Influence of Search Depth on Position Evaluation

Matej Guid and Ivan Bratko

Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia

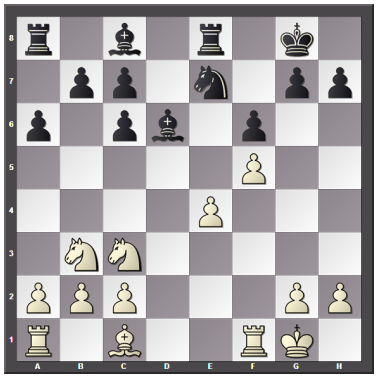
Abstract. By using a well-known chess program and a large data set of chess positions from real games we demonstrate empirically that with increasing search depth backed-up evaluations of won positions tend to increase, while backed-up evaluations of lost positions tend to decrease. We show three implications of this phenomenon in practice and in the theory of computer game playing. First, we show that heuristic evaluations obtained by searching to different search depths are not directly comparable. Second, we show that fewer decision changes with deeper search are a direct consequence of this property of heuristic evaluation functions. Third, we demonstrate that knowing this property may be used to develop a method for detecting fortresses in chess, which is an unsolved task in computer chess.

1 Introduction

The purpose of a heuristic evaluation function is to guide the game-tree search. Heuristic evaluation functions have to enable a program to find a direction of play towards a win, not only to maintain a won position. Backed-up (i.e., *negamaxed* or *minimaxed*) heuristic values should in some way also reflect the progress towards the end of the game, and should therefore change as the search depth increases. Given a won position, if backed-up heuristic values remained the same with an increasing level of search, this would just ensure that the value "win" is maintained, without any guarantee of eventually winning.

Heuristic evaluations are supposed to reflect a goodness of a particular position. Actually, what exactly this value means was never strictly defined. Various authors viewed this value as a position's "worth", "merit", "strength", "quality", or "promise" [1]. It is well known that searching deeper generally leads to stronger play [2]. A common belief is that searching deeper leads to better approximations of the value of the root node of the search tree (after minimaxing) to the unknown "true" value of the position at the same root node. That is, it is typically assumed that searching deeper results in a more accurate evaluation in terms of approaching the unknown "true" value of the root-node position.

It is generally accepted that a "perfect" heuristic evaluation function would statically (i.e., without any search) assign the "true" value to the position in question and that searching deeper would not affect this evaluation. Since these "true" values are not known, it is accepted that they have to be approximated heuristically. For example, Luštrek et al. [3] propose a model that uses real



Program	Evaluation
ChessMaster 10	0.15
Crafty 19.19	0.20
Crafty 20.14	0.08
Deep Shredder 10	-0.35
Deep Shredder 11	0.00
Fritz 6	-0.19
Fritz 11	0.07
Rybka 2.2n2	-0.01
Rybka 3	-0.26
Zappa 1.1	0.13

Fig. 1. Lasker-Capablanca, St. Petersburg 1914, position after white's 12^{th} move. The table on the right shows backed-up heuristic evaluations obtained by various chess programs, when evaluating the diagrammed chess position using 12-ply search.

numbers for "both *true* and heuristic values." In the proposed model, "static heuristic values are obtained by corrupting the true values at depth d with random errors representing the fallibility of the heuristic evaluation function."

Chess programs usually use heuristic evaluations where the advantage of one unit represents material advantage of one pawn (or equivalent by means of accumulated pluses of positional features). Chess players got used to computer evaluations and even widely accepted the *centipawn* as the unit of measure used in chess as a measure of advantage, a centipawn being equal to 1/100 of a pawn. They often use computer evaluations to express position evaluation (e.g., "+1.15 according to Houdini"), completely ignoring the information about the depth at which the evaluation was obtained (at least as long as the search was "deep enough"). As Fig. 1 clearly shows, different programs assign different evaluations to a given position, even when using the same depth of search. This may lead to the misleading impression that the programs try to approximate some unknown *"true"* heuristic value of the position being evaluated.

