Perspectives on Falling From Grace

Mikhail Donskoy
Institute for System Studies
Moscow, USSR

Jonathan Schaeffer
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

ABSTRACT

Research in computer chess has been active for over three decades. Over that period, computer chess has fallen from the position of being a prominent research application in artificial intelligence to a peripheral area. In this paper, we take a retrospective look at what has been accomplished, in order to understand where the field is today and where it is headed tomorrow. Whereas the past has often been clouded by engineering passing as science, misspent effort for short-term gains, and research results with little applicability to other domains, there is evidence that computer chess is emerging from the shadow of its past and may now be recapturing some of its lost stature in the research world.

1. Introduction

Research in computer chess has progressed over that past three decades to the point where programs will soon be grandmaster strength. Superficially this accomplishment seems impressive. However, the building of a chess program that is better than all humans is a scientific pursuit, and it is relevant to ask what the lasting scientific contributions of achieving this result are. Perhaps the ends justify the means, but what are the means by which the computer chess community strives for this lofty goal? Rather than evaluating the accomplishments of computer chess research, in this paper we consider the methods by which the results have been obtained.

For the fledgling area of machine intelligence research (artificial intelligence or *AI*) chess was viewed as an important test application. Some researchers regarded computer chess as the *drosophila* of AI . However, three decades of progress has relegated the problem of building chess programs to only a peripheral relationship with AI. AI has moved on to other applications, even though the chess problem is still far from being solved. Instead, computer chess seems to have become a separate research area, with only historical ties to AI. To many researchers, it is regarded as mere "game playing". Why has computer chess fallen from grace from the mainstream research community?

These questions cannot be completely addressed here; there are many points of view and different sides to be argued. This paper examines computer chess from the research perspective with a view to better understanding computer chess as a science. A retrospective look at what has been accomplished is presented with a view to understanding where the field is today and where it is headed tomorrow. Whereas the past has often been clouded by engineering passing as science, misspent effort for short-term gains, and research results with little applicability to other domains, there is evidence that computer chess is emerging from the shadow of its past and may be recapturing some lost stature in the research world.

Chess is to artificial intelligence as the fruit fly is to genetics.

Section 2 puts some of the issues into historical perspective. Section 3 critiques the scientific aspects of computer chess, to gain a better understanding of why the area has fallen from grace. Many will argue that computer chess is now more engineering than science, and these issues are addressed in section 4. Section 5 concludes the paper.

2. Historical Trends

The development of computer chess programs can be divided into three distinct eras. The first, or *pioneering* era, spanned the 1950s to the mid 1970s. It was characterized by the extensive use of chess knowledge to both guide the search and assess chess positions. By today's standards, many of these programs used *ad hoc* techniques.

The second phase, or *technology* era, began in the mid 1970s and was marked by the exploitation of brute-force alpha-beta (αβ) search. The observation that there was a strong correlation between machine speed and program performance meant that one had only to use faster computers in order to improve performance. The advent of fast sequential processors, followed by parallel systems, and finally special purpose machines, resulted in a dependence on advancing technology to solve the chess problem. To some extent, this era is not yet over but it is clear that with massively parallel VLSI chess programs (for example, Hitech [10] and Deep Thought [14]) there is not much further one can go in this area, although there are still interesting problems to be solved (such as how to use parallel processors effectively).

Recently, a new era of computer chess work has been begun. With current technology being pushed to its limits, it was time to go back and, in some sense, re-discover the ideas and motivations of the pioneering era. This *algorithm* era is resulting in a variety of innovative ways of searching (for example, [3, 4, 17, 22, 27]) and uses of knowledge (for example, [7]).

One way distinguishing between the eras is by the search methods that dominated each era. Programs of the pioneering era used selective search methods exclusively. Their search algorithms were unstable, required extensive application-dependent knowledge, were difficult to debug, and it was hard to predict their performance. The technology era preferred the simplicity of brute-force αβ. αβ was easy to implement, reliable, and had predictable performance. The new era is still dominated by αβ, but the ideas of *selective deepening* have complicated the search procedure (for example, [3]). This era represents a compromise in search strategy over the extreme positions of its predecessors.

