Playing Computer Chess in the Human Style: is it Feasible?

Anastasia Afanaseva and Sergey Afonin

Moscow State University, Russian Federation

Abstract. For extended period of time it was widely accepted that computers can not play chess in the human style. Top level chess programs follow the game tree exploration approach and analyze huge number of positions and possible moves. To our best knowledge, no successful realization of a program following an alternative approach, such as the one introduced by M.M. Botvinnik, was ever conducted. In this paper we analyze the possibility of such implementation. Number of optimization problems are identified and corresponding algorithms are proposed.

Keywords: Computer chess, chunking, optimization problems.

1 Introduction

For more than over decade computer chess programs outperform top-level human players. This advance was determined by the significant improvements of the computer hardware. The increased computational power and capacity of fast internal memory make it possible to run brute-force algorithms with appropriate pruning strategies in a reasonable running time. A remarkable example of such advances is the tablebases for 7-pieces endgames, computed at Moscow State University on "Lomonosov" supercomputer in 2012. This 140TB-database includes funny examples such as proved mate in 545 moves[1]. Nevertheless, brute-force algorithms differ from the humans' cognitive processes and one of the most important goal of research on computer chess, i.e. modeling of humans behavior during complex problems solving, was not achieved in general. Such modeling requires a deep knowledge from neurophysiology, psychology, computer science and many other areas. Ongoing research on such topics, e.g. [4, 8], indicates importance of the problem.

In this paper we consider a quite dated approach, frequently referenced to as "the M.M. Botvinnik's method", aimed to model cognitive processes of a human chess player. This choice is personally motivated because one of the authors was a member of the last Botvinnik's team. The key notion in this approach is a *chain* — a piece's trajectory with all relevant supporting and defending moves. Let us illustrate it by an example. Suppose, that White is going to move a Pawn from a5 to a8 by a sequence of moves a5–a6–a7–a8Q. If some square, say a7, is occupied by a White piece, then the original plan requires supporting moves,

[1] `http://chessok.com/?page_id=27966`

say Na7-c8. If this move is not possible (or not acceptable by some reason), extra supporting moves might be required to make it happens. Similarly, Black is trying to defend either by blocking the trajectory or by taking some squares under control, so Bg6-e4-a8 might also be in that chain. In turn, White might attack e4 in order to invalidate Black's defense, and so on.

In some sense a chain represents the plan of playing. Starting with an idea one can find all obstacles preventing it's realization, and consider possible ways of elimination of such obstacles. Many optimization problems fit this terminology and the general approach might be useful to solve over problems. Attempts for such generalization were made, e.g. [2, 3, 9], but in our opinion generalizations are premature before the implementation of a master level chess program. Any generalization introduces extra level of complexity because one should explain rules, goals and heuristics in terms of a general framework which is not always possible.

The chain construction is a complex problem. One piece might have several chains in general. When all chains (for both players) are constructed, f consistent representation of the current position should be constructed by merging the chains. At this stage one may discover that a move in one chain is also a move in another, e.g. fork, or using a piece for supporting a chain is not possible because this piece required in a more valuable chain. Having a consistent representation one can determine *candidate moves* and run their validation by moving pieces on a virtual board. In contrast to game tree exploration procedure dominating in the modern chess programs, chains are constructed using mostly static position, the set of candidate moves are expected to be very small, and the total number of positions visited on validation stage shall not exceed one hundred positions. According to M.M. Botvinnik, the described procedure reflects human player's behavior.

The possibility of successful development of a human-like computer chess program in general, and the one following Botvinnik's method in particular, was widely criticized, e.g. [1]. According to our best knowledge, such programs were never developed. This is certainly so for the Botvinnik's method. Our goal is to answer the following questions:

- Can this approach, stated in the 70s, benefits from several orders of magnitude increase of memory capacity and CPU performance, or it is broken by itself, and
- Is it possible to validate the approach using modern eye-tracking techniques?

The first question is mostly related to programming issues. The eye-tracking research depends of the successful implementation of the chain construction algorithm. By itself, chain construction is not sufficient for playing. If a chain construction algorithm will be implemented, then eye-tracking methods could be used for the general idea validation. Recorded eye movements could be compared to the computed chains.

In this paper we discuss optimization problems and decision questions that should be answered in order to implement the chain construction algorithm.