Direction oriented play (in which the winning player strives to *increase* the advantage, as opposed to advantage-preserving play) is a property of every successful program in all typical games where heuristic search is used. Therefore it seems reasonable to expect this property to be reflected somehow in the programs' heuristic evaluations. In this paper, we demonstrate empirically by using a well-known chess program that with increasing search depth backed-up evaluations of won positions (from the white player's perspective; in the theoretical sense: white wins if both sides play optimally) will on average be increasing, and that evaluations of lost positions will on average be decreasing. More importantly, we discuss three possible impacts of this property of heuristic evaluation functions on game playing, and point out that heuristic evaluations obtained by searching to different search depths are not directly comparable, in contrast to what is generally assumed both in literature and in practical applications.

We believe that there has been no study of this property of heuristic evaluation functions and its impacts on game playing. When giving arguments in support of look-ahead, Pearl [4] explains the notion of *visibility*, which says that since the outcome of the game is more apparent near its end, nodes at deeper levels of the game-tree will be more accurately evaluated and choices based on such evaluations should be more reliable. Scheucher and Kaindl [5] advocate that a heuristic evaluation function should be multivalued to be effective and that game-theoretic heuristic values alone would not produce desirable results. Luštrek et al. note that multiple values make it possible to maintain a direction of play towards the final goal. Gomboc et al. [6] show that it suffices for evaluation functions to tell only whether a certain position *is* better than some other position, and not *how much* better. Donkers et al. [7] examine three types of evaluation functions: *predictive*, *probability estimating*, and *profitability estimating* evaluation functions, and investigate how evaluation functions can be compared to each other. Several authors have studied properties of heuristic evaluation functions, particularly with respect to the propagation of static heuristic errors through minimaxing (see [8] for an overview). Various papers focused on the so-called *go-deep* experiments [9–17], as we did in the present study. None of the related work focused on (1) the increasing (or decreasing) backed-up evaluations when searching deeper and (2) the impact of this phenomenon on the theory and practice of game playing.

2 Experiment

In order to study deep-search behavior of chess programs, particularly with respect to changes of heuristic evaluations with increasing search depth, we conducted *go-deep* experiments on a large number of chess positions.

2.1 The Experimental Settings

The chess program Rybka was used in the experiments. The program was used to analyze more than 40,000 positions from real games in a *go-deep* fashion: each position occurring in these games after move 12 was searched to a fixed depth ranging from 2 to 12 plies. Search to depth d means d ply search extended with *quiescence search* to ensure stable static evaluations.

We defined six different groups of positions based on the backed-up heuristic evaluations obtained at the deepest search depth available, as given in Table 1. Evaluations by computer chess programs are given by the following standard: the more positive evaluations mean a better position for White and the more negative evaluations mean a better position for Black, while evaluations around zero indicate an approximately equal position. In usual terms of chess players, the positions of Groups 1 and 6 could be labeled as positions with "decisive advantage," positions of Groups 2 and 5 with "large advantage," while Groups

Table 1. The number of positions in each of the six groups of data in three data sets. The groups were devised based on backed-up heuristic evaluation values obtained at a search depth of 12 plies using the chess program Rybka.

Group	1	2	3	4	5	6
Evaluation (x)	$x < -2$	$-2 \leq x < -1$	$-1 \leq x < 0$	$0 \leq x < 1$	$1 \leq x < 2$	$x \geq 2$
Positions	1,263	1,469	9,808	22,644	3,152	2,133

3 and 4 consist of positions regarded as approximately equal or with a "small advantage" at most.[1]

For each data set and for each group separately we observed the behavior of the backed-up evaluations with increasing depth of search.

2.2 Experimental Results

The comparison of backed-up evaluations obtained at adjacent search depths shows different behavior for positions of each group of our test data. The graph in Fig. 2 clearly shows that backed-up heuristic evaluations for Groups 1 and 6, where positions are likely to be within the zones of theoretical win and loss, on average increase with increasing search depth in positions with a decisive advantage for the white player (i.e., won positions), and decrease with increasing search depth in positions with a decisive advantage for the black player (i.e., lost positions from the perspective of the white player).

