Retrieval of Similar Chess Positions

Debasis Ganguly Johannes Leveling Gareth J. F. Jones
School of Computing, Centre for Next Generation Localisation
Dublin City University, Dublin 9, Ireland
{dganguly, jleveling, gjones}@computing.dcu.ie

ABSTRACT

We address the problem of retrieving chess game positions similar to a given query position from a collection of archived chess games. We investigate this problem from an information retrieval (IR) perspective. The advantage of our proposed IR-based approach is that it allows using the standard inverted organization of stored chess positions, leading to an efficient retrieval. Moreover, in contrast to retrieving exactly identical board positions, the IR-based approach is able to provide *approximate search* functionality. In order to define the *similarity* between two chess board positions, we encode each game state with a textual representation. This textual encoding is designed to represent the position, reachability and the connectivity between chess pieces. Due to the absence of a standard IR dataset that can be used for this search task, a new evaluation benchmark dataset was constructed comprising of documents (chess positions) from a freely available chess game archive. Experiments conducted on this dataset demonstrate that our proposed method of similarity computation, which takes into account a combination of the mobility and the connectivities between the chess pieces, performs well on the search task, achieving MAP and nDCG values of 0.4233 and 0.6922 respectively.

Categories and Subject Descriptors

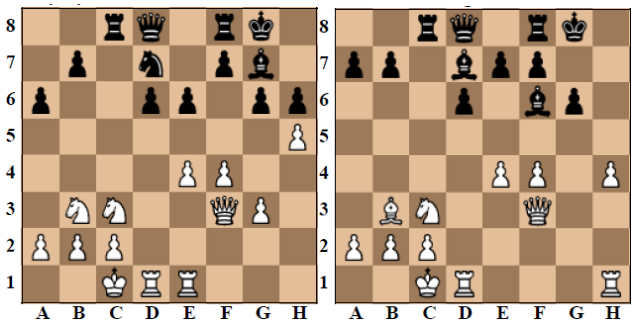
H.3.1 [**INFORMATION STORAGE AND RETRIEVAL**]: Content Analysis and Indexing—*Abstracting methods*

Keywords

Query by Example, Similar Chess Positions

1. INTRODUCTION

It is beneficial for a chess player during a live game to know whether previously archived chess games lead to positions approximately similar to the position considered at present. A knowledge about the successive moves from these retrieved games can then be used to devise a winning or a game saving strategy in general, and even decide a sequence of the next best moves in particular. Kotov, the Soviet chess grand master, opines that if one can recall similar positions from earlier games, then it is "easier to reach an assessment of how things stand, and to hit upon the correct plan or analyze variations" [8].



(a) Karjakin vs. Nakamura, position after move 20. (b) Zagorsek vs. Zirkelbach, position after move 37.

Figure 1: Two similar (not identical) chess positions.

Existing chess search systems equipped with a query-by-example (QBE) [19] search interface are limited to search only the *exact* matches in response to a given query position. However, the strict requirement of an exact match may fail to retrieve any matching board position This is because it is highly unlikely for a position in the middle or end stages of a game to exactly match another position from a previously played game due to the massive combinatorial state space of a chess game (the number of possible board positions in a chess game has been estimated to be 10^{43} [16]).

To illustrate with an example, let us take the board position arising after the 20^{th} move in a game between Karjakin vs. Nakamura as a sample input query shown in Figure 1a.[1] The online search interface *365Chess*[2] performs a known-item search by retrieving this very position from this game itself. It however fails to retrieve a very similar position as shown in Figure 1b occurring in a different game. In addition to the obvious visual similarity with regard to the position of the chess pieces, one can also notice a few other thematic or structural similarities, e.g. the black bishop plays an important defensive role on black's kingside, and that the white

[1]Section 3 briefly reviews chess notations and terminologies.
[2]http://www.365chess.com/search_position.php

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.

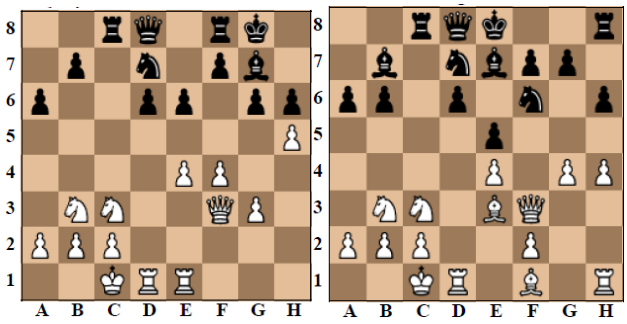
http://dx.doi.org/10.1145/2600428.2609605 .

queen has limited reachability on the opponent's half due to the presence of two centrally located pawns etc. An interesting observation is that both games resulted in a win for white. This example clearly shows that the strategy adopted by white in the subsequent course of one of the games might have been used to devise a similar strategy in the other.

The underlying approach of a QBE chess position exact search system primarily involves storing each position of every game in a hash organized file. Retrieval is then simply a computation of the hash function of the query board position and returning the board positions from the collection which hash to the same value as the query. The hash function mostly used is the Zobrist hashing [20]. While this approach is highly efficient and scalable for very large game collections, the major disadvantage is that it does not allow provision for approximate search, the importance of which has already been discussed (c.f. Figure 1). The search system CQL[3] (Chess Query Language) alleviates this problem of exact item search by allowing for approximate matches with the help of wild-card queries [4]. However, there are three important limitations in CQL, as follows. Firstly, in CQL a user has to meticulously formulate the query text in a complex query language for a given board position. Secondly, in order to allow for more relaxed matches, instead of using a simple hash-based lookup, CQL employs a *position filter* constructed from the query to each position of every game in the collection and reports the ones which evaluate to true when applied to this query position filter. For example, the CQL query "(position [RQ]b2 bg8)" matches any position with a white rook or a white queen on $b2$ and a black bishop on $g8$. For reporting the results against this query, CQL checks if there is a white rook or a white queen on the square $b2$ and a black bishop on the square $g8$ and reports the ones which satisfy this constraint. This implementation of the matching phase in CQL leads to a slow runtime even for moderate sized game databases. The third limitation of CQL is that it suffers from the classical problem of Boolean retrieval, i.e. it has no way of ranking the search results because it applies a Boolean filter instead of computing a similarity score. The order in which relevant game positions are presented to the searcher can save a considerable amount of time in the analysis of the current query position. Returning to our example query of Figure 1a (reproduced in Figure 2a), the position shown in Figure 2b, in spite of being visually quite similar to the query, may not quite provide useful insights into the analysis of the current game position. This can be seen from a few contrasting observations such as: i) the strategy for white in Figure 2b would be to develop attack along the centre of the board because the king is not castled and stays in the $e8$ square, whereas the most rewarding line of attack for white in the query position would be along the kingside; and ii) the white queen has more mobility along the f file in comparison to the white queen of the query position.

