Intrinsic Performance Rating" (IPR) based only on analysis of one's moves rather than results of games.

For continuity with this past work—and because an expanded model with versions of KOMODO and STOCKFISH as judges is not fully trained and calibrated at press time—we apply the scheme of [48] to rate the most prominent human-computer matches as well as some ICCA/ICGA World Computer Chess Championships (WCCC). This comes with cupfuls of caveats: RYBKA 3 to reported depth 13 is far stronger than CRAFTY to depth 12 but needs the defense

[49] of the latter to justify IPR values over 2900 and probably loses resolution before 3100. The IPR currently measures *accuracy* more than *challenge* put to the opponent and is really measuring similarity to RYBKA 3. Although moves from turn 9 onward (skipping repeating sequences and positions with one side ahead over 300cp) give larger sample sizes than games, the wide two-sigma error bars reflect the overall paucity of data and provisional nature of this work.

5 A "Moore's Law of Games" and Future Prospects

Table 4 lays out IPRs over 37 years of top events in computer chess. Despite individual jumps in results, their wide error bars, and loss of resolution beyond 3000, some coherent points emerge from the long view:

- There has been steady progress.
- Early estimated ratings of computers were basically right.
- Computers had GM level in sight before DEEP THOUGHT's breakthrough.
- Not long after the retirement of DEEP BLUE, championship quality became accessible to off-the-shelf hardware and software.
- A few years later smartphones had it, e.g. HIARCS 13 as "Pocket Fritz."
- Progress as measured by Elo gain flattens out over time.

The last point bears comparison with Moore's Law and arguments over its slowing or cessation. Those arguments pivot on whether the law narrowly addresses chip density or clock speed or speaks more general measure of productivity. With games we have a fixed measure—results backed by ratings—but a free-for-all on how this productivity is gained.

We may need to use Elo's transportability to other games to meter future progress. The argument that Elo sets a hard ceiling in chess goes as follows: We can imagine that today's strong engines E could hold a non-negligible portion d of draws against *any* strategy. This may need randomly selecting slightly inferior moves to avoid strategies with foresight of deterministic weaknesses. If E has rating R, then no opponent can ever be rated higher than $R + x$ by playing E, where with reference to (1), $p_{-x} = 0.5d$. The ceiling $R+x$ may be near at hand for chess but higher for Go—despite its recent conquest by Google DeepMind's ALPHAGO [50]. Games of Go last over a hundred moves for each player and have hair-trigger difference between win and loss.

A greater potential benefit comes from how large-scale data from deep engine analysis of human games may reveal new regularities of the human mind, especially in decision-making under pressure. Why and when do we stop thinking and take action, and what causes us to err? For instance, this may enable transforming the analysis of blunders in [51] into a smooth treatment of error in

