A Computational Model for Estimating the Difficulty of Chess Problems

Simon Stoiljkovikj
Ivan Bratko
Matej Guid MATEJ.GUID@FRI.UNI-LJ.SI
Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia

Abstract

In educational setting, it is often desirable to anticipate how difficult a given problem will be for a student. Assessing difficulty is also very difficult for human experts. It is an open question how to assess the difficulty automatically. In this paper, we present experiments with a computational approach to estimating the difficulty for humans of combinatorial problems such as chess. The approach is based on heuristic search by a computer to solve a given problem. Importantly, this search mimics human's problem solving taking into account the human's domain-specific knowledge. In this paper, we investigate this approach in assessing the difficulty for humans of chess tactical problems. In the experiments, we used chess tactical problems supplemented with statistic-based difficulty ratings obtained from the Chess Tempo website. We assumed these ratings as true difficulties. A "meaningful" search tree was generated by computer game tree search which attempted to emulate human's problem solving. Automatic detectors of difficulty were then induced with machine learning techniques from properties of meaningful trees. In the experiments, a number of selected chess problems were classified into difficulty classes by human experts and by our automatic detector of difficulty. The accuracy of classifications by computer models compare favorably with the accuracy of human experts.

1. Introduction

The question of automatic determination of problem difficulty for humans is complex and concerns many aspects. It depends on the type of problem and on the human's knowledge about the problem domain. In this paper, we focus our investigation on *tactical* chess problems, in which the difficulty arises from the combinatorial complexity of problems. In solving such problems, humans have to use their knowledge of the domain, including pattern-based perceptual knowledge and the skill of position analysis through calculation of concrete variations of what can happen on the board (De Groot, 1965). In the case of chess tactical problems, human players will encounter difficulty when the problem exceeds the limitations of their cognitive abilities, i.e., their ability to detect relevant motifs and to calculate variations (Chase & Simon, 1973). Similar kinds of knowledge and skill are required in solving other types of problems, for example in mathematics, everyday planning and decision making, and acting skillfully in unexpected social situations. Therefore, we believe that observations pertaining to difficulty in chess will apply to problem solving in other domains.

The goal of our research is to find a formal measure of difficulty of mental problems for humans. The goal is then to implement such a measure, possibly as an algorithm, which would enable automated difficulty estimates by computers. Obvious applications of this are in intelligent tutoring systems, or in better evaluation of student's exam results, which would take into account the difficulty of exam problems. Also, in automatic commenting it is desirable to know the difficulty of various aspects of the problem for the user, to decide whether this aspect (or part of problem solution) needs to be commented on. Typically, only those aspects of the solution that are difficult for the reader should be commented on.

In general, relatively little research has been devoted to the issue of problem difficulty. Some specific puzzles were investigated with this respect: Tower of Hanoi (Kotovsky, Hayes, & Simon, 1985), Chinese rings (Kotovsky & Simon, 1990), 15-puzzle (Pizlo & Li, 2005), Traveling Salesperson Problem (Dry et al., 2006), Sokoban puzzle (Jarušek & Pelánek, 2010), and Sudoku (Pelánek, 2011). Kotovsky and Simon pointed out two widely recognized sources of problem difficulty: the amount of knowledge required to solve the problem, and the exponential growth of the search space as search depth increases (Kotovsky & Simon, 1990). One reason a larger problem space makes problems difficult is that problems that require longer solutions increase memory demands: previously visited states typically have to be remembered for finding a solution (Atwood & Polson, 1976). Problem solvers prefer not to return to previously visited states – the more states a problem solver visits, the more states have to be maintained in memory to inform the search (Simon & Reed, 1976). LeRoux observed the effect of unrecognized equivalence as the source of problem difficulty: people often erroneously believe some equivalent states are actually new states, and thus worth visiting during the search for a solution (LeRoux, 2011).

Guid and Bratko (Guid & Bratko, 2013) proposed an algorithm for estimating the difficulty of chess positions in ordinary chess games. However, we found that this algorithm does not perform well when faced with chess tactical problems. The reason for this is that computer chess programs tend to solve tactical chess problems very quickly, usually already at the shallowest depths of search. Thus the computer simply recognizes most of the chess tactical problems to be rather easy, and does not distinguish well between positions of different difficulties (as perceived by humans). Estimating difficulty of chess tactical problems therefore requires a different approach, and different algorithms.

Chess grandmaster Kotov suggested the following strategy for choosing the best move at chess: first identify candidate moves and then methodically examine them to build up an "analysis tree." The shape of such tree (in terms of size and number of branches) may then determine how difficult a problem is (Kotov, 1971). Our approach is based on a similar idea: to use computer heuristic search for building meaningful search trees from a human problem solver's point of view. We intend to demonstrate that by analyzing properties of such trees, the program is capable of automatically predicting how difficult the problem will be for humans to solve.

In the experiments, we used chess tactical problems supplemented with statistic-based difficulty ratings obtained from the Chess Tempo website. In Fig. 1, one such tactical problem is presented. Superficially it may seem that the high number of pieces implies that the problem should be hard (at least for most players). However, a rather low Chess Tempo rating suggests the opposite. What makes this particular chess tactical problem easy? In order to understand it, we must first get acquainted with the solution. The solution to this problem is given in Section 2.3, together with the

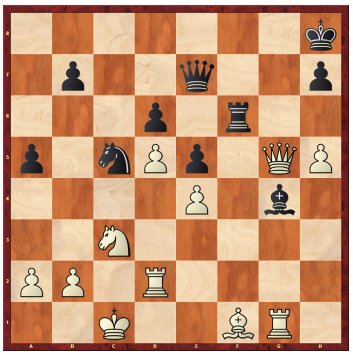

Figure 1: Chess tactical problem: Black to move wins.

computer-generated meaningful search tree (from a human problem solver's point of view). From that tree, the computer is able to infer information about the difficulty of this problem.

