Automatic Recognition of Similar Chess Motifs

Miha Bizjak[1] and Matej Guid[1]

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

Abstract. We present a novel method to find chess positions similar to a given query position from a collection of chess games. We consider not only the static similarity resulting from the arrangement of chess pieces, but also the dynamic similarity involving the recognition of chess motifs and the tactical, dynamic aspects of position similarity. By encoding chess tactical problems as text documents, we use information retrieval techniques to enable efficient approximate searches. We have also developed a method for automatically generating tactical puzzles from a collection of chess games. We have experimentally shown the importance of including both static and dynamic features for successful recognition of similar chess motifs. The experiments have clearly shown that dynamic similarity plays a very important role in the evaluation of the similarity of chess motifs by both the program and chess experts.

Keywords: Problem solving · Chess motifs · Automatic similarity recognition

1 Introduction

The focus of this paper is on automatic retrieval of similar tactical problems from a large collection of chess games. The term *tactic* is used in chess to describe a series of moves that exploit a particular position on the board and allow the player to gain material, gain a positional advantage, or even force a checkmate.

For chess players to progress, tactical problems are incredibly important. Knowing tactical motifs helps them to recognise when a winning or drawing combination might exist in a position. By solving tactical problems, chess players improve their tactical skills. It is not uncommon for games to be decided by tactics, because even a single mistake gives the opportunity to use a tactic that changes the outcome of the game. A large number of patterns or *tactical motifs* have been defined in chess literature to help players discover tactical opportunities during a game [2]. The ability to recognise chess motifs during a game is one of the key components of becoming a competent chess player [4].

In order to provide useful teaching material to their students, chess instructors often look for examples from actual games that exhibit relevant chess motifs. However, the human mind is not capable of sifting through thousands or even millions of games to find problems with similar chess motifs and similar solutions to those overlooked by students. Contextually similar chess positions could also be used to annotate chess games [6] and in intelligent tutoring systems [8].

Our research aims for the automatic retrieval of similar chess motifs for a given query position in a large collection of archived chess games. It needs to be both efficient in terms of speed and effective in terms of the quality/similarity of the retrieved results.

1.1 Related Work

There are a large number of possible positions in a game of chess, so a query-by-example [9] system that only looks for exact matches would not be effective. To mitigate the problem, the Chess Query Language (CQL) [3] allows searching for approximate matches of positions, but requires the user to define complex queries in the system-specific language. In addition, CQL works directly with PGN game archive files and checks each game sequentially, making it inefficient for querying larger databases.

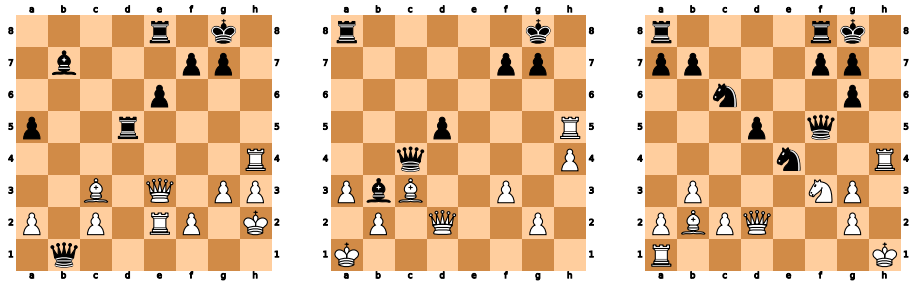
To overcome these problems, an information retrieval-based approach was proposed, in which a textual representation is constructed for each board position and information retrieval (IR) methods are used to compute the similarity between chess positions [5]. Instead of creating a query manually, the user simply enters a FEN and the query encoding the features of the position is automatically created internally. Initially, a naïve coding was used, which only contained the positions of the individual pieces. Additional information about the mobility of the individual pieces and the structural relationships between the pieces improved the results.