To alleviate the above limitations of existing chess position search systems, this paper approaches the problem from an IR perspective, which can, in theory, overcome the problems of the existing approaches. This is because a standard IR-based approach a) is not limited to identical match finding such as in the *365Chess* interface; b) uses a standard inverted list file structure to efficiently constitute a list of candidate documents for retrieval in contrast to the sequential approach of CQL; and c) involves the computation of a similarity score between the documents in the collection and the query so that the documents in the retrieved list can be ranked unlike an arbitrarily ordered list of results returned by CQL. The problem however cannot simply be solved directly using an off-the-shelf IR system as a black-box.

[3]http://www.rbnn.com/cql/



(a) Karjakin vs. Nakamura, after move 20.

(b) Glantz vs. Bekker Jensen, after move 24.

Figure 2: An example of a chess position which may not be relevant to the analysis of the query position.

The critical questions in this case are: i) How can a chess board position be encoded as a string the constituent terms of which can be indexed by a standard text-based IR system? ii) How can the structural relationship between the chess pieces be represented in the index, and how can the similarity computation function make use of this during the retrieval, e.g. with reference to the query position shown in Figure 1a, how can the fact that the restricted mobility of the white queen due to the presence of the white pawns at $e4$ and $f4$, be utilized to predict that the position (in Figure) 1b is more relevant to the query than the position 2b?

In the subsequent sections of this paper, we describe how these questions are addressed towards developing an IR-based chess position retrieval system. The rest of the paper is organized as follows. Section 2 surveys previous work related to this research. In Section 3, we provide a short introduction to the chess terminologies and notations relevant for reading the rest of the paper. Section 4 formally defines the chess position search problem. Section 5 discusses various approaches towards encoding a chess board position as a text string so that the similarity between two chess board positions can be computed with the help of a standard IR model. This is followed by Section 6 which describes the complete algorithm for the search problem. Section 7 describes the experiments conducted to evaluate our proposed approach. Finally, Section 8 concludes the paper by summarizing the observations and providing directions for future research and generalizing this approach for other domains.

2. RELATED WORK

Due to the novelty of the search task itself, the authors, to the best of their knowledge, are not aware of any previous IR-based approach towards solving the chess position search problem. The search problem, however, is closely related to the problem of retrieving approximately matching entities (typically non-text) from a collection, given a query entity (typically also non-text) as an example. Exam-

ples of this class of search tasks include content-based image search [18], mathematical equations search [7], searching for the least edit distance reference sentence for example-based machine translation (EBMT) [10], etc. These search tasks differ widely from standard text search in the following ways. Firstly, in contrast to key-word type queries used in ad-hoc IR, these search problems often use the QBE paradigm [19] to avoid the user inconvenience of manual query formulation. As a result of this the queries themselves are comparable in size to the documents in the collection. Secondly, since the queries and the documents are comparable in size, the similarity measure often used in these search tasks is that of the edit distance (inverse similarity), i.e. the minimum number of operations needed to transform a document to the query and vice-versa. For example, the similarity between two mathematical expressions can be computed by the inverse of the edit distance between their parse tree structures [7]. The edit distance metric can, in principle, be applied to the chess position search problem as well, where the distance (inverse similarity) between two positions could be computed by the number of operations required to transform one board position to another. However, the edit distance based approach, although effective in practice, may lead to inefficient retrieval because the edit distance values cannot be computed by utilizing the standard inverted list indexing framework. Consequently, the main disadvantage of the edit distance method is the inefficiency involved in computing the edit distance values between each document in the collection and the given query [7]. A solution which overcomes this inefficiency was proposed in [10]. Instead of computing the true edit distances this method approximates the relative ordering of documents by edit distance values with the help of an IR-based approach. In the context of mathematical equation retrieval, [12] proposed a text-based encoding of mathematical expressions to the accomplish search task by a standard inverted index based IR approach, which is more efficient compared to computing edit distances between each mathematical expression in the collection and the query. We propose a similar textual encoding for representing the chess board positions so that the board pos they can be indexed and retrieved under the framework of a standard text retrieval system.

In relation to the chess game playing research, we find the following relevant to our work in this paper. De Sa [15] applies logistic regression to predict the likelihood of a win from a given board position. Cognitive aspects of "similarity" between two chess positions have been reported in [17]. Automatic chess playing programs select the next best move by selecting the best branch in the game tree generated from the present board state [2]. The evaluation function of a board state is estimated by making use of the mobility of and the connectivity between the chess pieces [2], which we also use in our work to compute the similarity between two chess positions.

3. BACKGROUND

This section provides a brief introduction to the chess specific terminologies such as the algebraic notation and the most widely used file formats for storing chess games.

Chess Algebraic Notation. Chess is a two player game played on a 8x8 board. Each square of the chess board can thus be represented by a coordinate pair. The standard algebraic notation (SAN) prescribes the use of a letter and a number for this. The columns from white's left are numbered from a to h, whereas the rows are marked from 1 to 8 bottom-up. The coordinate of the bottom left square (the origin in this coordinate system) is thus a1. For example, the white king of Figure 2a is placed at the square $c1$.

PGN Notation. The PGN (Portable Game Notation) [5] format is used to encode the moves of a chess game. The string encoding of a move comprises of the piece name being moved (in case of a pawn the piece name is empty) and the coordinate of the destination square of the move. Taking an opponent piece as a part of the move is represented with an additional character 'X' before the destination square coordinate. For example, the next three moves from the position depicted in Figure 2a are "20... Bxc3 21. bxc3 Qf6", which implies black takes the knight on the $c3$ square with his bishop, following which white takes black's bishop with his pawn after which black moves his queen to the $f6$ square. In our work, we use the PGN encoded move sequences of a game to compute and store the textual encoding of each intermediate board position matrix in an index.

FEN Notation. In contrast to the PGN notation (which encodes a move), the FEN (Forsyth-Edwards Notation) notation encodes a particular position encountered during a chess game. A FEN string can be decoded into a board position matrix. In our search system, a user can specify a query with the help of a FEN encoded string.

4. PROBLEM FORMULATION

In this section, we define the chess position retrieval problem as an ad-hoc search problem. The task of the chess search problem is to retrieve a list of chess positions most similar to a given query position. The intention is to use the associated information from similar positions in the analysis of the current game position.