The paper is organized as follows. In Section 2, we introduce the domain of chess tactical problems and the concept of meaningful search trees. We also describe features that could be computed from such trees. Our experimental design is described in Section 3. Results of experiments are presented in Section 4. We conclude the paper in Section 5.

2. Method

2.1 Domain Description

In our study, we consider adversarial problem solving, in which one must anticipate, understand and counteract the actions of an opponent. Typical domains where this type of problem solving is required include military strategy, business, and game playing. We use chess as an experimental domain. In our case, a problem is always defined as: given a chess position that is won by one of the two sides (White or Black), find the winning move. A chess problem is said to be *tactical* if the solution is reached mainly by calculating possible variations in the given position, rather than by long term positional judgement. In this paper, we are not primarily interested in the process of actually solving a tactical chess problem, but in the question, how difficult it is for a human to solve the problem. A recent study has shown that even chess experts have limited abilities to assess the difficulty of a chess tactical problem (Hristova, Guid, & Bratko, 2014).

We have adopted the difficulty ratings of Chess Tempo (an online chess platform available at `www.chesstempo.com`) as a reference. The Chess Tempo rating system for chess tactical problems is based on the Glicko Rating System (Glickman, 1999). Problems and users (that is humans that solve the problems) are both given ratings, and the user and problem rating are updated in a manner similar to the updates made after two chess players have played a game against each other, as in the Elo rating system (Elo, 1978). If the user solves a problem correctly, the problem's rating goes down, and the user's rating goes up. And vice versa: the problem's rating goes up in the case of incorrect solution. The Chess Tempo ratings of chess problems provide a basis from which we estimate the difficulty of a problem.

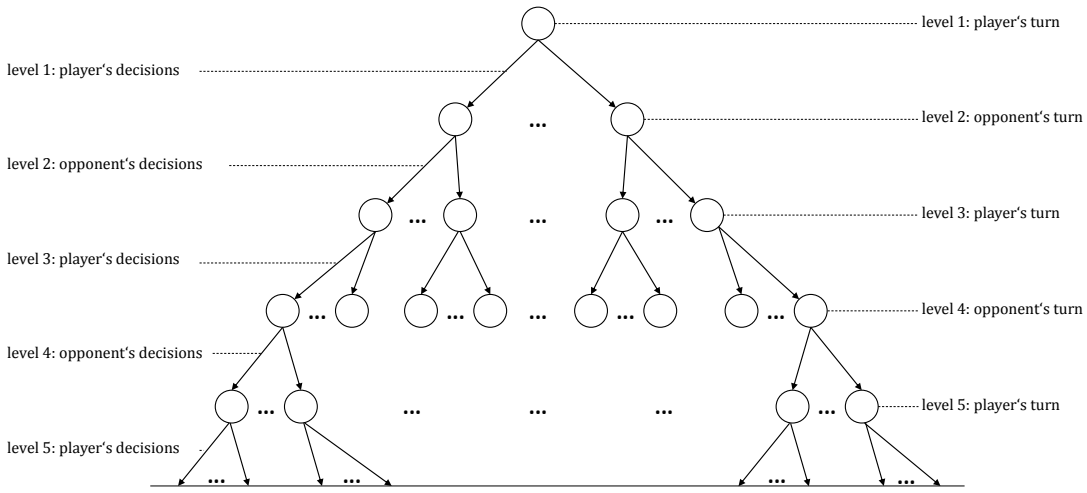

Figure 2: A part of a game tree, representing a problem in adversarial problem solving.

2.2 Meaningful Search Trees

A person is confronted with a problem when he wants something and does not know immediately what series of actions he can perform to get it (Newell & Simon, 1972). In artificial intelligence, a typical general scheme for representing problems is called *state space*. A state space is a graph whose nodes correspond to problem situations, and a given problem is reduced to finding a path in this graph.

The presence of an adversary complicates that search to a great extent. Instead of finding a linear sequence of actions through the problem space until the goal state is reached, adversarial problem solving confronts us with an expanding set of possibilities. Our opponent can make several replies to our action, we can respond to these replies, each response will face a further set of replies etc. Thus, in adversarial problem solving, the state space is usually represented by a *game tree*. In computer problem solving, only a part of complete game tree is generated, called a *search tree*, and a heuristic evaluation function is applied to terminal positions of the search tree. The heuristic evaluations of non-terminal positions are obtained by applying the *minimax principle*: the estimates propagate up the search tree, determining the position values in the non-leaf nodes of the tree.

Game trees are also a suitable way of representing chess tactical problems. In Fig. 2, a portion of a problem's game tree is displayed. Circles represent chess positions (states), and arrows represent chess moves (actions). Throughout the article, we will use the following terms: the *player* (i.e., the problem solver) makes his decisions at *odd levels* in the tree, while the *opponent* makes his decisions at *even levels*. The size of a game tree may vary considerably for different problems, as well as the length of particular paths from the top to the bottom of the tree. For example, a *terminal state* in the tree may occur as early as after the player's *level-1* move, if the problem has a checkmate-in-one-move solution.

