ChessY: A Mathematica toolbox for the generation, visualization and analysis of positional chess graphs.

Michelle Rudolph-Lilith

SoftwareX 9 (2019) 39–43

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

ChessY: A Mathematica toolbox for the generation, visualization and analysis of positional chess graphs

Michelle Rudolph-Lilith

Unité de Neurosciences, Information et Complexité (UNIC), CNRS, 1 Ave de la Terrasse, 91198 Gif-sur-Yvette, France

ARTICLE INFO

Article history:
Received 9 August 2018
Received in revised form 17 December 2018
Accepted 17 December 2018

Keywords:
Chess
PGN parser
Graph theory
Game theory
Mathematica

ABSTRACT

The game of chess is undoubtedly one of the most popular two-player strategy board games in history, enjoyed by casual players and competing celebrated professionals alike, and serves as prototype research subject in a vast variety of fields. Although a plethora of parsers on a large number of different platforms is readily available for processing records of chess games provided in online databases, *Mathematica* remains, somewhat surprisingly, exempt. *ChessY* attempts to fill this gap, by providing a simple set of tools for handling Portable Game Notation (PGN) chess records and their translation into positional chess graphs, thus opening the door for a systematic analysis of chess games within the powerful confines of graph theory using *Mathematica*.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_142
Legal Code License	BSD-2-Clause
Code versioning system used	none
Software code languages, tools, and services used	Wolfram Mathematica
Compilation requirements, operating environments & dependencies	Mathematica v10.0 or above
If available Link to developer documentation/manual	http://mrudolphlilith.github.io/publications/Peer-ReviewedArticles/A38/A38.html
Support email for questions	rudolph@unic.cnrs-gif.fr

1. Motivation and significance

With its 64 squares, 32 pieces of 6 types, and discrete moves of pieces between squares governed by a strictly limited, arguably simple set of rules, chess can undoubtedly be viewed as a classical example of a network in graph theory [1]. Although many chess-related problems and their solutions, such as the Knight's Tour Puzzle or Queens Domination Problem, find representations within a graph-theoretical framework (e.g., see [2,3]), a systematic analysis of available datasets of chess games recorded over many decades within a graph-theoretical context remains, however, scarce and mostly restricted to visualizing various chess strategies and search algorithms (e.g., see [4] and references therein).

One reason hindering a thorough graph-theoretical analysis of chess games is certainly the inherent difficulty associated with parsing online PGN (Portable Game Notation) databases. PGN records utilize the Standard Algebraic Notation (SAN) standardized by FIDE (Fédération Internationale des Échecs; [5]) to encode positional changes during a chess game by specifying the type and target square of the piece being moved. However, such a compact representation is subject to potential ambiguities, for instance, multiple pieces can reach a given target square, which require sophisticated and often computationally demanding parsing algorithms in order to be fully resolved. Although a whole plethora of both open-source and proprietary parsers and chess game engines in a variety of programming languages is readily available (see [6] for a recent and comprehensive review), such engines focus exclusively on assessing game positions using evaluation functions not unlike that originally proposed by Shannon [7], and selecting advantageous moves through searches in vast databases of available chess games. Moreover, the API of many of these parsers and engines provides only limited access to the full set of possible moves of chess pieces in a given position, thus rendering the construction of chess graphs difficult and dependent.

Another and perhaps more important reason limiting the wider exposure of chess games within the graph-theoretical literature is

E-mail address: rudolph@unic.cnrs-gif.fr.
URL: http://mrudolphlilith.github.io.

https://doi.org/10.1016/j.softx.2018.12.004

the peculiar makeup of positional chess graphs. Each square on the chessboard corresponds to one graph node, and all potential moves which the pieces occupying squares on the chessboard can legally perform comprise the set of edges between these nodes, as exemplified in Fig. 1. However, the defining makeup of positional chess graphs deviates from that of the unweighted undirected relational, i.e. simple, graphs which most prominently feature in classical and applied graph theory (see [8] for a comprehensive recent presentation on the subject). Indeed, most software packages available for the analysis of real-world networks are only equipped for handling simple graphs, with an inclusion of chess graphs typically requiring the manual alteration of core data and procedural structures.