We further divided the data of won positions of Group 6 into four subsets, based on backed-up heuristic evaluation values obtained at search depth of 12. Like before, the highest search depth served as the best available approximation of the utility value of each analyzed position. The results are presented in Fig. 3.

The analysis of the results confirms our expectations that in won positions, the backed-up evaluations tend to increase with depth of search, while in lost positions searching deeper tends to produce decreasing backed-up evaluations.

We now investigate implications for the practice of game programming, of the fact that position evaluations tend to increase with search depth.

3 Searching to Variable Depths Revisited

Having in mind the demonstrated property of heuristic evaluation functions we could ask ourselves: are heuristic evaluations obtained by a search to different search depths really directly comparable? Consider a minimax-based program searching to variable search depths. Due to various types of *search extensions*

[1] Of course, this is only an approximation: The terms "decisive advantage", "large advantage", and "small advantage" are not strictly defined in the literature.

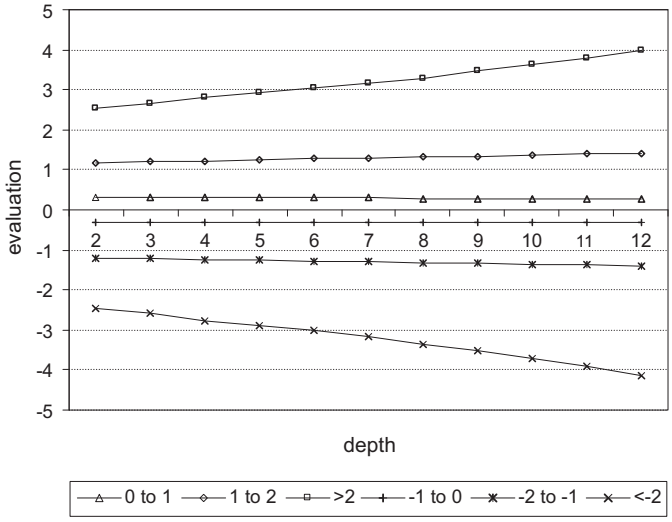


Fig. 2. Average backed-up evaluations at different search depths for each group of positions.

(searching more deeply from seemingly more promising parts of the search tree), state-of-the-art chess programs frequently conduct a search to different depths of search. Afterwards, the backed-up evaluations are being compared in such a way that the depth of search at which they were obtained is completely ignored.

However, in won positions, for example, backed-up heuristic values obtained from deeper searches should, on average, be expected to be higher due to the increasing backed-up evaluations. According to this observation, in such positions the following holds: if two moves result in approximately equal backed-up values, the one resulting from *shallower* search is more likely to lead to a better decision. Obviously, the depth at which the backed-up evaluation was obtained must be taken into account in order to perform relevant comparisons of backed-up heuristic evaluation values.

This point is illustrated by an example from a real game (see the diagram in Fig. 4). The right side of the figure shows the program's backed-up evaluations obtained at search depths in the range from 7 to 17 for two winning moves in the diagrammed position: **40.a5-a6** and **40.Nc7-e6**. The evaluations tend to increase with increasing depth of search, indicating that both moves lead to a win, assuming optimal play. The program finds the move **40.a5-a6** to be the best move, at any search depth. However, if the search depth used for evaluating the move **40.a5-a6** was less than 14 plies and the search for evaluating the move **40.Nc7-e6** was extended to 17 plies, Rybka would choose the inferior move. Indeed, the move **40.Nc7-e6** would yield the black player some practical chances of escaping into a drawn king and two knights versus king endgame. It is also