2 Brief Description of the Approach

In this section we briefly describe the approach followed by this paper. It is mostly based on Botvinnik's ideas on decomposition of position into smaller parts, similarly to chunking theory of de Groot, Simon and Chase [5, 6]. It was originally expected that this approach is not only the basis for a chess program, but a tool for solving a wider class of optimization problems as well. For example, this approach was applied to some economics problem, leading to a well known economics' model. Nevertheless, we will use chess terminology in the reset of the paper.

As we have mentioned early, the key notion of the approach is a chain. A chain is associated to a piece and determined by piece's trajectory from its current position on the board to a desired square. This trajectory is called *subchain-0*. If a square on the trajectory in not "passable" (either blocked by another piece, or exchange value on the square is not acceptable) then supporting subchain-1 is needed. Subchains-2 are used in support for subchains-1, and so forth.

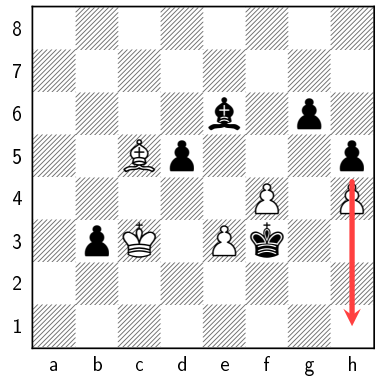
At the very top level the overall algorithm can be decomposed into following stages:

- for every piece find appropriate *goals* that should be achieved (e.g., a square to be reached, or opponent's piece to be captured);
- for every goal construct *subchain-0* — a *trajectory* that might be used by the piece for achieving its goal;
- validate that subchain-0 is realizable, or extend subchain-0 by supporting subchains-1 (e.g. take away other pieces from the subchain-0's trajectory);
- construct all the necessary supporting subchains of higher orders (subchain-2 is a supporting subchain for a subchain-1);
- merge all the chains into consistent representation of the current position;
- find candidate moves;
- validate candidate moves by tree searching.

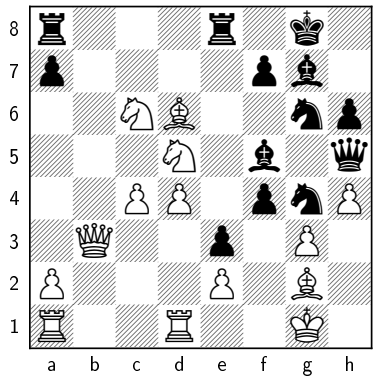
Such scheme assumes that chains are constructed independently and then merged together, so the problem divided into two parts: chain construction, and chains merging. Chains are used for position and piece evaluation. Roughly speaking, a valuable piece participates in large number of valuable chains, and there are many valuable attacking chains in a good position.

Many aspects should be taken into consideration. It is difficult to find a real-life example that can be described using one chain. In the rest of this section we describe questions that should be addressed by an algorithm for construction position representation.

Large number of chains. In order to be able to find a reasonable plan of playing, subchains-0 should be sufficiently long. If a goal is specified by the target square, the simplest form of a goal, then the set of subchain-0 candidates consists of all trajectories of the piece on the empty board, starting from its current position and ending on the target square. The *horizon*, counted in plies, may vary depending on piece's kind or stage of the game. Nevertheless, the total number of chains, i.e. trajectories passed some filtering condition, could be large.



(a) Kotov – Botvinnik, 1955. 59. . . . ?



(b) Ivanchuk – Yusupov, 1991. 23. . . . ?

High variability of chains. Every chain contains the only subchain-0. If subchains of higher orders are used in that chain, then one can expect high ambiguity between subchains addressing the same problem. For example, if the attacker need to take some square on the subchain's-0 trajectory under control, then it could be done in many ways, in general. Number of variations growths rapidly with the maximum subchain's order. In case of independent construction of chains all possibilities should be preserved or enumerated later when additional information from other chains become available.

Timing. Defending subchains are time-constrained. It is not only required to defend against opponent's subchain, but do it in time. The more plies required for the attacker to reach a square, the longer defending subchains are possible. One can easily compute a raw estimation of number plies required to reach given square on the subchain's trajectory, but this value might change depending on modification of subchains associated with preceding squares.

Functional dependencies between chains. Clearly, one piece may participate in many chains. A fork is the most natural example. If such overlapping accrues then a move in one chain is also a move in another one. As a result timing constraints in one of overlapping chains change.