The underlying scenario of this search problem is somewhat analogous to the content-based image retrieval (CBIR) problem, where the intention is to retrieve the top-most similar images, given a query image [18]. Conceptually speaking, the chess position search problem can be visualized as a CBIR problem, in which the intention is to retrieve a ranked list of snapshot images of chess positions ordered by decreasing values of similarities with respect to the query snapshot image. However, one important aspect in which the chess position search differs from other standard search tasks is due to the presence of an additional constraint that it is desirable to retrieve only a single board position (the one which best matches the query) from each game, instead of retrieving more than one position from a single game. The search objective in this case is somewhat analogous to detecting the best entry point (the best matching chess position) from which a user should start reading a document (a chess game) so as to satisfy his information need [13].

It is reasonable to apply this additional constraint for the chess position search because sets of consecutive positions in the same game tend to be more similar to each other than to other positions in different games. Consequently, retrieving this whole set of consecutive positions from a single game does not allow similar positions from other games to be retrieved at top ranks. According to the objective of the search task, it is desirable to retrieve a single best matching position from each unique game, so that the current game can then be analyzed by extracting information from the subsequent moves following the best matching position of this

game. Each retrievable unit in this case thus has to be associated with a parent element, that is the game in which this position has occurred, and the system must ensure that the parent of each retrieved position is unique. We now define the chess position search problem formally as follows.

Given a chess game collection of N chess games, where each game $g^i = \{g^i_1, \ldots, g^i_{m^i}\}, i = 1 \ldots N$ (each game g^i is a list of m^i positions), a query chess position q from a game Q, and a similarity function $sim(p_1, p_2)$ denoting the similarity between two chess positions p_1 and p_2, return a list of chess positions $L = \bigcup_{i=1}^{K} g^i_j$ such that:

1. $g^k \neq g^{k'}\ \forall k, k' \in \{1, .., K\}$, i.e. all games are distinct from each other,
2. $sim(g^i_j, q) \geq sim(g^i_{j'}, q)\ \forall j' \in \{1, .., m^i\} - \{j\}$, i.e. retrieve the best matching position from each game, and
3. $sim(g^i_j, q) \geq sim(g^{i'}_{j'}, q)\ \forall i' > i$, i.e. games ordered by non-increasing values of the best matching position.

The crucial part in the problem definition is designing a suitable similarity function $sim(p_1, p_2)$, which should be able to determine how similar two chess positions $p1$ and $p2$ are to each other. For computing this similarity, one needs to encode a chess position as a feature vector so that the similarity between these vectors can be computed by the standard dot product. The features themselves need to be carefully chosen so as to represent the key factors in a game position, such as the position, mobility and connectivity of the pieces [2].

5. SIMILARITY COMPUTATION

This section addresses the problem of encoding chess board positions. We first start with a naive feature encoding approach and then progressively work towards enhancing it with more contextual information.

5.1 A Naive Encoding

A very simple and naive approach of encoding a chess board state is through a feature vector $f \in \{0, 1\}^{64}$, i.e. a Boolean vector of 64 (8×8) dimensions where $f_i = 0$ if the square is empty or 1, if it is occupied. We adapt the convention that the component f_1 corresponds to the top left square, i.e. $a8$ whereas the component f_{64} corresponds to the square $h1$. Under such an encoding scheme, the chess board of Figure 3a (reproduced from Figure 1a) is $\{0, 0, 1, 1, 0, 1, 1, 0, 0, 1, \ldots\}$.

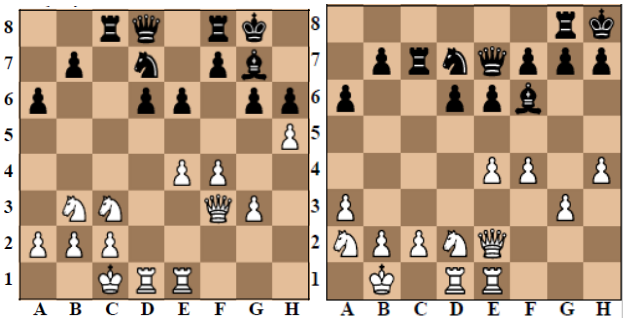
This method does not take into account the type or the colour of the pieces. It is easy to see that the dot product would falsely lead to a very large number of high similarity values based only on the occupancy factor of the squares. For example, a rook at the $c8$ square of the position in Figure 3a may match with a knight at $c8$ in another board position. An easy solution is to use a higher number of dimensions to correspond to the piece type and colour. Since there are a total of 6 piece types and 2 colours, f can be represented as a vector of 64×12 dimensions, where the i^{th} component is 1 if the square corresponding to the component is occupied by a piece of the type and the colour which the component corresponds to [15].

This notation of the feature vectors is very inconvenient to use in practice. Adopting the IR convention from now on, we can think of each dimension as a *term* representing a particular piece of a specific colour in a specific location. For example, the two terms $Nc3$ and $bg7$ indicate the presence of a white knight at $c3$ and a black bishop at $g7$, where we use uppercase characters for the white and lowercase ones for the black, respectively. A chess board feature vector can thus be represented as a document with constituent terms. An advantage of the textual representation is that chess board positions can simply be thought of as text documents and hence standard IR similarity measures, such as cosine or BM25 [14], can directly be applied on the board positions as well. The constituent terms of the textual representation of the board position of Figure 3a is represented as follows.

rc8 qd8 rf8 kg8 pb7 nd7 pf7 bg7 pa6 pd6 pe6 pg6 ph6 Ph5 Pe4 Pf4 Nb3 Nc3 Qf3 g3 a2 b2 c2 Kc1 Rd1 Re1

The problem with the naive encoding is that it does not allow for an approximate match by a relaxation of the piece positions, e.g. Figure 3b shows an artificially created board position with a *positional transformation* on a majority of the pieces. By *positional transformation*, we mean shifting a piece to another square according to the rules of the game. For example, the white queen has been moved from $f3$ to $e2$, which is a valid move. It can be seen clearly that the similarity value between the textual representation of the board positions will be considerably lower because of a large number of term mis-matches. However, this artificially constructed board position, if it had occurred in a real game, would have potentially acted as a relevant board position with the capability of providing valuable insights into the current game analysis. This is because even if the pieces have been transformed, the structural relationship between the pieces remains very similar, e.g. the mobility of the white queen is still restricted, the black bishop is still a threat on the white kingside and so on.

We now describe additional pieces of information to represent the structural relationship between the pieces of a chess board, namely the set of squares reachable by a piece and the set of pieces being attacked and defended by a piece.