Besides improving the above approach by introducing advanced static patterns such as pawn structures, we aim to develop a state-of-the-art chess engine for finding similar chess tactics that also takes into account the dynamic aspects of chess tactics. In terms of the progress of chess players, this dynamic aspect is much more relevant when considering a contextually similar tactical problem.

2 Domain description

Figure 1 illustrates three chess tactics with the same chess motif: the white rook is sacrificed in the corner of the board and the black king must capture it, allowing the white queen to appear with check (it cannot be captured due to the activity of the white bishop along the long diagonal) and deliver checkmate on the next move. Note that the similarity is due to a dynamic aspect of the combination, which is based on the underlying chess motif. We use standard chess annotation to describe chess moves.

Figure 1 shows three chess tactics with the same chess motif: the white rook is sacrificed in the corner of the board and the black king must capture it so that the white queen can appear with check (it cannot be captured due to the activity of the white bishop along the long diagonal) and deliver checkmate on the next move. Note that the similarity is due to a dynamic aspect of the combination based on the underlying chess motif. We use the standard chess annotation to describe chess moves.



(a) (b) (c)

Fig. 1. Chess tactics with the same chess motif. In all three positions, White wins with 1.♖h8+ ♔xh8 2.♕h6+ ♔g8 3.♕xg7 checkmate.

To illustrate the difference between *static* and *dynamic similarity*, consider the removal of the white rook on the h-file in each of the three positions in Fig. 1. The new positions would be statically very similar, but the dynamic similarity would disappear and the tactic mentioned above would no longer apply.

We are particularly interested in detecting dynamic similarity, i.e. finding positions with similar motif(s) in the solution of the tactic. However, we also want to consider static similarity, i.e. finding problems with similar initial positions. Fig. 1 also illustrates the behaviour of our program: when querying a chess tactic in (a), the program found two similar chess tactics, shown in (b) and (c).

3 Similarity computation

To determine similarity between tactical problems we use an approach based on information retrieval. A set of features is computed from each problem's starting position and its solution move sequence. The features are then converted into textual terms, forming a document that represents the problem. A collection of documents is used to build an index, which can then be queried using the textual encoding of a new position to retrieve the most similar positions in the index. For the implementation of the system for indexing and retrieval of similar tactics we use the *Apache Lucene Core* library. Search results are ranked using the BM25 ranking function [7].

For each tactic, the input consists of a starting position in FEN format and a solution move sequence in algebraic notation. The solution can be provided with the position or calculated using a chess engine. Sections 3.1 and 3.2 describe the features and terms that are generated, and Figure 2 shows an example of a text encoding.

Feature set	Generated terms
static_positions	Ra1 Kh1 Pa2 Bb2 Pc2 Qd2 Pg2 ...
	Rb1|0.89 Rc1|0.78 Rd1|0.67 ...
	B>pg7 Q>pd5 n>Qd2 n>Pg3 ...
	R<Kh1 R<Pa2 K<Pg2 P<Pb3 ...
	R=pa7 R=ra8 q=Pc2 r=Ra1 r=Pa2 ...
static_pawns	Ig2 Ig3 id5 Lc2 Sg2-g3 sg6-g7 P(2) p(3)
dynamic_general	?px ?0x ?p+ ?# ?S !Sr !#b !#n !#q
	!#r !#bn !#bq !#br !#nq !#nr !#qr
dynamic_solution	!-R !-k !-Q !-k !-Q !-Rk !-kQ !-Qk !-kQ
	!xR !xp !xRp
	!k>R !R>k !Q>k !p>Q !Q>r ...

Fig. 2. Text encoding of a tactical position in Fig. 1c.

3.1 Static features

The static part of the encoding includes information about the positions of pieces on the board, structural relationships between pieces and pawn structures present in the position.

The implementation is based on previous work on similar position retrieval [5] and our early work [1] and is intended to serve as a baseline on which we aim to improve by implementing encoding of dynamic features.