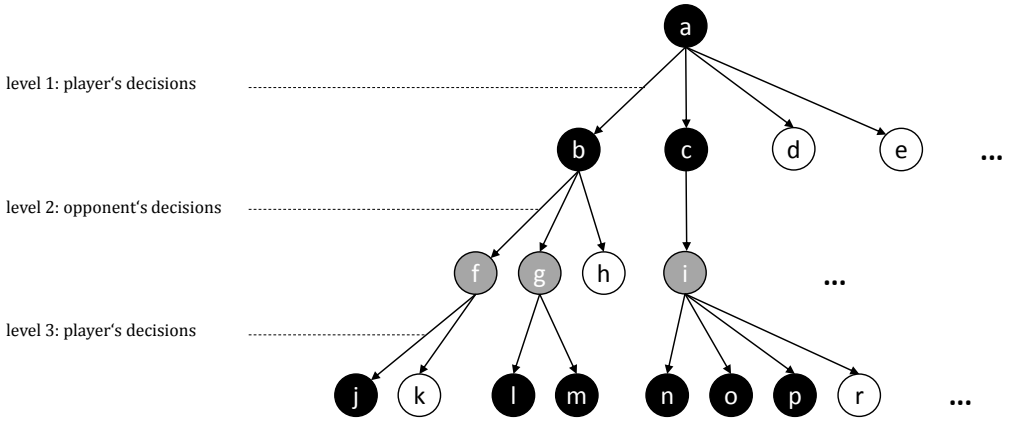

Figure 3: The concept of a meaningful search tree.

In type of problems in which the difficulty arises from the combinatorial complexity of searching among alternatives, it is typically infeasible for a human to consider *all* possible paths that might lead to the solution of the problem. Human players therefore heuristically discard possibilities (moves) that are of no importance for finding the solution of a particular problem. In doing so, they are mainly relying on their knowledge and experience. In fact, human problem solvers "construct" (mentally) their own search trees while solving a problem, and these search trees are essentially different than the ones obtained by computer heuristic search engines. The search trees of humans, in the sequel called "meaningful trees," are typically much smaller, and, most importantly, they mainly consist of what represents meaningful (from a human problem solver's point of view) states and actions in order to solve the problem.

A natural assumption is that the difficulty of a chess problem depends on the size and other properties of the chess position's meaningful tree. In order to enable automated assessment of the difficulty of a problem for a human, we therefore focused on constructing search trees that are meaningful from a human problem solver's point of view. Such trees should, above all, consist of actions that a human problem solver would consider. The basic idea goes as follows. Computer heuristic search engines can be used to estimate the values of particular nodes in the game tree of a specific problem. Only those nodes and actions that meet certain criteria are then kept in what we call a *meaningful search tree*. By analyzing properties of such a tree, we should be able to infer certain information about the difficulty of the problem for a human.

The concept of a meaningful search tree is demonstrated in Fig. 3. Black nodes represent states (positions) that are won from the perspective of the player, and grey nodes represent states that are relatively good for the opponent, as their evaluation is the same or similar to the evaluation of his best alternative. White nodes are the ones that can be discarded during the search, as they are not winning (as in the case of the nodes labeled as *d*, *e*, *h*, *k*, and *r*), or they are just too bad for the opponent (*h*).

If the meaningful search tree in Fig. 3 represented a particular problem, the initial problem state *a* would be presented to the problem solver. Out of several moves (at level 1), two moves lead to the solution of the problem: *a–b* and *a–c*. However, from state *c* the opponent only has one answer: *c–i* (leading to state *i*), after which three out of four possible alternatives (*i–n*, *i–o*, and *i–p*) are winning. The other path to the solution of the problem, through state *b*, is likely to be more difficult: the opponent has three possible answers, and two of them are reasonable from his point of view. Still, the existence of multiple solution paths, and very limited options for the opponent suggest that the problem (from state *a*!) is not difficult.

Meaningful trees are subtrees of complete game trees. The extraction of a meaningful tree from a complete game tree is based on heuristic evaluations of each particular node, obtained by a heuristic-search engine searching to some arbitrary depth *d*. In addition to d, there are two other parameters that are chess engine specific, and are given in centipawns, i.e. the unit of measure used in chess as a measure of advantage, a centipawn being equal to 1/100 of a pawn. These two parameters are:

- The minimal heuristic value *w* that is supposed to indicate a won position.
- The margin *m* by which the opponent's move value *V* may differ from his best move value $BestV$. All the moves evaluated less than $BestV - m$ are not worth considering, so they do not appear in the meaningful tree.

It is important to note that domain-specific pattern-based information (e.g., the relative value of the pieces on the chessboard, king safety etc.) is *not* available from the meaningful search trees. Moreover, as it is suggested in Fig. 3, it may also be useful to consider whether a particular level of the tree is odd or even.

2.3 Illustrative Examples

In Fig. 4, a Chess Tempo tactical problem is shown. Superficially it may seem that the low number of pieces implies that the problem should be easy (at least for most players). However, a rather high Chess Tempo rating (2015.9 points calculated from 1656 problem-solving attempts) suggests that the problem is fairly difficult.

What makes a particular chess tactical problem difficult? Again, in order to understand it, we must first get acquainted with the solution. In Fig. 4, Black threatens to win the Rook for the Bishop with the move 1... Bf2xe1 (that is, Black bishop captures White rook on square e1; we are using standard chess notation). And if White Rook moves from e1, the Bishop on e2 is *en prise*. However, first the Black Rook must move from e5, otherwise the White Pawn on f4 will capture it.