Perhaps the preceding discussion best illustrates why computer chess fell from grace. With brute-force αβ searching, chess became an uninteresting problem to AI. The solution to improve performance was to just build faster machines. However, some of the fundamental problems of playing good chess (as for example those outlined in [5]) are beyond the reach of hardware technology alone. After a decade of chasing the faster machine mirage, it now appears necessary to return to the basics, finding better search algorithms and better ways of using chess knowledge to solve the problem. In some sense, research in computer chess has come full circle.

3. Scientific Credibility

This section outlines some of the obstacles to computer chess achieving credibility in the scientific community. The topics mentioned do not pretend to be exhaustive, but illustrate some of the major obstacles.

3.1. Beyond Alpha-Beta.

In the technology era, work on finding new search algorithms almost ceased. The emphasis was placed on finding ways of improving αβ search. Then several researchers discovered that chess programs were building trees which were within a factor of two of the minimal tree [8, 24]. Where was there to go, if perfect search efficiency was only a factor of two away?

Happily, the past few years have seen a resurgence of interesting new approaches to game-tree search. Null moves, actually an old idea [2] but with new applications [4, 11, 25], and singular extensions [3] have enhanced the productivity of αβ search by concentrating effort where it is most likely to succeed. Entirely new approaches to minimax search include conspiracy numbers [17, 26], min/max approximation [22], and Solution Tree and Costs search [27]. Perhaps it is also time to revive some of the (relatively) older search ideas such as B* [6, 19], the method of analogies [1], and the notion of influence [2]. All of these ideas merit serious consideration.

For many years, the strength of the $\alpha\beta$ cutoff caused us to ignore alternate search algorithms. The idea was simple and yields an exponential savings in tree size. It is unfortunate that computer chess was given such a powerful idea so early in its formative stages. Conspiracy numbers has provided the insight that there is important information to be gained at nodes where we normally cut-off. The singular extensions algorithm is the first algorithm that uses this information to find its way into practice.

To the average researcher in AI, yet another way to search minimax trees is of little interest. It is an algorithmic problem, not an AI problem. Until we develop algorithms where the search is guided by knowledge, this area will be of little interest outside the small community of computer chess aficionados.

3.2. A Little Knowledge Can Go a Long Way, and a Lot of Knowledge...?

It is amazing how far computer chess programs have progressed using minimal chess knowledge. In fact, it is not uncommon to observe *better* performance in a program with *less* knowledge. Unfortunately, this apparent paradox is often used as a rationalization for avoiding the issues of knowledge representation and usage in chess programs.

Computer chess programs have been stymied by the *knowledge acquisition* bottleneck. Chess grandmasters have a large store of knowledge at their disposal. It is difficult to extract that knowledge in machine usable terms. Humans often describe chess positions in abstract terms that are hard to quantify. Further, the influence of intangibles on a grandmaster's play, such as intuition, are not to be under-estimated. These problems are not dissimilar from those encountered by expert system designer, this being a fundamental problem for all of AI.

There are at least three major problems problems here. First, there is the quantity of knowledge required. It has been estimated that a chess grandmaster has 50,000 patterns or *chunks* of knowledge at his disposal [9]. Second, there is the quality of the knowledge. Some pieces of knowledge are adequate for improving the play of non-masters, but as the strength of the opponents increase, the quality and complexity of the knowledge required increases. Third is the way the knowledge is represented. Many common chess notions are easily understood by humans but are hard to capture in machine usable form (e.g. initiative). Often one resorts to approximations that capture some of the effect without understanding the implications of the loss of accuracy.

Perhaps the difficulty in using chess knowledge is best summed up as follows. One grandmaster stated that he spent the first half of his career learning the basic principles of chess, and the second half trying to unlearn them! The problem encountered when using chess knowledge (or any non-factual knowledge for that matter) is that for every heuristic, there is an exception. For some heuristics, the exceptions are so infrequent that most chess programs ignore them. Yet, it is precisely under these conditions that human players excel. In this area, so closely akin to other problems in AI, computer chess researchers have made little progress. It is probably fair to say that most ignore the problem.

Given the enormity of the task of acquiring, representing, and using knowledge, why haven't more computer chess researchers tackled the problem? Some recent attempts are basically simplistic and somewhat naive while avoiding the real (harder) problems [12, 13, 16, 23]. Some progress has been made recently in this area [7] but this can only be viewed as a constructive start.