Piece positions and connectivity The section describing piece positions and connectivity encoding consists of three parts:

- *naive encoding* - the positions of all the pieces on the board.
- *reachable squares* - all squares reachable by pieces on the board in one move, with decreasing weight based on distance from the original position.
- *connectivity between the pieces* - the structural relationships between the pieces in the positions. For each piece it is recorded which other pieces it attacks, defends or attacks through another piece (*X-ray attack*).

Pawn structures For the static part of the encoding, we also use pawn structure detection algorithms to detect the following pawn structures in the position and encode them into terms: isolated pawns, (protected) passed pawns, backward pawns, doubled pawns and pawn chains.

3.2 Dynamic features

In the dynamic part of the encoding, we focus more on the solution of the tactical problem and try to grasp the motif behind it.

We first encode some general features of the solution and then add more specific terms describing the sequence of moves. The types of pieces that are moved or captured in the move sequence and the interactions between pieces are described in the encodings. We also try to identify piece sacrifices in the solution sequence. We are mainly interested in the motifs that occur in the solution and make the encodings of the features independent of where exactly on the chessboard they occur. For this reason, we do not include any exact positions of the pieces in the textual terms, but only describe the pieces by their types.

General dynamic features In this part we encode some basic features of the solution move sequence that can help us determine similarity. We use a single term for each of the following features if it holds for the solution:

- *?px* - the player captures a piece in at least one of the moves
- *?ox* - the opponent captures a piece in at least one of the moves
- *?+* - the player gives a check at least once during the sequence
- *?=* - the player promotes a pawn in at least one of the moves
- *?S* - the player sacrifices one or more pieces
- *?#* - the solution ends with a checkmate
- *?1/2* - the solution ends in a draw

Solution sequence features In this section we encode information about the solution move sequence. The encoding includes a term for each:

- type of piece moved: *!-{piece symbol}*
- type of piece captured: *!x{piece symbol}*
- attack between pieces that occurs during the solution: *!{attacking piece symbol}>{attacked piece symbol}*
- type of piece sacrificed: *!S{piece symbol}*
- type of piece involved in checkmate: *!#{piece symbol}*

We count a piece as involved in checkmate if it is attacking either the king directly or any of the squares adjacent to the king. To include information about the order of moves and captures we also include a term for each two consecutive moves and captures in the solution. We also include a term for each pair of pieces involved in checkmate to capture more specific combinations of pieces.

4 Building a database of tactical puzzles

To build a system that can recommend relevant tactical problems for any given game, a large enough collection of tactics to draw from is as important as the search algorithm itself. In our experiments, we used the above method to obtain 46,370 tactics. We developed our own method to automatically generate tactical puzzles from a collection of games that have the following properties.

- To find the most relevant candidate positions, we analyse sequences of three positions. We look for the pattern where the first player makes a blunder (the engine's rating changes by more than a certain threshold), which is immediately followed by another blunder by the second player. This means that the correct move in the position is probably not obvious, giving us a potentially interesting tactical problem. The change in evaluation must be significant and effectively change the theoretical game value.
- We analyse candidate positions with a chess engine to find the optimal line of play and other relevant moves in order to determine whether they are suitable as tactical problems and to find solutions to them. We also determine the correct length of a solution, which is another difficult problem.
- Finally, we perform an additional filtering step. Namely, the solution must be the only unambiguously winning move sequence from the initial position, the player must gain a clear material advantage or checkmate the opponent, and the final position must be stable.