So the question related to the problem is: what is the best place for the attacked Black Rook? Clearly it must stay on e-file, in order to keep attacking the White Bishop. At first sight, *any* square on e-file seems to be equally good for this purpose. However, this exactly may be the reason why many people fail to find the right solution. In fact, only one move wins: 1... Re5-e8 (protecting the Black Rook on d8!). It turns out that after any other Rook move, White plays 2.Re1-d1, saving the Bishop on e2, since after 2... Rd8xd1 3.Be2xd1(!) the Bishop is no longer attacked. Moreover, even

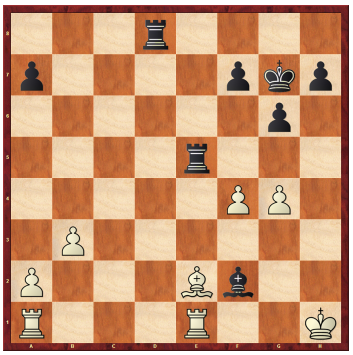

Figure 4: An example of a fairly difficult chess tactical problem: Black to move wins.

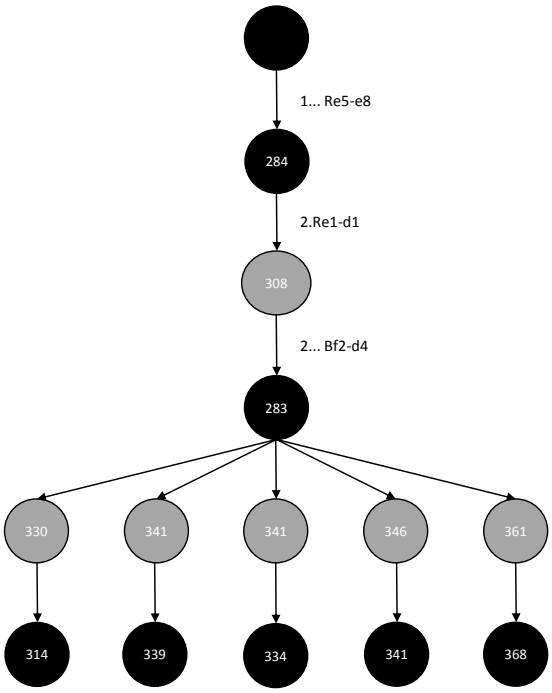

Figure 5: The meaningful search tree for the fairly difficult problem in Fig. 4.

after the right move 1... Re5-e8, Black must find another sole winning move after White's 2.Re1-d1: moving the Bishop from f2 to d4, attacking simultaneously the Rook on a1 and the Bishop on e2.

Fig. 5 shows the meaningful tree for the above example. Chess engine STOCKFISH (one of the best computer chess programs currently available) at 10-ply depth of search was used to obtain the evaluations of the nodes in the game tree up to level 5. The parameters w and m were set to 200 centipawns and 50 centipawns, respectively. The value in each node gives the engine's evaluation (in centipawns) of the corresponding chess position.

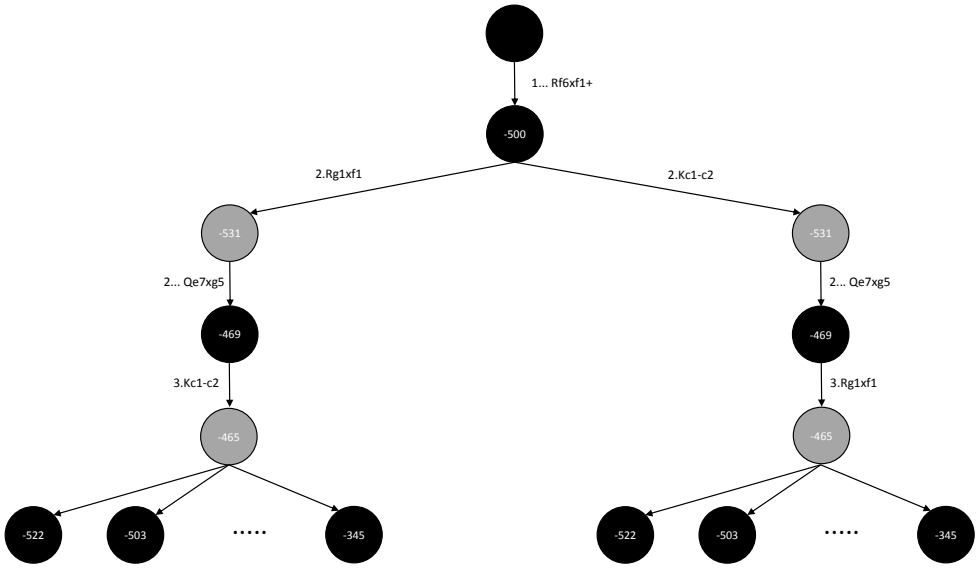

Figure 6: The meaningful search tree for the fairly easy problem in Fig. 1.

As explained before, the meaningful tree is supposed to contain moves that an experienced chess player will consider in order to find the solution of the problem. In this sense, the chess engine-computed meaningful tree approximates the actual meaningful tree of a human player. In the present case, the tree suggests that the player has to find a unique winning move after every single sensible response by the opponent. This implies that the problem is not easy to solve by a human. On the other hand, we have no (pattern-based) information about the cognitive difficulty of these moves for a human problem solver. An alternative to chess-engine's approximation of human's meaningful tree would be to model complete human player's pattern-based knowledge sufficient. However, that would be a formidable task that has never been accomplished in existing research.