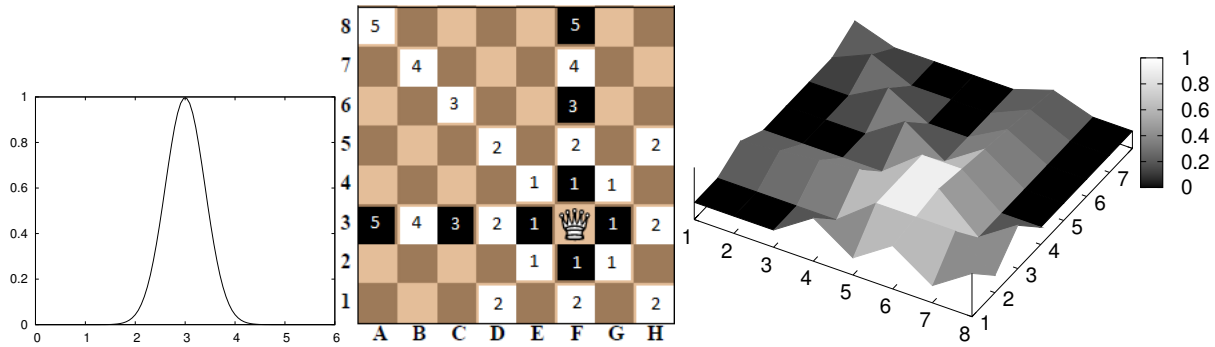
(a) Karjakin vs. Nakamura, after move 20.

(b) A transformation of the board on left.

Figure 3: An example to illustrate the limitation of exact matching inherent in naive encoding.

5.2 Reachable Squares

Informally speaking, every chess piece has got its own "active zone" which is the set of squares reachable from the current position of the piece. The role played by the piece during the game is largely restricted to this zone. For example, the set of squares reachable by the white rook on $d1$ is



(a) Weight function for term position propagation in a text document

(b) $r\text{-}closure(Q, 6, 3)$

(c) Weight matrix for $Qf3$

Figure 4: An illustration of term propagation during indexing.

$\{d2, d3, d4, d5\}$. The idea is analogous to "propagating" the influence of a term to adjacent positions in a document by a proximity-based kernel density function, which is maximum at the current position and decreases gradually with increasing distances from the current position [11]. In the case of a document, the position of a term refers to the offset (measured in terms of the number of words) from the starting word of the document. For example, Figure 4a shows that the weight function of a term is maximum at its true position (which in this example is the Gaussian curve centred at offset 3). The weights with which the term contributes in the rest of the document progressively decrease on both sides of its true position.

In our case, we apply a similar approach with some necessary modifications. In standard positional-based term indexing, a term can (in theory) exist at all positions with progressively decreasing weights. In the case of chess search, the term (which is equivalent to a chess piece) can only exist in one of the set of squares *reachable* from its current position. Assuming the bottom left corner square is $(1, 1)$ and the top right corner is $(8, 8)$, let us formally define the set of squares reachable by a piece p from its current position $(x, y) \in \{1, 8\}^2$ as follows.

$$r\text{-}closure(p, x, y) = \{(x, y)\} \cup \{(x', y') : (p, x, y) \models^R (p, x', y')\} \quad (1)$$

$(p, x, y) \models^R (p, x', y')$ denotes that the piece p can be placed from the square (x, y) to the empty square (x', y')[4]. in one move according to the rules of the game. Thus, while a positional-based text indexing approach [11] considers all integer offset positions of a term, we simply need to restrict the positions to the set $r{-}closure$ as defined in Equation 1.

Figure 4b illustrates the *r-closure* for the white queen present at the square $f3$ of the chess board of Figure 3a. The figure highlights the set of reachable squares for the piece $Qf3$. Note that in this clarifying example, we make a simplifying assumption that there are no other pieces on the board to obstruct the reachability of the queen. As a distance measure, we use the *Chebyshev distance* (also known as the *chess board distance*) has been widely used in the game of chess, and is defined as the minimum number of moves a king requires to move between the source and the destination positions. Figure 4b shows these distance values of the reachable squares measured from the square $f3$.

The weight function of a chess piece (i.e. the likelihood with which it can occur in other positions in addition to its true position) is now defined for all points in the *r-closure* set of that piece. Similar to the approach shown in Figure 4a, these weight values are derived from a function of the current position and the distance to the destination position. More specifically, for a piece p located in (x, y), we define the following weight function for each member $(p, x', y') \in r\text{-}closure(p, x, y)$.

$$w(p, x, y, x'y') = 1 - \frac{7 \times d((x, y), (x', y'))}{64} \quad (2)$$

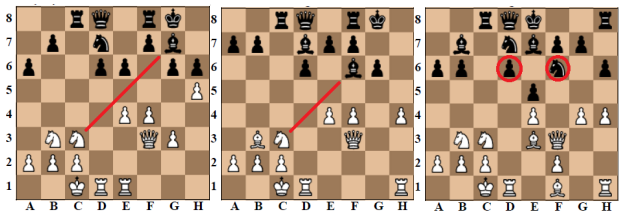
where $d((x, y), (x', y'))$ is the chess board distance between positions (x, y) and (x', y'). Note that the slope of $\frac{7}{64}$ of the weight function ensures a minimum value of $\frac{1}{8}$ at the maximum possible distance value 8, and a maximum value of 1 at distance 0, i.e. at the true position of the piece.

Figure 4 shows the plot of the weight function, defined in Equation 2. Note that the maximum intensity of the weight function can be seen by the white surface at $(6, 3)$, i.e. at the square $f3$, and that the values gradually decrease along the horizontal, vertical and diagonal lines. For squares outside the set $r\text{-}closure(Q, 6, 3)$, the weight function is zero as can be seen by the black patches of surfaces in Figure 4. The weight function that we use in our study is a linear function instead of the more complex Gaussian function used in [11], so as to avoid tuning of additional parameters.

This process of computing the reachable squares and then weighting them by the function w is performed for all pieces present in a board position. The weight function obviously depends on the piece for which it is computed because of the variation in the rules of allowable moves, e.g. the weight function plot for a rook, which only moves vertically and horizontally, is different from that of a bishop which only moves diagonally. The reachable positions of a piece along with their weights are stored as delimiter separated values in the textual representation of the document.[5] As an ex-

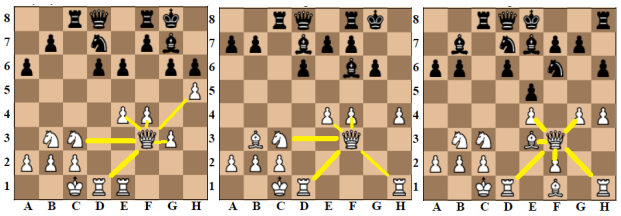
[4]Possible moves to a non-empty square following the capture of an opponent piece are not considered here. These moves are handled in Section 5.3.1

[5]Storing the weights as delimiter separated values was purely a Lucene specific implementation decision to utilize



(a) (b) (c)

Figure 5: Black dark square bishop attacks.