For example, deeper searches compensating for less detailed knowledge [7].

Kevin Spraggett, Canadian Grandmaster, private communication.

One can view AI as having to tackle three main problems: representing knowledge, searching for solutions, and the interactions between search and knowledge. For effective search, some knowledge must be applied. On the other hand, to use applicable knowledge in large problem domains one needs good search methods. The two are not separable. The majority of computer chess research is on search algorithms, with the issues of knowledge and the search-knowledge interaction largely being neglected.

Will brute-force search be sufficient to achieve world championship level play? If so, then such a program would tell us little that could be of use in the wider field of AI. If not, then perhaps chess programmers should be addressing the fundamental problems of acquiring and using knowledge now!

3.3. A Conflict of Interest?

Unlike most other fields of computer science (and indeed, unlike most other sciences), computer chess publicly exhibits its progress. Since 1970, there have been annual tournaments that measure the rate of program improvement. The consequences are both good and bad.

The desire to have one's chess program perform as well as possible in the public spotlight is strong motivation for many to work ardently on their programs. The potential returns for success in these tournaments is large, whether the benefits are monetary, publicity, or prestige. There is no doubt that these tournaments have been a tremendous boost to computer chess.

However, there is a serious disadvantage to this continuous public attention. As a scientist, one searches for the solution to difficult problems, even if success may take many years. How does one resolve this with annual spectacles where one has an obligation to have an improved program each year? This encourages small, short-term projects whose results will have a direct bearing on program performance. Long-term research projects, work on the really difficult problems that we must still solve, get neglected as being long shots that are unlikely to help program performance. It really is a matter of perceived return for time invested. Many chess programmers choose short-term gains at the expense of long-term success. Perhaps the best computer chess researchers are the ones with no interest in writing their own competitive programs.

Since the benefits of winning are high, there is little incentive for quickly disseminating new research results. Many chess programmers hold back on new ideas to ensure that their programs maintain a competitive edge. Consequently, each team of chess programmers may have to discover ideas that are well known to other teams. This can only slow down progress in computer chess.

3.4. Experimental Computer Science

One notable aspect of computer chess work, not typical of most research in computer science, is the large experimental component. For example, demonstrating the effectiveness of a new search idea may require the searching of hundreds or thousands of trees to acquire enough data to be able to pass judgment on the idea. The probabilistic nature of search trees means that experimentation will always be a large part of the study of search algorithms.

While the importance of experimentation in developing chess programs is not to be underestimated, some programmers rely too heavily on the experimental results without necessarily understanding the underlying theory. One has only to look at an experimental domain such as physics to realize that experiments are usually only conducted in order to test the strength of a theory. In computer chess, it is often the other way around; one does experiments and only then tries to understand what is happening.

A major problem is the lack of a theory of computer chess. Yes, there is lots of theory on search algorithms ($\alpha\beta$ in particular), although only a small portion is relevant for a practitioner. However, there is no theory of chess knowledge and its interactions with the search. Chess

Monty Newborn asserts that this is the longest ongoing public scientific experiment programmers alter their program and have no means for understanding what the potential consequences of the change are. Instead, then usually conduct experiments to see if the change is beneficial. Without fully understanding the subtleties of the problem we are trying to solve, it is no wonder that much work is wasted on blind experimentation.

3.5. Coping with Error

Chess programs must cope with at least three types of error. First, there is error in terminal node evaluations, implying incorrect or inadequate chess knowledge. Minimax search has the insidious property of hiding such errors. Second, there are errors in decisions as to where to search in the tree. Fixed depth $\alpha\beta$ does not have this type of problem (which is one of the reasons why it is so popular), but selective search does. Third, there are search efficiency errors (such as move ordering) that affect the rate of search, but usually not its outcome.

Of the three types of search errors, the first two are the most serious since they can influence the quality of the move chosen by the program. Do we have any measures of how serious these problems are? It has been said that chess programs are the world's most sophisticated random number generators. How can we argue with that statement when we have no idea what the significance of the results produced are? A numerical analyst could not judge the quality of a numerical computation without knowing some bound on the error. Why should chess programs be exempt? How often have chess programs made the right move (our sole measure of "correctness") for the wrong reason? How often has a chess program correctly solved a problem, only to have a bug fix result in the program being unable to solve the problem?