Year	Engine	Score	IPR	moves	Event/Opponent(s)	Opp. IPR	moves
1978	CHESS 4.7	1.5/6	2120 +- 490	159	IM David Levy	2280 +- 415	159
1983	BELLE	5.5/10	2180 +- 300	279	US Open oppts.	2175 +- 320	280
1983	BELLE	3/5	2070 +- 415	126	WCCC oppts.	2130 +- 420	131
1983	CRAY BLITZ	4.5/5	2265 +- 350	144	WCCC oppts.	2065 +- 405	152
1986	CRAY BLITZ	4/5	2605 +- 315	153	WCCC oppts.	2135 +- 395	155
1986	HITECH	4/5	1975 +- 625	111	WCCC oppts.	1805 +- 660	115
1988	HITECH	5/7	2495 +- 270	188	avail. Open oppts.	2165 +- 385	194
1989	HITECH	3.5/4	3085 +- 275	97	GM A. Denker	2100 +- 555	98
1989	HITECH	3.5/5	2445 +- 325	146	WCCC oppts.	2485 +- 245	149
1989	BEBE	4/5	2415 +- 420	141	WCCC oppts.	1910 +- 505	144
1989	CRAY BLITZ	3.5/5	2470 +- 375	195	WCCC oppts.	2255 +- 360	196
1989	CHIPTEST	2/5	2540 +- 285	185	The Hague oppts.	2545 +- 250	181
1989	DEEP THOUGHT	5/5	2600 +- 255	126	WCCC oppts.	1890 +- 445	132
1988	DEEP THOUGHT	5.5/7	2780 +- 230	269	Long Beach oppts.	2400 +- 265	270
1989	DEEP THOUGHT	4/4	2885 +- 325	74	Levy	1820 +- 560	79
1989	DEEP THOUGHT	2/4	2325 +- 400	85	GM R. Byrne	2215 +- 690	83
1989	DEEP THOUGHT	2/5	2955 +- 245	131	Miles/Renet/Valvo	2585 +- 275	131
1989	DEEP THOUGHT	3.5/4	2830 +- 290	130	Amer. Open oppts.	1990 +- 440	133
1989	DEEP THOUGHT	0/2	2265 +- 815	53	Kasparov	2445 +- 340	51
1991	DEEP THOUGHT	2.5/7	2205 +- 430	213	Hannover oppts.	2400 +- 265	214
1993	DEEP BLUE	1.5/4	2820 +- 215	173	GM Bent Larsen	2800 +- 210	172
1993	DEEP BLUE	4.5/9	2720 +- 210	249	Copenhagen oppts.	2340 +- 295	246
1995	DEEP BLUE	3/3	3080 +- 220	100	ACM oppts.	2550 +- 420	103
1995	DEEP BLUE	3.5/5	2695 +- 430	119	WCCC oppts.	2420 +- 475	120
1996	DEEP BLUE	2/6	2915 +- 200	222	Kasparov	2610 +- 235	220
1997	DEEP BLUE	3.5/6	2850 +- 190	205	Kasparov	2585 +- 260	205
1999	REBEL 10	0.5/2	2915 +- 590	58	GM V. Anand	2660 +- 605	58
2000	DEEP JUNIOR	4.5/9	2845 +- 165	300	Dortmund oppts.	2605 +- 220	298
2002	DEEP FRITZ	4/8	3055 +- 140	221	GM V. Kramnik	2885 +- 155	221
2003	DEEP JUNIOR	3/6	2855 +- 275	148	Kasparov	2750 +- 305	148
2003	FRITZ X3D	2/4	2955 +- 175	107	Kasparov	2475 +- 395	108
2004	FRITZ	3.5/4	2945 +- 255	134	Bilbao HC oppts.	2530 +- 305	136
2004	HYDRA	3.5/4	3045 +- 230	176	Bilbao HC oppts.	2510 +- 275	176
2004	DEEP JUNIOR	1.5/4	2835 +- 290	124	Bilbao HC oppts.	2910 +- 155	121
2005	FRITZ	2/4	2705 +- 350	170	Bilbao HC oppts.	2740 +- 280	170
2005	HYDRA	3/4	3080 +- 190	99	Bilbao HC oppts.	2600 +- 340	101
2005	JUNIOR	3/4	3085 +- 100	251	Bilbao HC oppts.	2935 +- 115	251
2005	SHREDDER	9.5/10	2990 +- 165	239	Lopez ITT oppts.	2265 +- 275	243
2005	HYDRA	5.5/6	3160 +- 115	210	GM M. Adams	2825 +- 175	208
2006	DEEP FRITZ	4/6	2985 +- 160	208	Kramnik	2740 +- 265	208
2009	POCKET FRITZ	9.5/10	2905 +- 165	290	Mercosur oppts.	2250 +- 265	292
2011	JUNIOR	6/8	3065 +- 120	311	next 4 in WCCC	3035 +- 65	1,418
2013	JUNIOR	7.5/10	2995 +- 120	446	next 4 in WCCC	3095 +- 55	1,615
2015	JONNY	7/8	2970 +- 110	432	next 4 in WCCC	3035 +- 50	1,668

Fig. 4. IPRs from major human-computer events and some computer championships.

perception. Although computer chess left the envisaged mind and knowledge-based trajectory, its power-play success may boost the original AI aims.

References

1. Shannon, C.: Programming a computer for playing chess. Philosophical Magazine **41** (1950) 256–275
2. Turing, A.: Computing machinery and intelligence. Mind **59** (1950) 633–660
3. Marsland, T.A.: A short history of computer chess. In: Computers, Chess, and Cognition. Springer-Verlag, New York (1990) 3–7
4. Campbell, M., Feigenbaum, E., Levy, D., McCarthy, J., Newborn, M.: The History of Computer Chess: An AI Perspective. `http://www.computerhistory.org/collections/catalog/102651382` (2005) Video, The Computer History Museum.
5. Larson, E.: A brief history of computer chess. The Best Schools Magazine (2015)
6. Levy, D.: Computer chesspast, present and future. Chess Life and Review **28** (1973) 723–726
7. Elo, A.: The Rating of Chessplayers, Past and Present. Arco Pub., New York (1978)
8. Silver, N.: Introducing Elo Ratings. `https://fivethirtyeight.com/datalab/introducing-nfl-elo-ratings/` (2014)
9. Glickman, M.E.: Parameter estimation in large dynamic paired comparison experiments. Applied Statistics **48** (1999) 377–394
10. Sonas, J., Kaggle.com: Chess ratings: Elo versus the Rest of the World. `http://www.kaggle.com/c/chess` (2011)
11. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science **317** (2007) 1518–1522
12. Allender, E.: The complexity of complexity. In: This volume. (2017)
13. Storer, J.: On the complexity of chess. J. Comp. Sys. Sci. **27** (1983) 77–100
14. Fraenkel, A., Lichtenstein, D.: Computing a perfect strategy for n x n chess requires time exponential in n. Journal of Combinatorial Theory **31** (1981) 199–214
15. Lichtenstein, D., Sipser, M.: Go is polynomial-space hard. Journal of the ACM **27** (1980) 393–401
16. Robson, J.: The complexity of Go. In: Proceedings of the IFIP Congress. (1983) 413–417
17. Robson, J.: N by N checkers is Exptime complete. SIAM Journal on Computing **3** (1984) 252–267
18. Iwata, S., Kasai, T.: The Othello game on an n*n board is PSPACE-complete. Theoretical Computer Science **123** (1994) 329–340
19. Zakharov, V., Makhnychev, V.: Creating tables of chess 7-piece endgames on the Lomonosov supercomputer. Superkompyutery **15** (2013)
20. Bailey, D., Borwein, P., Plouffe, S.: On the rapid computation of various polylogarithmic constants. Mathematics of Computation **66** (1997) 903–913
21. Lucci, S., Kopec, D.: Artificial Intelligence in the 21st Century. Mercury Learning, Dulles, Virginia USA (2013)
22. Chess Programming Wiki: Parallel Search. `chessprogramming.wikispaces.com/Parallel+Search` (2017 (accessed))
23. Hyatt, R.: Rotated bitmaps, a new twist on an old idea. ICCA Journal **22** (1999) 213–222
24. IBM Research: How Deep Blue works. `https://www.research.ibm.com/deepblue/meet/html/d.3.2.html` (1997)
25. Chess Programming Wiki: Automated Tuning. `https://chessprogramming.wikispaces.com/Automated+Tuning` (2017 (accessed))