Let us now consider a fairly easy chess tactical problem in Fig. 1 (see Section 1). In the diagrammed position, Black moves the rook from f6 to f1 (1...Rf6-f1+), giving a check to the White King. On the next move, Black simply takes the White Queen by moving the Black Queen from e7 to g5 (2... Qe7xg5), with large material gain and a winning position. The meaningful tree for this example is shown in Fig. 6. An obvious property of this tree is that the winning player has a wide range of winning moves to choose from at the level 5 in this tree. A high branching factor at level 5 in the tree is actually an important indicator that a problem is not difficult: many moves win for the player. This means that the position is already so strong that it no longer matters what move the player chooses because almost everything wins. A reasonable explanation is that the solution of such a problem is rather short (only two moves at most by the winning side), and (typically) associated with sufficiently large material gains.

Table 1: Attributes and their basic descriptions.

#	*Attribute*	*Description*
1	Meaningful(L)	Number of moves in the meaningful search tree at level L.
2	PossibleMoves(L)	Number of all legal moves at level L.
3	AllPossibleMoves	Number of all legal moves at all levels.
4	Branching(L)	Branching factor at each level L of the meaningful search tree.
5	AverageBranching	Average branching factor for the meaningful search tree.
6	NarrowSolution(L)	Number of moves that only have one meaningful answer, at level L.
7	AllNarrowSolutions	Sum of NarrowSolution(L) for all levels L.
8	TreeSize	Number of nodes in the meaningful search tree.
9	MoveRatio(L)	Ratio between meaningful moves and all possible moves, at level L.
10	SeeminglyGood	Number of non-winning first moves that only have one good answer.
11	Distance(L)	Distance between start and end square for each move at level L.
12	SumDistance	Sum of Distance(L) for all levels L.
13	AverageDistance	Average distance of all the moves in the meaningful search tree.
14	Pieces(L)	Number of different pieces that move at level L.
15	AllPiecesInvolved	Number of different pieces that move in the meaningful search tree.
16	PieceValueRatio	Ratio of material on the board, player versus opponent.
17	WinningNoCheckmate	Number first moves that win, but do not lead to checkmate.
18	BestMoveValue	The computer evaluation of the best move.
19	AverageBestMove(5)	Average best-move value of all best moves occurring at level 5.

Table 2: The values of multi-level attributes for the example in Fig. 4.

#	*Attribute*	*level 1*	*level 2*	*level 3*	*level 4*	*level 5*
1	Meaningful(L)	1	1	1	5	5
2	PossibleMoves(L)	47	22	39	23	178
4	Branching(L)	1	1	1	5	1
6	NarrowSolution(L)	1	1	1	0	5
9	MoveRatio(L)	0.021	0.045	0.026	0.217	0.028
11	Distance(L)	3	1	2	13	16
14	Pieces(L)	1	1	1	2	2

2.4 Attribute Description

There are certain properties of a meaningful tree that are indicative of high or low difficulty. Table 1 shows the attributes that were used in the experiments. Below we describe in more detail some multi-level attributes, i.e., the attributes that have one value for each level L in the meaningful search tree. Table 2 shows the values of multi-level attributes for the example in Fig. 4. We also describe the attribute `SeeminglyGood`, which indicates the number of *non-winning* first moves with only one good reply. This attribute is not computable from meaningful search trees and was obtained separately.

2.4.1 Meaningful(L) and PossibleMoves(L)

The attribute `Meaningful(L)` indicates the number of moves in the meaningful search tree at level L. The procedure to obtain the moves at particular level L in the meaningful tree is given in Algorithm 1. The attribute `PossibleMoves(L)` is also used in this procedure, and represents the number of all legal moves at level L. The aforementioned (see Section 2.2) parameters w (i.e., the minimal heuristic value that is supposed to indicate a won position), and m (i.e., the margin by which the opponent's move value *V* may differ from his best move value) also play an important role. In all the experiments, we consistently used the following (arbitrarily set) values: w = 200 centipawns and m = 50 centipawns.

Algorithm 1 The procedure to obtain moves at level L in the meaningful tree.

```
meaningful[L] = []
for each move in possibleMoves[L] do
  if winningSideToMove then
    if |move.score| >= w then
      append move to meaningful[L]
    end if
  else
    if |best_move.score - move.score| <= m then
      append move to meaningful[L]
    end if
  end if
end for
```

The number of all legal moves at level L in the tree (given in `PossibleMoves(L)`) may reveal (directly or indirectly) a great deal of positional factors such as the nature of the position (an open versus a closed game), the number of pieces on the board, whether the king is in check (which typically leaves very few options), etc. Both attributes, `Meaningful(L)` and `PossibleMoves(L)`, consist of different values for each level L of the meaningful search tree.

2.4.2 Branching(L)

The branching factor at each level L of the meaningful search tree (represented by the attribute `Branching(L)`) is obtained in a rather straightforward fashion, by dividing `Meaningful(L-1)` by `Meaningful(L)`. A higher branching factor reflects more viable alternatives (which is usually favorable) for the side to move.