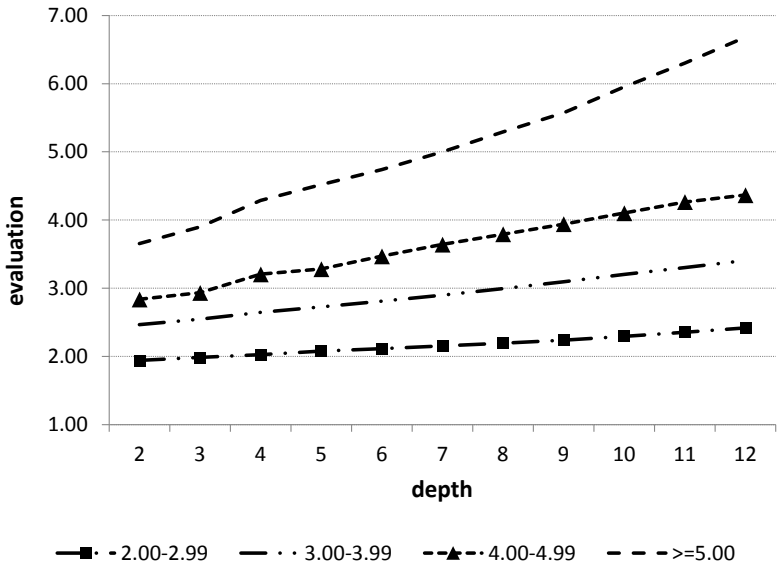


Fig. 3. The average backed-up evaluations at each depth for each subset of won positions of Group 6.

well known that exchanging pawns in won or nearly-won endgames, generally speaking, favors the weaker side [18].

The demonstrated property of heuristic evaluation functions becomes particularly important when using the computers to analyze huge numbers of chess moves. In [19], the games from the World Chess Championship matches were analyzed with a chess program in an attempt to assess objectively one aspect of the playing strength of chess players of different eras. The basic criterion for comparison among the players was, to what extent a player's moves deviate from the computer's moves. This type of computer analysis of chess games was repeated using the same program at different search depths [20] and using different programs at different search depths [21].

An important question when conducting a computer analysis of chess moves is: should the analysis be time-limit based or fixed-depth based? That is, should the machine spend the same amount of time for each move or should it rather perform the search to some fixed depth? The first option seems attractive, since approximately the same processing time is devoted to each move and also makes it possible to predict the cumulative time required for the analysis. However, it is highly questionable precisely due to the fact that searching to variable depth occurs in this case.

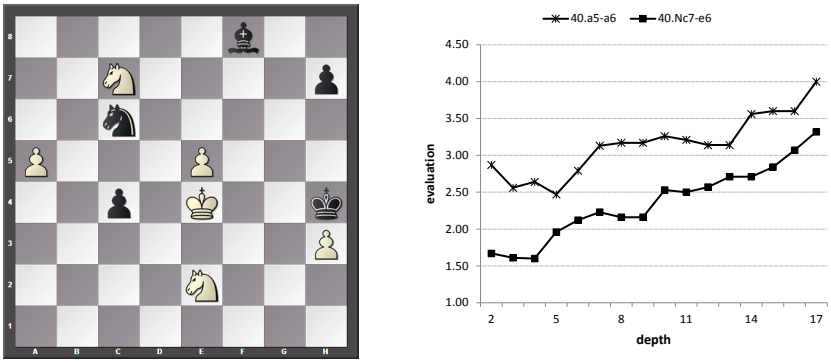


Fig. 4. Botvinnik-Smyslov, World Chess Championship match (game 16, position after black's 39^{th} move), Moscow 1954. White has two tempting continuations in this winning position: 40.a5-a6, keeping the white pawns on the board, and 40.Nc7-e6, attacking the black bishop. Both of them probably win, however, after 40.Nc7-e6 black can play 40...Nc6xa5!, and now if Black manages to sacrifice the knight for White's only remaining passed pawn, for example, after 41.Ne6xf8 (taking the bishop) 41...Na5-c6 42.Nf8xh7?? (taking the black pawn, but this is a mistake), Black saves himself with 42...Nc6xe5! 43.Ke4xe5 Kh4xh3, sacrificing the knight for achieving a drawn KNNKP endgame. In the game, Botvinnik played 40.a5-a6! and won five moves later. The right side of the figure shows the program's backed-up evaluations with increasing depth of search for the best two moves according to the program.