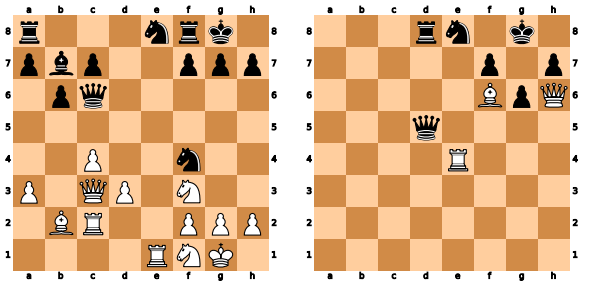
5 Evaluation

To evaluate the effectiveness of our approach, we conducted three experiments. In the first experiment, we took a large number of pairs of similar tactical problems from a renowned chess tactics training course. The pairs of similar training examples in this course were determined by chess experts. The task of our program was to find the pairs based on similarity calculations. In the second experiment, our program picked out the most similar chess tactics to the queried chess tactics and a chess expert was asked to explain the reasons for the similarity. In the third experiment, a group of experts was asked to find the most similar chess tactics to the queried chess tactics and the results were compared to the results of the program.

5.1 Matching pairs of puzzles from a chess training course

In the first experiment, we use a number of problems that we have collected from the Chess Tactics Art (CT-ART 6.0) training course. Many puzzles in this course consist of pairs of positions: one is taken from a real game, another represents a simplified version where the same tactical motif usually appears on a smaller 5×5 board. This fact allowed us to obtain a set of position pairs that were considered similar by human experts. We manually checked the puzzles and verified the similarity between the solutions of the individual problem pairs. A total of 400 pairs were collected for the test data set.

An example of such a pair is shown in Figure 3. The solution to both problems is to sacrifice the rook on the e-file to remove the defender of the g7 square, resulting in checkmate with the queen. The solution in the simplified problem contains the same motif, but there are much fewer pieces, so the solution is generally easier for the students to find.



static	30.85
dynamic	53.30
total	**84.15**
r<ne8	10.90
R>ne8	4.62
!xnR	4.35
!#bq	4.03
!-rQ	3.97
...	

Fig. 3. A pair of tactical problems from the data set: base problem (left) and simplified problem (right). The solution is in both cases 1.♖xe8+ followed by 2.♕g7 checkmate.

We first build an index using the simplified version of the problem from each pair, then perform a query on the index with each of the regular problems. For each query we record the rank of the matching position in the results and calculate how often the matching position appears as the top result or within the first N results.

We tested the search accuracy using the following feature subsets: all static features, all dynamic features and all features combined. All runs use the default BM25 parameters $k_1 = 1.2$ and $b = 0.75$ and all included feature sets are weighted equally. The results are presented in Table 1.

	Accuracy		
Feature set	top-1	top-5	top-10
all static features	0.252	0.370	0.433
all dynamic features	0.418	0.652	0.761
all features	**0.481**	**0.736**	**0.814**

Table 1. Success rates for different configurations.

Using either static or dynamic features only does not yield the best results. The results are significantly improved when both static and dynamic features are combined. This shows that each set of features covers a different aspect of the tactic, both of which need to be considered when determining similarity.

5.2 Chess expert's explanations of similarity

In the second experiment, we selected 10 contextually different chess tactical problems and then automatically retrieved 5 most similar positions for each of them from a large database of 46,370 tactical problems constructed from the lichess.org game database. Building the index took about 2 minutes (it only needs to be done once), and retrieval was fast: only about 4 seconds.

The resulting most similar positions were shown to a chess expert. The expert was asked to comment on the reasons for the similarity of the resulting problems

				Expert 1		Expert 2		Expert 3	
ID	Top score	Avg. score	SD	Score	Rank	Score	Rank	Score	Rank
1	63.42	15.41	11.04	63.42	1	63.42	1	63.42	1
2	82.09	15.60	17.13	81.47	2(1)	81.47	2(1)	81.47	2(1)
3	72.13	22.19	15.56	60.67	2(2)	60.67	2(2)	60.67	2(2)
4	72.32	15.72	13.46	72.32	1	72.32	1	72.32	1
5	91.58	22.55	17.43	89.17	2(1)	89.17	2(1)	89.17	2(1)
6	78.05	16.08	15.20	78.05	1	78.05	1	78.05	1
7	72.67	16.07	13.04	72.67	1	72.67	1	72.67	1
8	69.03	23.22	14.55	69.03	1	69.03	1	69.03	1
9	78.77	24.69	14.54	78.77	1	78.77	1	61.58	2(2)
10	70.37	16.10	13.23	70.37	1	25.72	6(11)	70.37	1