(a) (b) (c)

Figure 6: Defense squares of $Qf3$.

ample, we show a part of the textual encoding of our earlier example, i.e. the board of Figure 3a, as follows.

> rc8 qd8 rf8 kg8 ...
> *Qf2|0.89 Qf1|0.78 Qg2|0.89 Qh1|0.78 Qe2|0.89 Qe3|0.89 Qd3|0.78 Qg4|0.89*
> *Re2|0.89 Re3|0.78 ...*

Note that the additional terms (shown in italics) represent the reachable positions of the pieces. The advantage of this encoding is illustrated by the fact that it enables the board position depicted in Figure 3b to be retrieved from the index, due to the presence of the matching terms $Qe2$ etc. The weight function ensures that, during retrieval with a query term (say $Qf3$), a board position (document) in which the white queen is at $f3$ will get a higher score than a document where the white queen is at $e2$.

5.3 Connectivity between the pieces

In the previous section, we have seen that the limitation of the exact match can be overcome by incorporating the reachability information for each piece on the board. This section explores the integration of structural relationships between the pieces into the encoding.

5.3.1 Attack Squares

Two board positions are more likely to be similar to each other if the same piece is found to attack similar squares of the opponent. Let us illustrate this with an example. The boards of Figures 1a, 1b and 2b have been reproduced in Figures 5a, 5b and 5c respectively. One of the reasons that the board in Figure 5b is more relevant to the query board of Figure 5a than the board of Figure 5c, is that both the boards in Figure 5a and 5b have the white knight at $c3$ under a threat by the black bishops at $g7$ and $f6$ respectively (even though $f6$ and $g7$ are distinct squares, $f6$ is reachable from $g7$). On the other hand, the bishop of the board in Figure 5c is passive (blocked by two of its own pieces) not attacking the knight at $c3$ or any other of white's pieces. Such observations about the active or passive nature of the pieces can be crucial in determining the relevance of board positions in game analysis [9].

Following the above reasoning, the textual encoding of the board state is thus appended with terms corresponding to a list of attacking positions for each piece. More formally speaking, we define

$$a\text{-}closure(p,x,y) = \{(p,p',x',y') : (p,x,y) \models^{A} (p',x',y')\} \quad (3)$$

the class "DelimitedPayloadTokenFilter". See Section 6 for the implementation details.

Hence, the $a\text{-}closure(p,x,y)$ of a piece p at (x,y) contains all those tuples (p,p',x',y') such that the piece p attacks an opponent piece p' at square (x',y'). For example, the set $a\text{-}closure(b,g,7)$ of Figure 5a comprises of the tuple $(b,N,c,3)$. Note that we do not put the source position in the member tuples of the set $a\text{-}closure$ to allow for the flexibility that the corresponding piece in a retrieved board position can attack an opponent piece from one of the squares *reachable* from the position which that piece occupies in the query. For example, storing the square $g7$ would not have allowed a match between the positions of Figures 5a and 5b.

A part of the textual encoding of Figure 5a, with additional terms corresponding to those of the attacking squares, is shown in italics as follows (we use an arbitrary delimiter ">" to separate the current piece and the attacked piece).

> rc8 qd8 rf8 kg8 ...
> Qf2|0.89 Qf1|0.78 Qg2|0.89 Qh1|0.78 Qe2|0.89 Qe3|0.89 Qd3|0.78 Qg4|0.89 ...
> *b>Nc3 r>Nc3 p>Ph5 R>pd6 ...*

5.3.2 Defense Squares

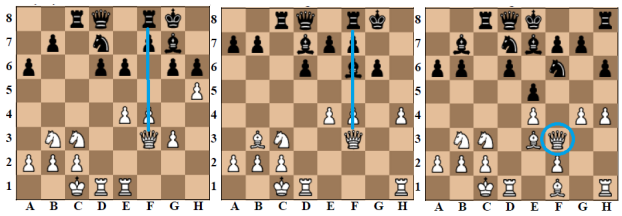
Analogous to the attacking squares, for each piece we also keep track of the pieces it defends. This again helps to determine the critical pieces and their connectivity across two board positions that need to be matched. For example, both the white queens at $f3$ of boards Figure 6a and Figure 6b defend a knight, a rook and two pawns at identical positions, whereas the queen in Figure 6c has only two matching defended pieces ($Rd1$ and $Pe4$) with respect to the query (Figure 6a). Similar to the $a\text{-}closure$ we formally define the set of defended squares as follows.

$$d\text{-}closure(p,x,y) = \{(p,p',x',y') : (p,x,y) \models^{D} (p',x',y')\} \quad (4)$$

Hence, the $d\text{-}closure(p,x,y)$ of a piece p at (x,y) contains all those tuples (p,p',x',y') such that the piece p defends an opponent piece p' at square (x',y').

5.3.3 Ray-Attack Squares

In chess, a *ray attack* or an *X-ray attack* [3] occurs when a piece attacks an opponent piece "through" other pieces in its way. Although the opponent piece cannot be taken in the next move, but still these situations often lead to threatening attacks on the opponent piece being ray-attacked. For example, $rf8$ in positions of Figure 7a and 7b *ray-attacks* the white queen on $f3$. Note that $Qf3$ of board 7c is not under a ray-attack by any of the opponent pieces, as a result of which this criteron can also play a crucial part in comparing the relevance of one board position to the query over another. Similar to the attack and defense closures of



(a) (b) (c)

Figure 7: Ray-attacks on $Qf3$.

Equations 3 and 4 respectively, the ray-attacking squares of a piece are defined as follows.

$$x\text{-}closure(p,x,y) = \{(p,p',x',y') : (p,x,y) \models^X (p',x',y')\} \quad (5)$$

A part of the textual representation for the board position of our earlier example, with additional terms corresponding to the defense and the ray-attack squares (using delimiters "<" and "=" for defense and ray-attack respectively), is shown below.

rc8 qd8 rf8 kg8 ...
Qf2|0.89 Qf1|0.78 Qg2|0.89 Qh1|0.78 Qe2|0.89 Qe3|0.89 ...
b>Nc3 r>Nc3 p>Ph5 R>pd6 ...
Q<Pe4 Q<Pf4 Q<Pg3 Q<Nc3 Q<Rd1 ...
R=qd8 R=nd7 R=pd6 r=Qf3 r=Pf4 ...