4 Decision Changes with Deeper Search

In [17], factors affecting diminishing returns for searching deeper were addressed. It was shown that in positions with a decisive advantage, the rates of the programs' decision change with increasing search depth differ from the ones in balanced positions. The authors demonstrated that changes of decisions of the program that manifest themselves in *go-deep* experiments depend on the utility values of positions that are the subject of such experiments. This type of experiment, also used in the present study, was introduced for determining the expectation of a new best move being discovered by searching one ply deeper, and were conducted by several authors [9–17]. The approach is based on Newborn's [22] discovery that the results of self-play experiments are closely correlated with the rate at which the best move changes from one iteration to the next. It was demonstrated that in positions with a decisive advantage, best moves according to the program change less frequently with increasing search depth than in balanced positions [17].

The property of heuristic evaluation functions on which we focused in this paper provides an explanation for this phenomenon. We observed that in positions with a decisive advantage, backed-up evaluations of better moves according to the program on average increase more rapidly than backed-up evaluations of less good moves. This phenomenon can be most clearly seen in Fig. 3. Since

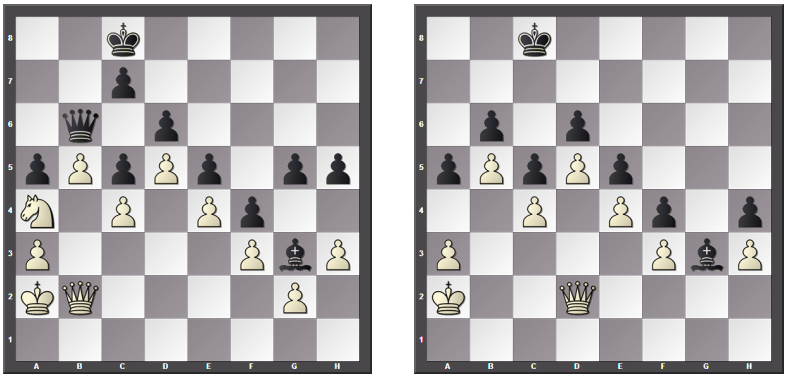


Fig. 5. In the left side diagram the white player is to move, and has a winning positional advantage. State-of-the-art chess programs without any exception choose the move 1.Na4xb6?? (the white knight takes the black queen), which leads to big material advantage. However, after 1...c7xb6 (the black pawn takes the white knight) 2.h3-h4 (otherwise Black plays 2...h5-h4 with a draw) 2...g5xh4 3.Qb2-d2 h4-h3! 4.g2xh3 h5-h4 Black's position (see the diagram on the right side) becomes an impregnable fortress and the win is no longer possible against adequate defence. Nevertheless, as GM Dvoretsky indicates, in the initial position the white player has a winning plan at disposal: 1.Qb2-d2! followed by, Ka2-b3, Na4-b2, Kb3-a4, Nb2-d3-c1-b3. By executing this plan White can gain the a5-pawn and win the game.

the backed-up evaluations of better moves on average increase more rapidly in positions with a decisive advantage, in such positions the differences between backed-up evaluations of candidates for the best move according to the program are likely to become bigger with increasing search depth. Thus the changes of programs' decisions with increasing search depth are less likely to occur.

5 Detecting Fortresses in Chess

In chess, the *fortress* is an endgame drawing technique in which the side behind in material sets up a zone of protection that the opponent cannot penetrate [18]. Current state-of-the-art programs typically fail to recognize fortresses and seem to claim winning advantage in such positions, although they are not able to actually achieve the win against adequate defence.

Detecting fortresses is an unsolved task in computer chess. The strongest chess programs are not able to detect fortresses such as the one shown in Fig. 5. Current state-of-the-art chess programs without an exception choose to take the black queen with the knight (1.Na4xb6), which leads to a big material advantage and to high evaluations that seemingly promise an easy win. However, after 1...c7xb6 (the black pawn takes the white knight) the backed-up evaluations, although staying high, cease to increase in further play. In fact, black position becomes an impregnable fortress and the win is no longer possible against adequate defence.