Dynamic nature of entities' values. In order to compute the value of a chain, or to decide whether or not a square is "passable" it is required to compute the exchange value that depends on values of affected pieces. Piece value is a function of its chains. Once pieces get their new values some chains might change their structure — a piece intended to participate in the chain could be too valuable for that.

Let us consider position on Fig. 1(a). This position was always among Botvinnik's tests. The winning plan for Black is h5-h1. Pawn can not move on h4, so a

supporting subchain-1 Kf3-g3-h4 or Kf3-g4-h4 required. (Actually, Kf3-g4-h4 is not suffice, as mentioned below). Immediate attack on h4 is not successful because of White's defense Bc5-e7 (subchain-2). White's defense can be eliminated by g6-g5, if played before Bc5-e7. Complete human analysis is as follows.

1... g5!! 2. f×g5 d4 3. e×d4 [3. ♗×d4 ♔g3 4. g6 ♔×h4 5. ♔d2 ♔h3!! 6. ♗f6 h4 7. ♔e2 ♔g2] **3... ♔g3** [3... ♔g4 4. d5 ♗×d5 5. ♗f2] **4. ♗a3** [4. g6 ♔×h4 5. g4 ♔g4, or 4. ♗e7 ♔×h4 5. g6+ ♔g4] **4... ♔×h4 5. ♔d3 ♔×g5 6. ♔e4 h4 7. ♔f3 ♗d5+** 0–1

Note, that **2... d4** considered as an obvious move: Black's Bishop should a) protects Pawn on b3 in order to keep the threat of b3-b2-b1Q, and b) controls g8 square, making White's plan g5-g6-g7-g8 impossible. From the computer's perspective this move is not self-evident at all. One should discover that Pawn on b3 is a crucial piece: White's King becomes overloaded in some variations, trying to defend against h5-h1 and b3-b1 simultaneously.

This example demonstrates that even a simple position with less than a dozen pieces could provides plenty of variations with complex correlations between different chains (plans). In more complex positions the number of possibilities could be enormous. For example, in the position depicted on Fig. 1(b), there are about 120 000 subchain-0 candidates, so importance of performance issues can not be underestimated.

3 Chain Construction

We consider the chain mostly as a data structure for storing possible variations, trying to keep construction procedure as simple as possible. A chain is not required to be absolutely precise. All non-trivial decisions and improvements are delayed for later stages. Nevertheless, chain construction is far from trivial. A chain should represents best variation using given values for pieces. In this section we discuss issues of this procedure.

The chain is a recursive structure containing trajectory of the main piece (subchain-0) and, possibly, a number of subchains associated with squares on this trajectory. Given a subchain-0 trajectory, e.g. Qh5-h2-f3 on Fig. 1(b), we first build the extended trajectory by including all the intermediate squares (h4, h3, g2). Squares of the extended trajectory divided into two classes: stop-squares, where the piece intended to stop, and internal squares. We call a stop-square *passable* if it is not occupied by another piece of the same color and the exchange value on that square is positive. An internal-field is passable if it is free, and, in case of castling, not under control.

For each non-passable field supporting subchains should be constructed. If the square is blocked by a piece of the same color as the chain's piece, then blocking piece should escape somewhere first. In case of unfavorable exchange value (let us recall that we assume that pieces' values depend on chains they are involved) following possibilities exist:

- take the square under control by a piece that can improve exchange value;

– exclude opponent's piece from the exchange by pinning it, or blocking its trajectory;
– capture opponent's piece.

It is possible that only a combination of actions give positive exchange value on the square. So it is reasonable to consider bunch of subchains — a set of supporting chains that make sense only if realized together. Clearly, a bunch may be a singleton set. As chains are constructed independently from each other it is not possible to decide at this point which bunch is the best. A chain contains locally optimal choice, i.e. the bunch that yields best result in this chain. Other possibilities are recorded, probably in less details. If the selected bunch will be rejected on a later stage as a result of chains merging then the chain should be updated and previously recorder subchain will be analyzed in more details, i.e. subchains of higher orders will be constructed.

Subchains are constructed for both colors. There is a crucial difference between supporting and defending subchains. Defending subchains are time-limited. For every square on subchain-0 trajectory one can estimate number of moves required for the main piece to reach that square, including moves in subchains associated with preceding squares. Defending subchain should not require more time. Non-trivial timing constraints frequently appear in the endgame. for example, It should be mentioned time constraints are fuzzy. "Slow" defending subchains, that require more plies than available, should also be included into the chain: playing some moves in a slow subchain might be for free due to overlapping of that moves with more valuable chains. The most obvious example of such overlapping is a check. If slow subchains will be ruled out at chain construction stage, then they will be completely eliminated from consideration.