Table 2. The results of the experiment with three chess experts.

with the original query positions, taking into account both static and dynamic aspects. The expert was able to explain the similarity in 48 out of 50 problems. Overall, the expert praised the program's ability to detect dynamic similarity of positions, even if the initial positions differ significantly.

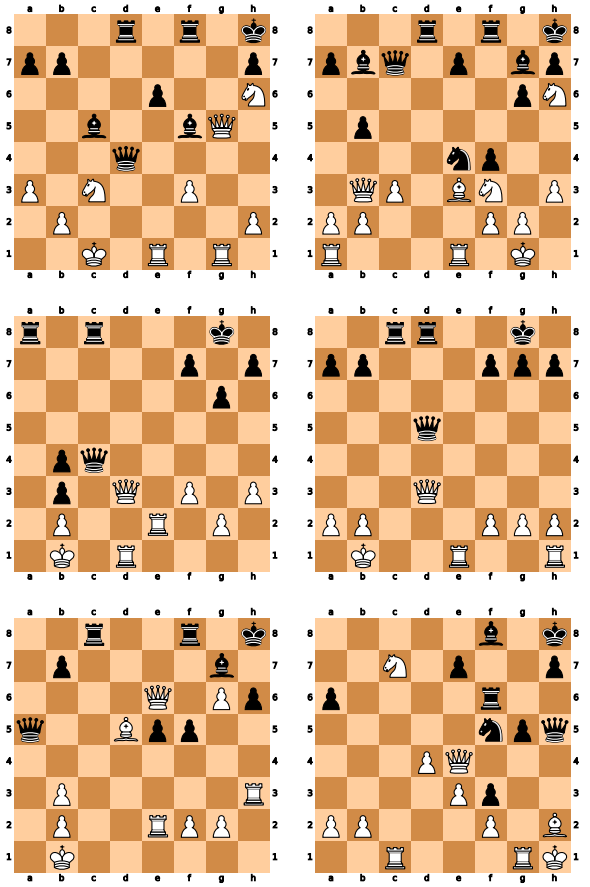
5.3 Comparison with a group of chess experts

In the third experiment, we compared the choice of the most similar tactics according to the program with the choice according to three chess experts.

We first selected 20 contextually distinct chess tactical problems and then automatically searched for three similar chess tactical problems using (1) static features only, (2) dynamic features only, and (3) all features. Half of these contextually different chess tactics were presented to a group of chess experts. The found chess tactics were presented to the experts in another database. In this database, 7 duplicates occurred because the program retrieved the same chess tactics with dynamic features or with all features. After removing the duplicates, 53 tactics remained for comparison. The experts' task was to look at the original 10 tactics and find the most similar tactic for each of them in the database of 53 tactics retrieved by the program.

The results of the comparison are shown in Table 2. Each row in the table first shows ID of a query tactic and the results according to the program: the highest score of the most similar tactics, the average score of all tactics in terms of similarity to the query item, and the standard deviation of these scores. Then the results of the three experts are displayed: the program's score of the most similar tactic according to a chess expert and the program's rank of this tactic among all 53 candidate tactics. In the cases where the experts did not choose the same most similar tactic as the program, we also show the rank of the tactic according to its dynamic score.