In [23], we demonstrated that due to a lack of increasing evaluations between successive depths that are otherwise expected in won positions, fortresses are detectable by using heuristic search to several successive search depths. Here we extend the original study by using a modern chess program and a far deeper search on an extended set of positions that are regarded as fortresses.

In the following experiment, we chose 16 positions from the book *Dvoretsky's Endgame Manual* that were recognized as fortresses by the author [24]. They are presented in Table 2 using Forsyth-Edwards (FEN) notation, which is a standard notation for describing a particular board position of a chess game. The positions were a subject of analysis by the program Stockfish. The program's backed-up evaluations of searching to depths ranging from 15 up to 40 plies were obtained.[2] Our claim was the following: Backed-up evaluations in positions that could be regarded as fortresses will not behave as it is usual for winning (losing) positions, that is they will not increase (or decrease) with increasing depth of search. The results of this experiment are demonstrated in Fig. 6 and they confirm this claim. For each of the 16 positions it holds that the backed-up evaluations remain practically the same from a certain search depth on. Similar behavior of backed-up evaluation values were obtained using various different chess programs for chess positions that are accepted as fortresses.

Table 2. The list of 16 chess fortresses given in Forsyth-Edwards (FEN) notation.

#	FEN
1	8/8/8/8/5k2/2n4p/7P/6K1 b
2	5k2/6p1/3K1pPp/3BpP1P/4P3/8/8/8 w
3	8/6kB/6P1/5K2/8/8/8/8 w
4	4K1k1/6b1/8/4n2Q/8/8/8/8 w
5	8/N1p1k3/1pPp4/1P1P1p2/3KpPp1/4P1P1/8/8 w
6	8/3k4/8/p1p2p1p/PpP2Pp1/1P3bP1/K6P/8 w
7	6r1/8/6b1/1p5k/1Pp1p1p1/2P1P1B1/1KP2P2/8 b
8	2k5/8/1p1p4/pPpPp3/2P1Pp1p/P4PbP/K2Q4/8 w
9	8/8/6k1/8/4N2p/7P/3N2K1/q7 b
10	6k1/1R6/4K1p1/7p/8/2b3P1/7P/8 w
11	4K3/5p1N/2k2PpB/6P1/8/8/b7/7q b
12	8/5pk1/5p2/3B4/5N1P/1P4P1/6K1/q7 b
13	5nQ1/4k3/3p2p1/3P1pP1/5P2/5K2/8/b7 w
14	8/4k3/1B1p4/p1pP4/PbP5/1K6/4B3/8 w
15	7r/8/8/p3p1k1/Pp1pPp2/1PpP1Pp1/2P1K1P1/8 b
16	8/B1p5/2Pp4/3Pp1k1/4P3/5PK1/3q4/8 b

Fortresses still represent the Turing test for computer chess programs, i.e., a test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human. In Fig. 7 the challenge in both diagrams is to achieve a draw, which is possible only by the means of *creating* a fortress. In each example there is theoretical draw that humans can find, but where computer will play the wrong move and actually lose [25].

[2] Stockfish 8 64-bit was used in the experiment. In the original study, the programs Rybka 3 and Houdini 1.5a x64 searched a subset of 12 positions up to 20 plies [23].