With data structures and functions tailored to handle positional chess graphs, *ChessY* provides a basic set of tools to parse, validate and visualize available PGN chess game records, and to construct, manipulate and analyze chess graphs. *ChessY*, thus, opens the door for a systematic and thorough study of chess games using the computer algebra environment *Mathematica* [9] as a powerful explorative engine. Unfortunately however, despite the historical role of chess as a most prominent board game, to the best of the Author's knowledge, no package or toolbox is yet available in *Mathematica* for both the parsing of available PGN records and the construction of chess graphs from such records, thus rendering *ChessY* the first thorough attempt in opening the door for a more systematic analysis of the game of chess using the graph-theoretical and algebraic tools made available on this powerful platform.

ChessY was already successfully utilized in a study of chess games between Grandmasters and computer players, aimed at identifying and characterizing strategical approaches employed in the gameplay of human players and computer chess algorithms [10]. This study found that for both Grandmasters and computer players the retention of more and higher-value pieces in conjunction with maintaining a high potential connectivity to squares on the board is an integral part of a winning strategy. However, a computer player leverages such a rigid "strength in number" strategy, at average, to a far greater extent compared to a Grandmaster. With such findings, *ChessY* might help to stimulate further studies of the intriguing game of chess within a graph-theoretical context, and contribute to delineating strategical and decision-making nuances observed in human players for potential incorporation into chess game engines.

2. Software description

ChessY is a small collection of high-level functions and data structures written in the *Mathematica* language for generating, visualizing and analyzing positional chess graphs. Chess positions and their associated graphs can be constructed either manually or through parsing available PGN chess game records. All generated data objects are provided as easily accessible multidimensional lists, thus are readily available for further manipulation within the *Mathematica* environment, outside the functional core provided by *ChessY*.

2.1. Software architecture

ChessY is built around three principal types of data objects, the chess position $\mathcal{P}$, nodes $\mathcal{N}$, and edges $\mathcal{E}$. While the position object contains a list of pieces, their location on the chessboard, and supplementary information characterizing a given position, nodes and edges are 1-dimensional and 2-dimensional lists, respectively, uniquely describing the positional chess graph associated with a given position:

$$\mathcal{P} = \{info, \{\{i_1, p_1\}, \{i_2, p_2\}, \ldots\}\},$$
$$\mathcal{N} = \{s_1, s_2, \ldots\},$$
$$\mathcal{E} = \{\{i_1^{\text{source}}, i_1^{\text{target}}, s_{i_1}\}, \{i_2^{\text{source}}, i_2^{\text{target}}, s_{i_2}\}, \ldots\},$$

were i denotes a unique square, or node, identifier ranging from 1 to 64, starting in the lower left-hand corner with the *a1* square on the chessboard and ending in the upper right-hand corner (*h8* square), p is a piece identifier containing color (w and b for white and black pieces, respectively) and type (K, Q, R, B, N, and P for king, queen, rook, bishop, knight and pawn, respectively), and s is the node state, e.g. $s = -1$ or $s = 1$ for squares occupied by white or black pieces, respectively, or $s = 0$ for unoccupied squares. Finally, *info* denotes a simple association list with information about the target node of a possible *en passant* move, the possibility of castling moves, as well as an eventually issued check or checkmate. This information is used by *ChessY* to process records of complete chess games, and to construct valid chess graphs associated with each position during a game.

The data objects presented above are returned by functions for the generation of chess positions and chess graphs, as well as *ChessY*'s parser for PGN chess game records, and are used as arguments in functions for the visualization of chess positions, graphs and complete games, as well as the analysis of positional chess graphs. The following list provides an overview of functions made available in *ChessY*.

Generation of chess positions:

`getPositionFromPieceFileRank`[$\{\tilde{p}_1, \tilde{p}_2, \ldots\}$]
`getPositionFromPieceNode`[$\{\tilde{p}'_1, \tilde{p}'_2, \ldots\}$]
`getPositionsFromGamePGN`[$\{m_1, m_2, \ldots\}$]

Functions for generating a chess position $\mathcal{P}$ by either manually placing individual pieces on the chessboard, or by parsing a valid PGN chess game record. Here, $\tilde{p} \in \{w, b\}\{K, Q, R, B, N, P\}\{a, \ldots, h\}\{1, \ldots, 8\}$ is a 4-character string identifying a piece placement through color, type, file and rank, $\tilde{p}' \in \{w, b\}\{K, Q, R, B, N, P\}\{1, \ldots, 64\}$ a string identifying a piece placement through color, type and node ID, and m a PGN-formatted string of a single chess move.