6. IMPLEMENTATION DETAILS

In this section, we combine the ideas of Sections 4 and 5 to devise an algorithm for indexing the chess positions for a given collection of chess games. Our algorithm can use a standard text IR system to store the game positions in an inverted file structure. The pseudo-code for indexing and retrieval is presented in Algorithm 1. In Step 3 of Algorithm 1, the parameter *numskip* is used to skip a number of initial moves. This is important because the opening of a chess game follows a preset sequence of moves chosen from a few standard opening move sequences. There is thus a high probability of board positions to be exactly identical to one another during the opening stages of a game, which is not particularly interesting for this search task. Moreover, not skipping the first few moves can contribute to an unnecessary increase in the size of the index due to the presence of a large number of pieces during the initial stages of a game. In particular, we set *numskip* to 12 in our experiments, i.e. we skip the first 12 rounds of moves. Each board position after a move in a game is encoded as a string constituted of a list of encodings for each piece computed in the loop from Steps 7-16. The storage of a text encoded board position in Step 18 of the algorithm is achieved by the use of Lucene (version 4.6)[6], which is a freely available widely used text retrieval system.

The query to the system is an FEN encoding of a board position. The FEN encoding of the query board state is transformed to an internal encoding (achieved by executing the procedure "Retrieve" of Algorithm 1). This query string is then used to retrieve similar board positions from the index. The query board position, in contrast to an indexed document board position, does not need to contain the terms corresponding to the reachable squares. To illustrate with an example, it is possible to retrieve a document (say D) containing the term $Qf3$ in response to a query term $Qf2$ (because D also contains the term $Qf2$ along with an associated weight of 0.89 computed by Equation 2). The implication is that it is possible to conduct an approximate search, e.g. retrieve a board position where the white queen is at $f3$ in response to a query position where the white queen is at $f2$, without needing to add the query term $Qf3$.

The IR model selected for use in retrieval was BM25 [14] with the default parameters of Lucene[7], which is $K = 1.2$ and $b = 0.75$. Retrieval is carried out in two steps. Firstly, we retrieve a list of 1000 board positions by using the BM25 model. Secondly, we refine this list by retaining the best matching board position from each game. This ensures that consecutive board positions from a single game (which are likely to be similar to each other) are not reported in the final result-list.

Algorithm 1 Chess Games Indexing and Retrieval

```
1: procedure IndexGames(G)
2:   for each game g ∈ G do
3:     Skip the first numskip moves.
4:     for each move m ∈ moves(G) do
5:       brdenc ← ∅
6:       board ← board matrix on applying move m
7:       for each piece (p,x,y) ∈ board do
8:         brdenc ← brdenc ∪ (p,x,y)
9:         for each (p,x',y',w) ∈ r-closure(p,x,y) do
10:          brdenc ← brdenc ∪ (p,x',y',w)          ▷ Eq. 2
11:        end for
12:        for each α ∈ {a,d,x} do
13:          for each (p,p',x',y') ∈ α-closure(p,x,y) do
14:            brdenc ← brdenc ∪ (p,p',x',y')      ▷ Eq. 3, 4, 5
15:          end for
16:        end for
17:      end for
18:      Add document brdenc to index
19:    end for
20:  end for
21: end procedure
22: procedure Retrieve(Q)
23:   brdenc ← ∅
24:   board ← board matrix of Q
25:   for each piece (p,x,y) ∈ board do
26:     brdenc ← brdenc ∪ (p,x,y)
27:     for each α ∈ {a,d,x} do
28:       for each (p,p',x',y') ∈ α-closure(p,x,y) do
29:         brdenc ← brdenc ∪ (p,p',x',y')
30:       end for
31:     end for
32:   end for
33:   Retrieve from index by executing query brdenc
34: end procedure
```

7. EMPIRICAL EVALUATION

In this section, we describe the experiments conducted to evaluate our proposed approach of chess position retrieval.

7.1 Dataset Construction

A standard IR test collection is comprised of three components, namely a document collection, a set of queries and a set of relevance judgments for these queries. Freely available chess game archives in the PGN format can be used as the document collection in this particular search task. However,

[6]https://lucene.apache.org/core/4_6_0/index.html

[7]http://lucene.apache.org/core/4_6_0/core/org/apache/lucene/search/similarities/BM25Similarity.html

Table 1: Document collection statistics

Collection	# Games	# Players	# Docs.	# Terms
ICOfY 13.2.1	96,397	32,366	7,587,963	13,580,212

Table 2: Different Retrieval Approaches

	Feature Encoding Configurations			
Run Name	Pos. Wt.	Attack	Defense	Ray-Attack
TruePos	N	N	N	N
RchblePos	Y	N	N	N
Attk	Y	Y	N	N
Dfns	Y	N	Y	N
RayAttk	Y	N	N	Y
CombNoRay	Y	Y	Y	N
CombAll	Y	Y	Y	Y

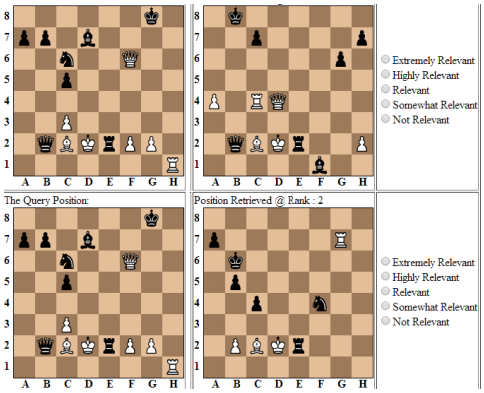


Figure 8: Screen-shot of the search results page.

due to the novelty of the search task itself, sets of relevance assessed queries are not available, to the best of our knowledge. To proceed with the evaluation, we therefore needed to construct a set of queries and relevance assessments.

7.1.1 Document Collection

As a first step towards building up the test dataset for evaluation, we indexed a set of freely available PGN game archive files named ICOfY (version 13.2.1 to 13.2.5)[8] according to the method outlined in Section 6. Each PGN file in the document collection was parsed with the help of a freeware PGN parser *pgnparse*[9]. The parameter *numskip* was set to 12, i.e. we ignored the first 12 rounds of moves from each game. Table 1 outlines the index statistics. The number of documents in this table refers to the number of chess positions indexed, which indicates that on average almost 78 positions are indexed per game. The number of unique terms refers to the number of unique piece configurations. On average each game position contributes 1.8 unique terms, which suggests that there is a significant overlap in the piece configurations across the index.

7.1.2 Setting up the Baselines

In order to test the hypothesis that the mobility and the structural relationship of the pieces can play a vital role in chess position search, we set up baseline approaches where we selectively do not use one or more of these attributes during the retrieval time similarity computation. The approaches are outlined in Table 2. The run "TruePos" uses the naive encoding of a chess board during the similarity computation, whereas the subsequent approaches aim to test the benefit of applying more information in the similarity computation, such as the weighted reachability ("RchblePos"), attacking squares information ("Attk") and so on. The last approach ("CombAll") uses a combination of all the features.