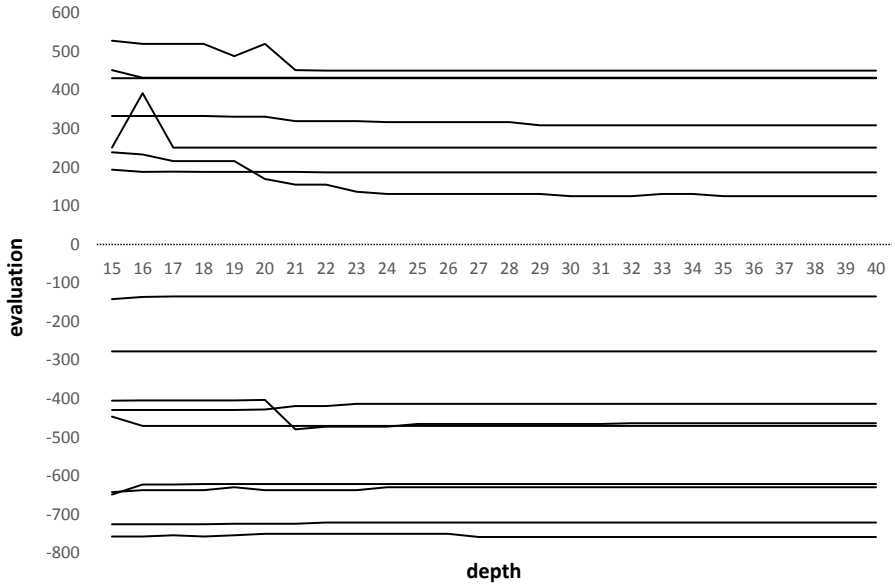


Fig. 6. In the positions that could be regarded as fortresses, backed-up evaluations obtained by STOCKFISH cease to increase (or decrease) as it is otherwise expected in winning (losing) positions.

What is particularly difficult for modern computer chess programs in both positions in Fig. 7 is that the only path to a draw (leading to a fortress) demands giving up material. As a consequence, the drawing first move typically appears as one of the least appealing options for the programs.[3]

Considering changes in heuristic evaluation from searching deeper would help in fortress recognition in both above cases as follows. When it becomes clear that the principal variation leads to a loss (i.e., when the backed-up evaluations suggest a winning advantage for the opponent, while the absolute values of evaluations keep increasing), it may be beneficial to analyze all possible moves up to some feasible search depth. If the backed-up evaluations of a certain move remain practically the same at all levels of search from a certain search depth on, the program should choose this move and achieve a draw by creating an impregnable fortress.

[3] The solutions to the problems given in Fig. 7 are 1.Ba4+!! Kxa4 2.b3+! Kb5 3.c4+! Kc6 4.d5+! Kd7 5.e6+! Kxd8 6.f5! (with a draw), and 1.Rxb7!! (1.Rxf7? g3!) 1...Rf8 (1...Rxb7 2. g3! Kg5 3.Ke2 Rb6 4.Kf1!) 2.g3! Kg6 3.Rb6+! Kg7 4.Rh6!! Kxh6 5.Ke2 Kg5 6.Kf1! Rh8 7.Kg2 a3 8.Kg1! Ra8 9.Kg2 Ra4 10.Kf1! (with a draw), respectively. The latter study was conceived by GM Miguel Illescas.

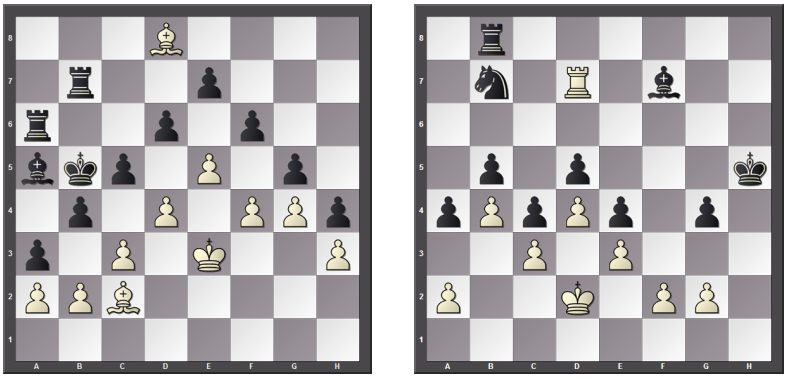


Fig. 7. White to play and draw.

6 Conclusions

In this paper, we discussed the phenomenon that from the white player's perspective, in won positions backed-up heuristic values tend to increase with the search depth. The message of this paper is that this has several implications for the practice and theory of computer game playing. These implications are summarized by the following three points.