2.4.3 NarrowSolutions(L)

Even though this attribute reflects one of the properties of the meaningful search trees, it also maps into the following well-known concept in the domain of chess: a "forced" move. A move is forced when a reply in a certain way is required, or the ways in which one can respond are greatly limited. In general, a forced move occurs right after a check, a capture, or a threat. Note that in the case of chess tactical problems a "threat" may simply be to achieve the balance or to prevent the winning side from actually winning. In the meaningful search trees, the number of forced moves at level L in the tree thus corresponds to the number of moves at level L that represent the only meaningful answer to the move at level L-1. When considering the value of the attribute `NarrowSolutions(L)`, it is particularly relevant whether a particular level of the tree is odd (the problem solver's move) or even (the opponent's move). For example, a high value of the attribute `NarrowSolutions(L)` at level 5 (this applies to the fairly difficult position in Fig. 4) indicates that the winning side had to find certain unique solutions after the opponent's second response.

2.4.4 MoveRatio(L)

The attribute MoveRatio(L) is obtained by dividing Meaningful(L) and PossibleMoves(L). Five values are obtained, one at each level (L). It expresses how many of all possible moves at particular level are "meaningful." From the perspective of the winning side (levels 1, 3, and 5 in our case), a higher proportion of meaningful (i.e., winning) moves suggests that the solution is not difficult to find. In particular, a high value of the attribute `MoveRatio(L)` at level 5 implies that the position is already winning at that point.

2.4.5 Distance(L)

This attribute tells us about the length of the piece moves at the particular level L in the meaningful tree. The sum of all distances between the start square and the end square for each of the meaningful moves at level L is calculated according to the Chebyshev distance. In the game of chess, this equals to the minimum number of moves needed by a king to go from one square on a chessboard to another. For example, the Chebyshev distance between f6 and e2 equals 4.

2.4.6 SeeminglyGood

One of the reason why a certain chess tactical problem is difficult is the presence of a "seemingly good" move. These are appealing moves that seem to lead to victory, but in fact they are not. Essentially, such a move cannot be included in the meaningful search tree (as it is not winning and thus not sensible from the problem solver's perspective). We thus defined "seemingly good" moves

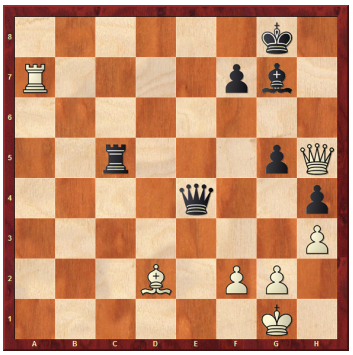

Figure 7: Black to move wins. Does the move 1...Rc5-c1+ lead to a win?

as non-winning moves to which the opponent only has one acceptable answer. The value of the attribute `SeeminglyGood` is the number of such "seemingly good" moves. The presence (or even a high number) of such moves speak in favor of the problem being a difficult one. The parameter w was set to a lower value in order to obtain non-winning (and also non-losing) first moves.

An example of a seemingly good move is given in Fig. 7. Black has a seemingly attractive variation at disposal: 1...Rc5-c1+ 2.Bd2xc1 (the only response that does not lose) Qe5-e1+ 3.Kg1-h2 Bg7-e5+, which wins for Black after 4.g2-g3 Qe1-f2+ 5.Kh2-h1 Qf2-f1+ 6.Kh1-h2 h4xg3 checkmate. As it turns out, this is actually a trap that many chess players fall into. The given variation in fact does not win, as White has the move 4.f2-f4 at disposal, and Black is suddenly in urge to force a draw by giving a perpetual check with the Queen (repeatedly moving to g3 and e1). The winning line is associated with a completely different motif: 1...Qe5-b1+ 2.Kg1-h2 Qb1-b8+, winning the Black Rook on a7. Such long moves with the queen, switching from one side of the board to another, are for some reason particularly difficult to find (Neiman & Afek, 2011).

3. Experimental Design

3.1 Evaluation of the Computational Model

The aim of the first experiments was to assess the utility of the meaningful search trees for differentiating between three groups of chess tactical problems, each group belonging to one of the following difficulty classes: "easy," "medium," and "hard". We used 900 problems from the Chess Tempo website: 300 of them were labeled as "hard" (with average Chess Tempo Standard Rating $\varnothing$ = 2088.8, σ = 74.3), 300 as "medium" ($\varnothing$ = 1669.3, σ = 79.6), and 300 as "easy" ($\varnothing$ = 1254.6, σ = 96.4). Chess engine STOCKFISH at 10-ply depth of search (the selected search depth sufficed to find a win in every problem in the data set) was used to build the meaningful search tree up to level 5. The parameters w and m were set to 200 centipawns and 50 centipawns, respectively.

The attributes presented in in Table 1 were used to induce predictive models. Attributes 1–10 represent features that mainly reflect properties of the meaningful search trees, while attributes 11–19 are more closely related to the properties of the chosen domain – chess in our case. We used five standard machine learning classifiers (and 10-fold cross validation) to estimate performance of (all) the attributes, and the performance of each aforementioned groups of attributes.

Table 3: Basic description of the Chess Tempo problem set.