Work in error analysis has been sadly lacking. Error estimates for chess programs are based solely on empirical observations. There are no techniques for measuring or compensating for errors. Many numerical analysts make their livelihood from precise mathematical analysis of error properties. The result is faster, more stable numerical algorithms. Is the same not possible for computer chess?

How much error are we willing to live with in a chess program? One cannot answer that question without knowing how much error we are currently dealing with. Human chess players live with error; the less error, the better the player. Even the human World Champion is not immune from trivial mistakes. Error is not to be feared; only to be kept under control. Chess programs should not necessarily be striving for perfection.

3.6. But Is It Useful?

Is computer chess, as currently practiced, AI? Maybe. It really depends on your definition of AI; there are some people who think that what most AI researchers do is not AI [20]. Many AI researchers may be guilty of the same errors as computer chess people are: building large systems that are capable of performing well on one problem but whose generality is questionable. Unfortunately, most people working in computer chess have done a poor job of selling their ideas to a community other than their own. There are more important research problems in the world than making computers play chess. Chess researchers cannot live in an ivory tower and solve a problem without concern over the generality of the ideas. A lot of good work has been done in computer chess and does have applicability in other areas. Regrettably, most of this work remains hidden from view and not generalizable.

Theorem proving is a problem that has a lot in common with computer chess. Both can build large trees to solve problems, but theorem proving is without as powerful a tool as alpha-beta cut-offs. Monty Newborn has successfully applied his experience with searching chess trees to theorem proving [18]. Many of the algorithmic ideas first developed in computer chess are relevant to other tree search based problems, and it is surprising that no one made the connection earlier. Why is theorem proving an acceptable research problem and computer chess is not? One reason is that people can see where to use theorem provers, whereas they can only see computer chess as game playing. Theorem provers are used as tools to solve what are perceived as more important problems.

The stigma of mere "game playing" must be overcome. Part of the problem lies with the commercial side of computer chess; some equate research in computer chess with sales of chess machines. Computer chess was once seen as, and still is, a restricted domain ideal for artificial intelligence research. However, the emphasis on performance, best epitomized by the annual tournaments and the competition for commercial sales, has clouded the view of many.

4. Engineering Credibility

There is no doubt that chess programs can be large, complex programs. Most chess programmers are proud of their accomplishment and the programming skills that made it possible. Many will argue that building a chess program is more an engineering feat than one of scientific interest. So how does the work on computer chess measure up to engineering standards?

4.1. Building Without Tools

David Slate and Larry Atkin, writing on the difficulties of developing their chess program, state that:

> Our problem is that the programming tools we are presently using are not adequate to the task... This lack of programming tools has plagued the whole field of computer chess. With the proper tool one might accomplish in a day a job that had been put off for years. Although serious efforts are underway to overcome this deficiency, the number of people working on this is surprisingly small, and their research is still in its infancy. ([28], page 116)

In the intervening 14 years since that was written, little, if anything, has been done to improve the number and quality of tools used to build, debug, and test chess programs.

Why is there so much effort spent on building chess programs and not much on tools to build chess programs? One answer, again, is the performance demands. Programmers are always striving for immediate gains, lacking long-term insight. Another, more subtle, performance aspect is what the tools produce. A tool that allows one to succinctly describe chess knowledge may be capable of expressing all sorts of wonderful things in a human readable form. Unfortunately, these tools will produce code that is not going to be as fast as hand-written code. The poorer the code, the slower the program runs, and (painfully) the worse the program performance. Can one afford the performance cost? The dilemma is akin to that encountered by ardent assembler programmers who argued that compilers would never succeed since the quality of code produced by them was inefficient. History has proven them wrong. With the increasing speed of today's processors, the "overhead" of using tools is small compared to the cost of programmer time.

It is a shame better tools are not available. As a result, a lot of talented people spend a great deal of time wrestling with basic chess program problems that might be more easily solved with the right tool.