1. When choosing the best move, candidate moves should be compared on the basis of their backed-up evaluations to equal depth for all the moves. Or, alternatively, it may be that the choice of the best move should depend on the gain in backed-up value from searching deeper.
2. Backed-up evaluations increasing with search depth in won positions offer an explanation for the finding in go-deep experiments that in won positions best-move changes with increasing search depth occur less frequently. The differences between the evaluations of won and non-won positions simply become more visible when depth increases.
3. The failure by a chess program of not recognizing a fortress can be attributed to the fact that in the choice of the best move, the depth of search is not taken into account. Again, considering the gain in heuristic evaluation from searching deeper would help in fortress recognition.

References

1. Abramson, B.: Control strategies for two-player games. ACM Computing Surveys **21** (1989) 137–161
2. Thompson, K.: Computer chess strength. In: Advances in Computer Chess 3, Pergamon Press (1982) 55–56
3. Luštrek, M., Gams, M., Bratko, I.: Is real-valued minimax pathological? Artificial Intelligence **170** (2006) 620–642

4. Pearl, J.: On the nature of pathology in game searching. Artificial Intelligence **20** (1983) 427–453
5. Scheucher, A., Kaindl, H.: Benefits of using multivalued functions for minimaxing. Artificial Intelligence **99** (1998) 187–208
6. Gomboc, D., Marsland, T.A., Buro, M.: Evaluation function tuning via ordinal correlation. In: Advances in Computer Games. IFIPAICT 135, Springer (2003) 1–18
7. Donkers, H.H.L.M., van den Herik, H.J., Uiterwijk, J.W.H.M.: Selecting evaluation functions in opponent-model search. Theoretical Computer Science **349** (2005) 245–267
8. Nau, D.S., Luštrek, M., Parker, A., Bratko, I., Gams, M.: When is it better not to look ahead? Artificial Intelligence **174** (2010) 1323–1338
9. Heinz, E.: DarkThought goes deep. ICCA Journal **21** (1998) 228–244
10. Heinz, E.: Modeling the "go deep" behaviour of Crafty and DarkThought. In: Advances in Computer Chess 9, IKAT, Universiteit Maastricht (1999) 59–71
11. Heinz, E.: Self-play in computer chess revisited. In: Advances in Computer Chess 9, IKAT, Universiteit Maastricht (1999) 73–91
12. Heinz, E.: Self-play, deep search and diminishing returns. ICGA Journal **24** (2001) 75–79
13. Heinz, E.: Follow-up on self-play, deep search, and diminishing returns. ICGA Journal **26** (2003) 75–80
14. Hyatt, R., Newborn, M.: Crafty goes deep. ICGA Journal **20** (1997) 79–86
15. Junghanns, A., Schaeffer, J.: Search versus knowledge in game-playing programs revisited. In: Proceeings of the 15th International Joint Conference on Artificial Intelligence, Volume 1, Morgan Kaufmann (1999) 692–697
16. Steenhuisen, J.R.: New results in deep-search behaviour. ICGA Journal **28** (2005) 203–213
17. Guid, M., Bratko, I.: Factors affecting diminishing returns for searching deeper. ICGA Journal **30** (2007) 65–73
18. Müller, K., Pajeken, W.: How to Play Chess Endings. Gambit Publications (2008)
19. Guid, M., Bratko, I.: Computer analysis of world chess champions. ICGA Journal **29** (2006) 3–14
20. Guid, M., Perez, A., Bratko, I.: How trustworthy is Crafty's analysis of world chess champions? ICGA Journal **31** (2008) 131–144
21. Guid, M., Bratko, I.: Using heuristic-search based engines for estimating human skill at chess. ICGA Journal **34** (2011) 71–81
22. Newborn, M.: A hypothesis concerning the strength of chess programs. ICCA Journal **8** (1985) 209–215
23. Guid, M., Bratko, I.: Detecting fortresses in chess. Elektrotehniski Vestnik **79** (2012) 35
24. Dvoretsky, M.: Dvoretsky's Endgame Manual, 2nd edition. Russell Enterprises, Inc. (2008)
25. Friedl, F.: On human and computer intelligence in chess. `http://en.chessbase.com/post/on-human-and-computer-intelligence-in-chess` (2017) [accessed 15-May-2017].