The results show that all three experts strongly agreed not only with the program's choices, but also with the choices of the other experts. With only one exception, the experts always chose the most similar or second most similar



1.♕g8+ ♔xg8
2.♘f7#

1.♕g8+ ♔xg8
2.♘f7#

static	11.56
dynamic	51.87
total	**63.42**

1.♖e8+ ♖xe8
2.♕xd5 +-

1.♖e8+ ♖xe8
2.♕xc4 +-

static	21.45
dynamic	51.22
total	**72.67**

1...♕xa1+ 2.♔xa1 ♖c1+ 3.♔a2 ♖a6+ 4.♕a6 ♖xa6#

1...♕xh2+ 2.♔xh2 ♖h6+ 3.♕h4 ♖xh4#

static	5.26
dynamic	63.77
total	**69.03**

Fig. 4. Similar tactics with corresponding solutions and scores.

tactic according to the program. The exception was when the second expert chose a tactic that was similar to the query tactic with ID 10. But even in this case, the expert pointed to the most similar tactic according to the program as an alternative (which also shows that the decision was not easy).

We also found that the dynamic part of the score played the most important role in the evaluation of similarity by both the program and the experts. When searching for similar tactics to query tactics with IDs 2 and 5, all three experts chose the tactics that had the highest dynamic score according to the program. Moreover, the dynamic score accounted on average for 77% (SD =0.11) of the total score of the most similar tactics according to the program and even 88% (SD =0.10) of the total score of the most similar tactics according to the majority vote of the experts.

Figure 4 shows three examples of the query tactics (the diagrams on the left) and the most similar tactics (the diagrams on the right) according to the program and also according to all three experts (the IDs of the tactics are 1, 7, and 8, respectively). The third example is particularly interesting: all three

experts agreed that this was the most similar tactic among all 53 candidate tactics, although the tactical motif occurs on a completely different part of the board. From the program's evaluation of similarity, we can see that the dynamic part of the evaluation contributed the most to the overall score.

6 Conclusion

We presented a novel method for retrieval of similar chess positions, which takes into account not only static similarity due to the arrangement of the chess pieces, but also dynamic similarity based on the recognition of chess motifs and dynamic, tactical aspects of position similarity. We also designed and implemented a method to automatically generate tactical puzzles from a collection of games.

The method for similar position retrieval was put to test in three experiments. The first experiment emphasised the importance of including both static and dynamic features for successful detection of similar chess motifs. In the second experiment we demonstrated the efficiency of the program on a large database of tactical problems generated from online chess games. A chess expert was able to explain the similarity in the vast majority of the retrieved problems and praised the program's ability to detect dynamic similarity of positions even if the initial positions differ significantly. The results of the third experiment showed that all three experts are congruent not only with the choices of the program, but also with the choices of each other. Importantly, the experiments clearly demonstrated that the dynamic part of the score played the most important role in the evaluation of similarity by both the program and the experts.

The resulting program can be useful for the automatic generation of instructive examples for chess training. Our approach is certainly not limited to chess.

References

1. Bizjak, M., Guid, M.: Towards automatic recognition of similar chess motifs. In: Proceedings of Slovenian Conference on Artificial Intelligence 2020. pp. 11–14 (2020)
2. Chess Informant: Encyclopedia of Chess Combinations. Chess Informant (2014)
3. Costeff, G.: The Chess Query Language: CQL. ICGA Journal **27**(4), 217–225 (2004)
4. Dvoretsky, M., Yusupov, A.: Secrets of Chess Training. Edition Olms (2006)
5. Ganguly, D., Leveling, J., Jones, G.J.: Retrieval of similar chess positions. In: Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval. pp. 687–696. ACM (2014)
6. Guid, M., Možina, M., Krivec, J., Sadikov, A., Bratko, I.: Learning positional features for annotating chess games: A case study. In: International Conference on Computers and Games. pp. 192–204. Springer (2008)
7. Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M., et al.: Okapi at TREC-3. Nist Special Publication Sp **109**, 109 (1995)
8. Woolf, B.P.: Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning. Morgan Kaufmann (2010)
9. Zloof, M.M.: Query-by-example: the invocation and definition of tables and forms. In: Proceedings of the 1st International Conference on Very Large Data Bases. pp. 1–24 (1975)