4.2. If It Works Once, It Works

One of the most serious problems faced by chess programmers is testing. Since there are roughly 10^{40} positions, it is not possible to exhaustively test everything. Given that adequately testing a chess program is a difficult task, many chess programmers take the easy way out and disregard the problem. Too many programs are developed by testing things once, or perhaps (extravagantly) a few times, and concluding that it works. Alternatively, there is the so-called "tournament experience"; the more games the program plays, the better the programmer feels about his program. This does not engender a high degree of confidence!

The writers of production quality compilers are faced with a similar problem but, in their case, the number of possible test programs is not finite. The compiler writing industry has solved this problem by establishing large collections of test programs for each language. Although such a test suite can never certify that the compiler is bug free, a program that successfully handles the test suite must be reasonably reliable.

There are few test suites available for chess programs and there are only two in common use. The 300 positions from Reinfeld's *Win At Chess* book are exclusively tactical problems [21], while the Bratko-Kopec set consists of 24 positions that test a program's ability at tactics and at finding pawn levers [15]. A test suite of only 324 positions is inadequate to properly cover a problem space of 10^{40} positions. Moreover, both problem sets have serious deficiencies. The *Win At Chess* set is such that most programs easily solve better than 80% of the problems. Of what significance is a program that solves 2% more problems than another? At least the Bratko-Kopec set is valuable because of the 12 positional problems. However, since this is an oft-used benchmark, many programs have been tuned to perform well on this set.

Why has the computer chess community not started an effort at assimilating test data used by individual programmers and constructing a useful test suite? Perhaps 10,000 problems? Maybe that is too small. Without such a test suite or better software testing methods, many chess programmer's time is needlessly wasted.

4.3. Where Should One Place One's Efforts?

The discovery of a correlation between speed and performance may have set computer chess research back a decade. The metrics of computer speed and nodes per second have been regularly used as an approximate measure of the strength of a program. As a result, programmers often devote enormous amounts of time and resources towards optimizing their programs: finding tricks to make routines run slightly faster, or hand optimizing assembler code. The net gains, usually only a small percentage speedup, cannot have a significant impact on the program's chess ability.

This effort is largely wasted. Not only is there a high likelihood of introducing a bug (and consequently wasting additional effort trying to find it), but it is not clear that the effort is well spent. If an equivalent amount of effort were spent on finding better search algorithms or better ways to use knowledge, the potential gains would be significant. For example, better search algorithms offer a chess program orders of magnitude more in gains than trivial optimizations. After all, if computer chess is a research project then time is best spent on research, not hacking.

5. Conclusions

It is perhaps unfortunate that alpha-beta came along so early in computer chess history. It has a lot of nice properties that would make it an ideal search algorithm for computer chess, but for the exponential growth in the search tree. Too much effort has been concentrated on this algorithm, as though it were the chess programmer's panacea. Now, more so than ever before, computer chess must break out of the fixed-depth, alpha-beta search frame of mind and consider alternatives. Brute force alpha-beta may have done as much damage to the progress and credibility of computer chess research as it has given chess programs high ratings.

Should an artificial intelligence system solve a problem using methods similar to the human approach (the cognitive view) or should it just solve the problem using any approach (the engineering view) [20]? The prevailing attitude is that it is not AI unless the former approach is used. Computer chess is an important problem worth solving, regardless of the methods used. Making computers play chess is both engineering and science, and one should draw on all resources for solving the problem. Perhaps this is the path that AI will eventually also have to follow.

In retrospect, it is not surprising that computer chess has fallen from grace. Since the research work has strayed from the main stream and has neglected many of the mandatory scientific aspects of our research, it is only natural that computer chess should be shunned by our colleagues. Of course, there is still time to make amends!

Acknowledgments

We greatly appreciated the constructive criticism from Greg Rawlins, Randy Goebel and Murray Campbell. This work was supported in part by the Canadian Natural Sciences and Engineering Research Council and the University of Alberta's Central Research Fund.