7.1.3 Queries and Search User Interface

Queries and relevance assessments were formulated by crowd-sourcing using an approach similar to [7]. The chess position retrieval system was made available as a web service so that the users could formulate arbitrary chess position queries and provide relevance assessments for each retrieved board position. Participants for this task were recruited by distributing a "call for participation" email among our colleagues, which contained detailed instructions for using the web interface. The email asked recipients to sign up for the experiment only if his familiarity with chess was higher than 2 in a scale of 1 to 5. Note that this experiment did not require the participation from professional chess players. However, a basic knowledge of chess was required for correct relevance judgments. Each participant from the set of 5 registered ones, were asked to use the web interface to search for chess positions. The interface allowed the user to play a valid sequence of chess moves in order to formulate a search query. For the purpose of the experiment however, the users were instructed to cut-and-paste sequences of moves from real games because the board positions encountered in real games, when used as queries, have somewhat higher likelihood of retrieving relevant positions from the game database than when using queries from fictitious board positions.

We used a different PGN game archive ICOfY (version 13.1.4) comprising of 3970 games to formulate our query set. Each participant was provided with unique chunks of 300 games from this file, and was asked to formulate a query by randomly selecting a sequence of moves of size at least $numskip + 1$ (which in this case is 13), because positions from the 13^{th} move onwards were indexed into the system. The fact that each participant was provided with a different chunk ensured that the set of queries were non-duplicates. Each participant contributed 5 queries each, as a result of which the evaluation set comprised of a total of 25 queries.

7.1.4 Relevance Assessments

A screen-shot of the web interface is shown in Figure 8. The system returns a ranked list of at most 200 best matching positions from unique games. The pool of 200 retrieval results was formed by a combination of different retrieval approaches outlined in Table 2. The similarity scores of the documents from each run were normalized (divided by the maximum score for that run). The pool was then constructed by selecting 200 documents with most normalized scores from the union of these runs.

In each row of the search results page, the first (left-most) column displays the query board position, while the second one displays a retrieved board position, alongside of which a five point relevance scale (ranging from non-relevant to extremely relevant) is displayed. The participants were asked to provide relevance assessments for each retrieved board

[8]http://icofy-base.de/1321PGN.7z
[9]http://sourceforge.net/projects/pgnparse/

position in response to a query. In addition to the visual similarity between two positions, the participants were instructed to also consider the usefulness of the retrieved game in the analysis of the current query position, i.e. to take into account the similarities in the key features of the positions. Note that we relied on a five-point relevance scale rather than on strict binary relevance judgments. This is because in the case of chess position search, the relevance judgments need to accommodate for the different levels of importance that a retrieved board position may have in analyzing the current game situation.

Our evaluation is based on manual judgments because it is not possible to apply a computational model of relevance for evaluation of the search effectiveness. For instance, a computational approach of relevance estimation might take into consideration the next moves and the outcome of the retrieved and the query board positions. However, such a model would assume that the next moves of the query board position are known, which in turn does not conform to the search use-case.

7.2 Results

In this section, we first report the effectiveness of our search method in terms of standard evaluation metrics. We then select the best performing approach and study the effect of varying the BM25 parameters on the retrieval effectiveness. This is followed by an investigation of the effect of the query length on retrieval behaviour.

7.2.1 Evaluation of the pooled runs

A standard metric for graded relevance judgments is the nDCG[6], which we use as one of the evaluation measures. To measure the ability of the system to retrieve documents at top ranks, we also measure nDCG@5. Additionally, we also transform each graded relevance judgment into a binary one by considering a document as relevant if the graded relevance judgment value is at least 2 and non-relevant otherwise. We then report the standard metrics of MAP and P@5 using these binary relevance values.

Note that the values for metrics such as MAP and nDCG cannot, in practice, be computed exactly because some relevant documents for a query may not exist in the pool of manually judged documents. However, in the case of chess IR, the number of board positions relevant to a query board position is expected to be small, as a result of which, it is less likely for a relevant document to be left out of the pool.

Table 3 shows that the approach "TruePos", which only uses the true positions of pieces, does not produce satisfactory results. This is expected because many relevant positions may not be retrieved due to the fact that the position

Table 3: Evaluation of the chess position search

Retrieval	Evaluation Metrics			
Method	P@5	MAP	nDCG@5	nDCG
TruePos	0.0290	0.0960	0.1008	0.1306
RchblePos	0.1229	0.1280	0.2014	0.2678
Attk	0.1441	0.2340	0.2532	0.3103
Dfns	0.1724	0.2692	0.2596	0.3404
RayAttk	0.0621	0.1385	0.1462	0.1728
CombNoRay	0.3866	0.3840	0.3809	0.6130
CombAll	**0.4233**	**0.3926**	**0.4188**	**0.6922**

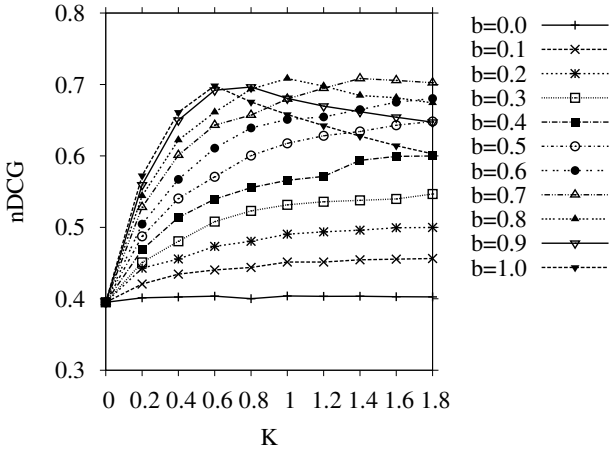


Figure 9: BM25 parameter optimization.

of each individual piece in a board position of the collection may be slightly different from that of the query (see Figure 3). Retrieval results are improved when the reachability information is included in the run "RchblePos". The results are further enhanced by the addition of connectivity information between the pieces. Both attack and defense connectivities improve the results in comparison to "RchblePos". "RayAttk" alone does not produce an improvement over "RchblePos". However, results are significantly improved when all the three connectivity types are used together in the approach "CombAll", as shown in the last row.