26. Hyatt, R., Cozzie, A.: The effect of hash signature collisions in a computer chess program. ICGA Journal **28** (2005) 131–139
27. Anantharaman, T., Campbell, M., Hsu, F.: Singular extensions: Adding selectivity to brute-force searching. Artificial Intelligence **43** (1990) 99–110
28. Hsu, F.H.: Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. Princeton University Press (2002)
29. Silver, A.: Komodo 8: the smartphone vs desktop challenge. `https://en.chessbase.com/post/komodo-8-the-smartphone-vs-desktop-challenge` (2014)
30. Berliner, H.: The B* tree search algorithm: A best-first proof procedure. Artificial Intelligence **12** (1979) 23–40
31. ChessBase: Big2017 Chess Database (2017)
32. Ban, A.: Automatic learning of evaluation, with applications to computer chess. Technical Report Discussion Paper 613, Center for the Study of Rationality, Hebrew University (2012)
33. Thompson, K.: Computer chess strength. In: Advances in Computer Chess 3, Pergamon Press (1982) 55–56
34. Condon, J., Thompson, K.: Belle. In Frey, P., ed.: Chess Skill in Man and Machine. Springer-Verlag (1982) 201–210
35. Berliner, H., Geotsch, G., Campbell, M., Ebeling, C.: Measuring the performance potential of chess programs. **43** (1990) 7–21
36. Hsu, F.H., Anantharaman, T., Campbell, M., Nowatzyk, A.: A grandmaster chess machine. Scientific American **263** (1990) 44–50
37. Top Chess Engine Championship: Ratings after Season 9 - Superfinal. `http://tcec.chessdom.com/archive.php` (2017 (accessed))
38. Moravec, H.: When will computer hardware match the human brain? Journal of Evolution and Technology **1** (1998)
39. Bratko, I., Kopec, D.: A test for comparison of human and computer performance in chess. In: Advances in Computer Chess 3, Elsevier (1982) 31–56
40. Kopec, D., Bratko, I.: The bratko-kopec experiment: a comparison of human and computer performance in chess. In: Advances in Computer Chess 3, Elsevier (1982) 57–72
41. Marsland, T.: The Bratko-Kopec test revisited. ICCA Journal **13** (1990) 15–19
42. Haworth, G.: Reference fallible endgame play. ICGA Journal **26** (2003) 81–91
43. Guid, M., Bratko, I.: Computer analysis of world chess champions. ICGA Journal **29** (2006) 65–73
44. Guid, M., Bratko, I.: Using heuristic-search based engines for estimating human skill at chess. ICGA Journal **34** (2011) 71–81
45. Rajlich, V., Kaufman, L.: Rybka 3 chess engine. `www.rybkachess.com` (2008)
46. DiFatta, G., Haworth, G., Regan, K.: Skill rating by Bayesian inference. In: Proceedings, 2009 IEEE Symposium on Computational Intelligence and Data Mining (CIDM'09), Nashville, TN. (2009) 89–94
47. Haworth, G., Regan, K., DiFatta, G.: Performance and prediction: Bayesian modelling of fallible choice in chess. In: 12th ICGA Conference on Advances in Computer Games, Pamplona, May 2009. Volume 6048 of Lect. Notes Comp. Sci., Springer-Verlag (2010) 99–110
48. Regan, K., Haworth, G.: Intrinsic chess ratings. In: Proceedings of AAAI 2011, San Francisco. (2011) 834–839
49. Guid, M., Pérez, A., Bratko, I.: How trustworthy is Crafty's analysis of world chess champions? ICGA Journal **31** (2008) 131–144
50. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529** (2016) 484–489
51. Chabris, C., Hearst, E.: Visualization, pattern recognition, and forward search: Effects of playing speed and sight of the position on grandmaster chess errors. Cognitive Science **27** (2003) 637–648