References

1. G. M. Adelson-Velsky, V. L. Arlazarov and M. V. Donskoy, Some Methods of Controlling the Tree Search in Chess Programs, *Artificial Intelligence 6,* 4 (1975), 361-371.
2. G. M. Adelson-Velsky, V. L. Arlazarov and M. V. Donskoy, *Algorithms for Games*, Springer-Verlag, New York, 1988.
3. T. Anantharaman, M. Campbell and F. H. Hsu, Singular Extensions: Adding Selectivity to Brute-Force Searching, *Artificial Intelligence,* 1989. In press. Also available in *Journal of the International Computer Chess Association 11,* 4(1988), 135-143 and in *AAAI Spring Symposium Proceedings 1988,* pages 8-13.
4. D. Beal, Null Moves, in *Advances in Computer Chess V*, D. Beal (ed.), Elsevier Science Publishers, Amsterdam, , 65-80.
5. H. Berliner, Some Necessary Conditions for a Master Chess Program, *International Joint Conference on Artificial Intelligence*, 1973, 77-85.
6. H. J. Berliner, The B* Tree Search Algorithm: A Best First Proof Procedure, *Artificial Intelligence 12*, 1 (1979), 23-40.
7. H. Berliner and C. Ebeling, Pattern Knowledge and Search: The SUPREM Architecture, *Artificial Intelligence 38,* 2 (1989), 161-198.
8. M. S. Campbell and T. A. Marsland, A Comparison of Minimax Tree Search Algorithms, *Artificial Intelligence 20,* 4 (1983), 347-367.
9. A. D. De Groot, *Thought and Choice in Chess*, Mouton, The Hague, 1965.
10. C. Ebeling, All the Right Moves: A VLSI Architecture for Chess, Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, 1987.
11. G. Goetsch and M. Campbell, Experiments with the Null Move in Chess, *AAAI Spring Symposium*, 1988, 14-18.
12. D. Hartmann, How To Extract Relevant Knowledge From Grandmaster Games, Part 2, *Journal of the International Computer Chess Association 10,* 2 (1987), 78-90.
13. D. Hartmann, How To Extract Relevant Knowledge From Grandmaster Games, Part 1, *Journal of the International Computer Chess Association 10,* 1 (1987), 14-36.
14. F. H. Hsu, A 2 Million Moves/Sec CMOS Single Chip Chess Move Generator, *International Solid State Circuits Conference Digest of Technical Papers,* 1987, 278.
15. D. Kopec and I. Bratko, The Bratko-Kopec Experiment: A Comparison of Human and Computer Performance in Chess, in *Advances in Computer Chess 3,* M. R. B. Clarke (ed.), Pergamon Press, 1982, 57-72.
16. T. A. Marsland, Evaluation-Function Factors, *Journal of the International Computer Chess Association 8,* 2 (1985), 47-57.
17. D. A. McAllester, Conspiracy Numbers for Min-Max Search, *Artificial Intelligence 35,* 3 (1988), 287-310.
18. M. Newborn, private communication, *Moscow, USSR,* 1988.
19. A. J. Palay, The B* Tree Search Algorithm - New Results, *Artificial Intelligence 19,* 2 (1982), 145-163.
20. D. Parnas, Why Engineers Should Not Use Artificial Intelligence, *INFOR Special Issue on Intelligence Integration 26,* 4 (1988), 234-246.
21. F. Reinfeld, *Win At Chess*, Dover, 1945.

22. R. L. Rivest, Game Tree Searching by Min/Max Approximation, *Artificial Intelligence 34*, 1 (1988), 77-96.

23. J. Schaeffer and T. A. Marsland, The Utility of Expert Knowledge, *International Joint Conference on Artificial Intelligence*, 1985, 585-587.

24. J. Schaeffer, *Experiments in Search and Knowledge*, Ph.D. thesis, Department of Computer Science, University of Waterloo, 1986.

25. J. Schaeffer, Speculative Computing, *Journal of the International Computer Chess Association 10*, 3 (1987), 118-124.

26. J. Schaeffer, Conspiracy Numbers, *Artificial Intelligence*, 1989. In press. Also in *Advances in Computer Chess V*, D. Beal (ed.), Elsevier Science Publishers, pp. 199-218, 1989.

27. G. Schruffer, Minimax-Suchen: Kosten, Qualitat und Algorithmen, Ph.D. thesis, Technical University Braunschweig, 1988.

28. D. J. Slate and L. R. Atkin, Chess 4.5 - The Northwestern University Chess Program, in *Chess Skill in Man and Machine*, P. W. Frey (ed.), Springer-Verlag, New York, 1977, 82-118.