7.2.2 BM25 parameter tuning

The best performing run "CombAll" used the default BM25 parameter settings of Lucene, i.e. $K = 1.2$ and $b = 0.75$. Figure 9 shows the effect of varying these parameters for this run. The figure shows that the nDCG can further be improved to 0.7083 (from 0.6922 of Table 3) by using $K = 1$ and $b = 0.7$. A value of K close to zero indicates more emphasis on inverse document frequency (idf) and less on term frequency (tf), which in the case of chess position search, produces poor results. This is expected because for textual representation of a chess board, the presence of a term (tf is always 1 in this case) bears more importance than the idf.

The iso-b lines of Figure 9 show that document length normalization plays an important role in retrieving relevant documents, as can be seen from the relatively lower values of nDCG for low values of b less than 0.5. Length normalization is important here because a board position with approximately identical set of pieces with respect to the query position is more likely to be relevant rather than a longer document representing a board position containing a higher number of additional pieces not present in the query.

7.2.3 Query length effect

To determine the effect of query length on retrieval effectiveness, we categorized the queries by their lengths, which in this case refers to the number of chess pieces present in the query board position. The results are shown in Figure 10.

It can be observed that the average precision and the average nDCG values for queries in the range $[11, 13]$, i.e. the queries where the number of positions is between 11 and 13 (inclusive), are the highest indicating that our proposed

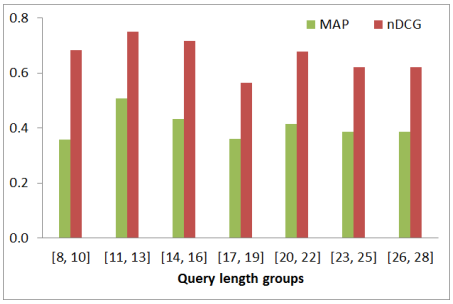


Figure 10: Average retrieval quality for each query group categorized by length (the number of pieces).

algorithm works particularly well for short queries. This finding can be explained as follows. A short query typically represents the ending stage of a game when only a small number of pieces remain. A match in a term (piece position or connectivity) towards the ending stages of a game is associated with more importance than a match in the middle stages of a game; e.g. towards the end stages of a game a match in two board positions where a king or a queen is attacked by the same piece almost certainly ensures that one position is relevant to the other.

8. CONCLUSIONS AND FUTURE WORK

This paper has explored the novel search task of retrieving top-most approximately matching chess game positions given a query position, so that the subsequent moves from the retrieved games may be useful in analyzing the current game and hence in predicting the next best move. In order to ensure effectiveness and efficiency, we have approached the search task from an IR perspective, in contrast to existing chess retrieval systems which are either restricted to provide the functionality identical matches or allow approximate retrieval of positions at the cost of heavy run-time overhead. In our IR-based approach, each position in a chess game is encoded as a set of terms, where each term is used to represent the following: a) the true position of a piece, b) the reachable positions of a piece with weights inversely proportional to the distances from its true position, and c) the structural relationships between the pieces, such as the ones being attacked and defended. Empirical evaluation of our proposed approach demonstrates that a combination of all these features produces satisfactory retrieval results. The conclusion of our study is that the chess position search problem can be solved effectively and efficiently by employing an IR-based approach. This observation can, in effect, lead to developing more effective chess playing algorithms, where the automatic process of game tree exploration for selecting the next-best move can be supplemented with information extracted from closely matching positions retrieved from previously played games. The system can also be useful in manual analysis of a current chess game.

Since a chess board position can be represented as a graph (piece configurations as nodes and inter-piece relationships as edges), we believe that the methodologies proposed in this paper can, in future, be generalized to conduct approximate search of graph objects. A practical application of graph search could be retrieval of chemical compounds which are structurally most similar to a given query compound.

Acknowledgments. This research is supported by Science Foundation Ireland (SFI) as a part of the CNGL Centre for Global Intelligent Content at DCU (Grant No: 12/CE/I2267), and Enterprise Ireland (Grant IP2012-0183) as part of the innovation partnership project CDA.

9. REFERENCES

[1] *Think like a Grandmaster*. Batsford Press, Oxford, 1995.
[2] J. Baxter, A. Tridgell, and L. Weaver. Learning to play chess using temporal differences. *Machine Learning*, 40(3):243–263, 2000.
[3] E. R. Brace. *An Illustrated Dictionary of Chess*, pages 310–310. 1977.
[4] G. Costeff. The Chess Query Language: CQL. *ICGA Journal*, 27:217–225, 2004.
[5] S. J. Edwards. Standard: Portable Game Notation Specification and Implementation Guide.
[6] K. Järvelin and J. Kekäläinen. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
[7] S. Kamali and F. W. Tompa. Retrieving documents with mathematical content. In *In Proceedings of SIGIR '13*, pages 353–362. ACM, 2013.
[8] A. A. Kotov. *Think like a Grandmaster*, chapter Introduction: An Unusual Experiment, pages 11–14. In [1], 1995.
[9] A. A. Kotov. *Think like a Grandmaster*, chapter Positional Judgement, pages 108–120. In [1], 1995.
[10] J. Leveling, D. Ganguly, S. Dandapat, and G. J. F. Jones. Approximate sentence retrieval for scalable and efficient example-based machine translation. In *Proceedings of the COLING '12*, pages 1571–1586, 2012.
[11] Y. Lv and C. Zhai. Positional language models for information retrieval. In *Proceedings of SIGIR '09*, pages 299–306, 2009.
[12] T. T. Nguyen, K. Chang, and S. C. Hui. A math-aware search engine for math question answering system. In *Proceedings of the CIKM*, pages 724–733, 2012.
[13] J. Reid, M. Lalmas, K. Finesilver, and M. Hertzum. Best Entry Points for Structured Document retrieval-Part II: Types, Usage and Effectiveness. *Inf. Process. Manage.*, 42(1):89–105, Jan. 2006.
[14] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Overview of the Third Text Retrieval Conference (TREC-3)*, pages 109–126. NIST, 1995.
[15] C. D. Sa. Classifying chess positions. Master's thesis, Stanford University, 2012.
[16] C. E. Shannon. Computer chess compendium. chapter Programming a Computer for Playing Chess, pages 2–13. Springer-Verlag New York, Inc., New York, NY, USA, 1988.
[17] P. L. Villagra and F. Jakel. Categorization and abstract similarity in chess. In *Proceedings of the COGSCI 2013*, pages 2860–2866, 2013.
[18] C. C. Yang. Content-based image retrieval: A comparison between query by example and image browsing map approaches. *J. Information Science*, 30(3):254–267, 2004.
[19] M. M. Zloof. Query-by-Example: the invocation and definition of tables and forms. In *Proceedings of the VLDB*, pages 1–24, 1975.
[20] A. L. Zobrist. A new hashing method with application for game playing. Technical Report 88, U. Wisconsin CS Department, April 1970.