#	*Problem ID*	*FEN*	*Rating*
1	72506	1r6/5pkp/3p2p1/1R1P4/n1P2P2/1pn4P/1R6/5BK1 b	1702.1
2	90217	8/2R2pk1/2n3bp/p5p1/3p2P1/P4Q2/2q2P1P/4B1K1 b	1254.4
3	32757	6k1/p5rp/2q2p1Q/2p1pP2/P1P4K/3P4/7P/1R6 b	2142.3
4	10844	r5k1/3P1p1p/1R4p1/2BR4/p1r4b/5P1P/5P2/6K1 w	2156.4
5	106131	2r1b3/pp3p1p/1n1R4/2q2k2/2P1r3/1B4Q1/P5PP/7K w	1698.7
6	78455	4k3/6b1/3n2pp/Q2Bq3/8/3P2PP/5r2/2R4K w	2154.6
7	91842	5k1r/pp3p2/3Rp3/5bP1/8/2P1QP2/4RK2/7q w	1253.8
8	97635	1rB2k2/4p3/p2p4/q2N4/2Pp2Q1/5P2/PP4P1/2K5 b	1245.3
9	136203	4q1rk/3b1p1p/p1rp1PpQ/4b3/1p4P1/1P4N1/1PP4P/1K1R1RB1 w	1695.7
10	122282	r2q1rk1/p4pbp/1p1p2p1/2nP4/8/3B4/PPPQ1BPP/1K1R3R b	1246.5
11	102310	6rB/5p1k/7p/6p1/2q1p3/3bP2P/1Q3PP1/R5K1 w	1702.0
12	77301	2Q5/1n3rk1/3p4/3Pp1p1/1P2P3/2R2PPK/2N5/6q1 b	1702.5
13	125627	2r5/3n1pkp/4p1b1/2qBP3/1p4NQ/1Pr3P1/5P2/3RR1K1 b	1250.7
14	102704	r4rk1/1b5p/p1qp2p1/1pn2P2/3B3Q/P1N5/1PP1B1PP/5R1K w	2145.2

3.2 Chess Experts' Difficulty Assessment

The aim of the second experiments was to observe chess experts' ability to assess the difficulty of chess tactical problems, in particular their ability to classify them into the three difficulty classes in a similar way as our computational model. The participants, 10 chess experts, estimated the difficulty of a selection of Chess Tempo problems. Only problems with established difficulty ratings (each attempted by at least 500 Chess Tempo users) were used. The participants consisted of 7 male and 3 female chess players (average age: 37 years, $\sigma = 16.4$). Their FIDE Elo ratings vary between 1963 and 2255 (average: 2098.5, $\sigma = 93.6$). The Elo rating system (Elo, 1978) is adopted by FIDE (World Chess Federation) to estimate the strength of chess players.

Participants were presented with 14 positions – chess tactical problems – randomly selected from Chess Tempo according to their difficulty ratings. Based on their Chess Tempo ratings, the problems were divided into three classes of difficulty: "easy" ($N = 5$ problems, average Chess Tempo rating $R = 1250.1$, $\sigma = 4.1$), "medium" ($N = 5$; $R = 1700.2$, $\sigma = 2.9$), and "hard" ($N = 4$; $R = 2151.9$, $\sigma = 6.5$). While the problems within the same difficulty class have very similar difficulty rating, each of the three classes is separated from the other by more than 400 Chess Tempo rating points. Table 3 displays the properties for the 14 tactical chess problems: Chess Tempo problem ID, problem's FEN (starting position in a standard chess notation), and Chess Tempo rating.

The participants were instructed to estimate the difficulty classes of the 14 problems. They were also informed that the difficulty of each problem was estimated from its Chess Tempo rating, and

Table 4: The results of experiments with the three groups of attributes.

	All Attributes			Attributes 1–10			Attributes 11–19		
Classifier	**CA**	**AUC**	**Brier**	**CA**	**AUC**	**Brier**	**CA**	**AUC**	**Brier**
neural network	0.83	0.95	0.24	0.81	0.94	0.26	0.54	0.73	0.57
classification tree	0.82	0.89	0.33	0.78	0.89	0.36	0.51	0.70	0.63
logistic regression	0.80	0.94	0.29	0.79	0.94	0.29	0.52	0.72	0.58
random forest	0.76	0.91	0.40	0.77	0.89	0.42	0.54	0.75	0.57
CN2 rules	0.72	0.88	0.39	0.74	0.90	0.36	0.53	0.73	0.57

Table 5: The participants' results of difficulty estimation.

ELO	*2145*	*2194*	*2255*	*2042*	*1963*	*1980*	*2040*	*2105*	*2162*	*2099*
# correct	7	9	8	7	7	7	4	9	10	6
% correct	0.50	0.64	0.57	0.50	0.50	0.50	0.29	0.64	0.71	0.43

that each one was from one of the following difficulty classes: "easy" (about 1250 rating points), "medium" (about 1700 rating points), or "hard" (about 2150 rating points). The problems were given in a similar form as Fig. 1, i.e., as diagrams supplemented with the description which side (to move) wins. They were presented in the same order as given in Table 3. The solutions to the problems were given separately, at the last page of the four-page document, in order to allow the participants to first try to solve the problems on their own. The participants had an access to all problem positions at once, allowing them a fair comparison between them. The questionnaire for the participants is available on `http://www.ailab.si/matej`.

4. Results

Table 4 shows the results of experiments with all three aforementioned groups of attributes (see Section 3). Classification accuracy (CA), Area under ROC curve (AUC), and Brier score are given for each of the five classifiers and for each group of attributes. All classifiers were able to differentiate between easy, medium, and difficult problems with a high level of accuracy when all the attributes given in Table 1 were used. However, it is interesting to observe that their performance remained almost the same when only attributes 1–10 (see Table 1) were used. These attributes are computable from the structure of meaningful search trees, and contain no domain-specific knowledge. On the other hand, the performance of the classifiers dropped significantly when attributes 11–19 were used. These attributes were still derived from the meaningful search trees, however, they do contain chess-related knowledge such as information about piece movements, piece values, and checkmates, but do not concern the structure of the trees at all.