Every non-passable square introduces extra uncertainty and complexity to the chain, especially if there are several non-passable fields in one chain. Same pieces might participate in supporting subchains for different squares. For example, if a chain contains two non-passable squares, some piece might leave its position in order to support main chain on the first non-passable square. This movement should be reflected while searching for support subchains for the second square. Similar dependencies appear between subchains of different levels. For example, first move in subchain-0 might also supports subchain-1 associated with another square of subchain-0.

When a chain is constructed a correct order of moves should be determined. In position on Fig. 1(a) square h4 is non-passable and there are three possible supporting subchains, namely Kf3-g3-h4, Kf3-g4-h4, and g6-g5. Black can play any of the three moves, while only one wins.

Summing up:

– chains are constructed independently, regardless of other threats presented in current position;
– each chain contains selected set of subchains that is optimal if this chain considered isolated;
– for each bunch of supporting and defending subchains a chain contains all possible variations, i.e. other chains;

– for each chain possible sequence of moves, or a set of equivalent sequences, is defined.

4 Consistent Position Representation

Chains are more or less "local". Once all the chains are constructed one can estimate dynamic values for pieces. Change of pieces' values affects the result of chain construction procedure, which in turn influences piece values. It is unlikely that simple iterative procedure would converge to a certain if pieces' values will change drastically.

Regardless of pieces' values convergence there are many cases where chains should be altered. The following types of chains interaction are among most frequent cases.

– Fork, or subchains overlapping. One move reveals two or more threats simultaneously. As we have mentioned early, a fork might appear between subchains on any level. It is possible that one of the overlapping subchain does not belong to it's chain main (selected by chain construction procedure) variation.
– Attraction. Forcing a piece to an unfavorable square.
– Piece overloading, zugzwang. For every chain, while considered independently, the goal is unreachable, but there is at least one piece that appears in both chains.

5 Conclusion and Future Work

In this paper the Botvinnik's approach to computer chess playing was considered. We identified some questions that have to be answered in order to implement one of the most important part – the chain construction procedure. It is absolutely unclear at the moment whether or not consistent position representation can be constructed from individual chain in reasonable amount of time.

Direction of future work includes implementation of the described procedure and validation of the approach by means of eye tracking. Modern wearable glasses are accurate enough to track eye movements over a chess board without inconveniences to the subject. Provided that a working implementation for chain construction procedure exists (which is not enough for chess-playing program) it is possible to compare actual eye movement with the chains computed by the program.

The method considered in this paper is a kind of an expert system. Development of an expert system includes knowledge acquisition stage. Thus, a bulk collection of positions with appropriate markup is needed. The current work lacks of real-life examples. As a more distant perspective, in case of successful implementation of a working program, one may consider application of machine learning techniques [7, 10] for parameters estimation.

References

1. Berliner, H. Playing computer-chess in the human style. *ICCA JOURNAL 16*, 3 (1993), 176–182.
2. Botvinnik, M. M. *Computers, Chess and Long-Range Planning.* Springer US, 1970.
3. Botvinnik, M. M. *Computers in Chess: Solving Inexact Search Problems.* Springer Science + Business Media, 1984.
4. Campitelli, G., Gobet, F., Head, K., Buckley, M., and Parker, A. Brain localization of memory chunks in chessplayers. *Int J Neurosci 117*, 12 (jan 2007), 1641–1659.
5. Chase, W. G., and Simon, H. A. Perception in chess. *Cognitive psychology 4*, 1 (1973), 55–81.
6. degroot, A. D. *Thought and Choice in Chess.* Mouton, The Hague, The Netherlands, 1965.
7. Fogel, D., Hays, T., Hahn, S., and Quon, J. A self-learning evolutionary chess program. *Proceedings of the IEEE 92*, 12 (Dec 2004), 1947–1954.
8. Reingold, E. M., and Charness, N. Perception in chess: Evidence from eye movements. *Cognitive processes in eye guidance* (2005), 325–354.
9. Stilman, B. *Linguistic Geometry: From Search to Construction.* Kluwer Academic Publishers, Norwell, MA, USA, 2000.
10. Wirth, C., and Furnkranz, J. On learning from game annotations. *Computational Intelligence and AI in Games, IEEE Transactions on 7*, 3 (Sept 2015), 304–316.