Table 5 shows FIDE ELO ratings of the participants and their success rate at the difficulty estimation of the 14 problems. The average classification accuracy (CA) by the participants was 0.53 ($\sigma = 0.12$), the highest value being 0.71, that is less than the expected accuracy by any of the classifiers from Table 4. We observed several disagreements between the participants' difficulty estimations. E.g., in 9 out of 14 cases the problem was classified as any of the three class values.

5. Conclusions

We tackled the problem of how to assess automatically the difficulty of a mental problem for a human, which depends on the human's expertise in the problem domain. We focussed in our experiments on assessing the difficulty of tactical chess problems.

In our approach, we assume that the difficulty depends on the amount of search required to be carried out by the player before he or she finds the solution. An experienced player only has to search a small part, here called "meaningful tree," of the complete search tree. The rest of the search tree is pruned away by a large amount of pattern-based chess knowledge that the player has accumulated through experience. To assess the difficulty of the problem for the player, one could try to automatically reconstruct the player's meaningful tree. This would however require a computer implementation of the player's pattern knowledge which would be extremely hard due to the complexity of this knowledge.

In this paper, however, we avoid the implementation of the player's chess knowledge. We approximate the player's meaningful tree by extracting "critical" chess variations from the complete search tree automatically by a strong chess playing program. The extracted subtree only contains best or "reasonable" moves which are selected according to the chess program's backed-up evaluations of positions. The assumption here is that the "reasonable" moves obtained in this manner correspond well to the moves that seem reasonable according to a chess player's pattern knowledge.

To construct an automatic difficulty estimator of chess problems based on meaningful trees, we constructed a set of attributes and performed classification learning using these attributes for the task of classifying positions into classes of difficulty. These were "easy," "medium," and "hard" in our present study. The attributes were of two types: (1) those that are derived from the structure of the meaningful tree only, and (2) attributes that reflect chess-specific contents of the concrete chess positions and moves in the tree. A set of standard machine learning methods were used to induce difficulty estimators from a stratified random selection of rated chess problems from the Chess Tempo database.

We carried out experiments with the induced difficulty predictors, in order to estimate the accuracy of such automatic predictors, and to compare their accuracy with that of human chess experts rated roughly between a candidate chess master and a chess master.

The important findings from the experimental results are:

1. The classification accuracy of automatic difficulty predictors in 3-class classification was about 80%. It did not depend very much on the kind of machine learning method used.

2. Using tree structure attributes only achieved practically the same accuracy as using all the attributes. However, using chess-contents attributes only produced a much lower accuracy of about 50%.

3. Classification accuracy (with the same 3 difficulty classes) achieved in the experiments with chess experts was on average just over 50%. This shows that estimating difficulty is difficult for human experts. Their predictive performance compared to the performance of automatic predictors using tree structure attributes was significantly inferior.

References

Atwood, M., & Polson, P. (1976). A process model for water jug problems. *Cognitive Psychology*, *8*, 191Ű216.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive psychology*, *4*, 55–81.

De Groot, A. D. (1965). *Thought and choice in chess*. The Hague: Morton.

Dry, M., Lee, M., Vickers, D., & Hughes, P. (2006). Human performance on visually presented traveling salesperson problems with varying numbers of nodes. *Journal of Problem Solving*, *1*, 20–32.

Elo, A. E. (1978). *The rating of chessplayers, past and present*. Arco Pub.

Glickman, M. E. (1999). Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics*, *48*, 377–394.

Guid, M., & Bratko, I. (2013). Search-based estimation of problem difficulty for humans. In H. Lane, K. Yacef, J. Mostow, & P. Pavlik (Eds.), *Artificial intelligence in education*, Vol. 7926 of *Lecture Notes in Computer Science*, 860–863. Springer.

Hristova, D., Guid, M., & Bratko, I. (2014). Assessing the difficulty of chess tactical problems. *International Journal on Advances in Intelligent Systems*, *7*, 728–738.

Jarušek, P., & Pelánek, R. (2010). Difficulty rating of sokoban puzzle. *Proc. of the Fifth Starting AI Researchers' Symposium (STAIRS 2010)* (pp. 140–150). IOS Press.

Kotov, A. (1971). *Think like a grandmaster*. Batsford.

Kotovsky, K., Hayes, J. R., & Simon, H. A. (1985). Why are some problems hard? Evidence from tower of Hanoi. *Cognitive Psychology*, *17*, 248–294.

Kotovsky, K., & Simon, H. A. (1990). What makes some problems really hard: Explorations in the problem space of difficulty. *Cognitive Psychology*, *22*, 143–183.

LeRoux, A. L. (2011). *Problem solving difficulty from unrecognized equivalency*. Doctoral dissertation, University of North Carolina at Greensboro.

Neiman, E., & Afek, Y. (2011). *Invisible chess moves: Discover your blind spots and stop overlooking simple wins*. New in Chess.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Prentice-Hall.

Pelánek, R. (2011). Difficulty rating of sudoku puzzles by a computational model. *Proc. of Florida Artificial Intelligence Research Society Conference (FLAIRS 2011)* (pp. 434–439). AAAI Press.

Pizlo, Z., & Li, Z. (2005). Solving combinatorial problems: The 15-puzzle. *Memory and Cognition*, *33*, 1069–1084.

Simon, H. A., & Reed, S. K. (1976). Modeling strategy shifts in a problem-solving task. *Cognitive Psychology*, *8*, 86–97.