Generation of chess graphs:

`getNodesFromPosition`[$\mathcal{P}$]
`getEdgesFromPosition`[$\mathcal{P}$]

Given a chess position $\mathcal{P}$, these functions deliver the complete set of nodes and edges which define a positional chess graph by returning $\mathcal{N}$, a 1-dimensional list of length 64 containing all node states, and $\mathcal{E}$, a 2-dimensional list of all weighted edges, respectively.

Visualization of chess graphs:

`showChessPosition`[$\mathcal{P}$]
`showPositionalChessGraph`[$\mathcal{N}, \mathcal{E}$]

Functions for generating graphics objects displaying a chess position $\mathcal{P}$ in a classical, pieces-on-chessboard style, and a chess graph with nodes $\mathcal{N}$ and edges $\mathcal{E}$, respectively.

`animateChessPositions`[$\{m_1, m_2, \ldots\}, \{\mathcal{P}_1, \mathcal{P}_1, \ldots\}$]
`animatePositionalChessGraphs`[$\{\mathcal{N}_1, \mathcal{N}_2, \ldots\}$, $\{\mathcal{E}_1, \mathcal{E}_2, \ldots\}$]

Functions returning interactive animated graphics objects displaying multiple chess moves m_i and positions $\mathcal{P}_i$, or multiple positional chess graphs defined by their nodes $\mathcal{N}_i$ and edges $\mathcal{E}_i$, respectively. These functions are used to animate complete chess games.

Analysis of chess graphs:

`getNumberOfNodes`[$\mathcal{N}$]
`getNumberOfEdges`[$\mathcal{E}$]

Functions returning a list with the number of nodes of specific types, color or being occupied in a given node state $\mathcal{N}$, and the number of edges E given a list of edges $\mathcal{E}$, respectively.

`getAdjacencyMatrix`[$\mathcal{E}$]

Function returning the adjacency matrices a_{ij} of a chess graph.

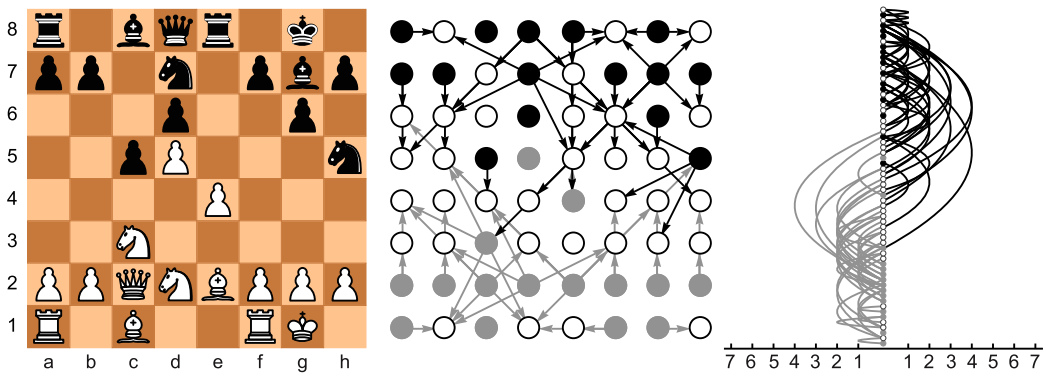

Fig. 1. Example of a chess position and its associated positional chess graph, generated by *ChessY* (Example 1). The left panel visualizes a chess position in classical format, the center and right panel the positional chess graph with 8×8 and linear node layout, respectively. The height of the arched edges in the linear layout indicates the chessboard (Chebyshev) distance between source and target nodes. The position depicted is move 11 (black) of Round 3 between Spassky (white) and Fischer (black) during the 8th World Championship in 1972.

`getConnectedness`$[E]$
Function returning the connectedness of a chess graph, defined as the ratio between the number of edges E and the number of edges in a fully connected graph with the same number of nodes.

`getControl`$[\mathcal{N}, \mathcal{E}]$
`getMobility`$[\mathcal{N}, \mathcal{E}]$
`getDominance`$[\mathcal{N}, \mathcal{E}]$
Functions which, given a positional chess graph defined by $\mathcal{N}$ and $\mathcal{E}$, return respectively the control, defined as the fraction of unique unoccupied nodes and nodes occupied by the opponent targeted by nodes in a given state, mobility, defined as the fraction of unique unoccupied nodes targeted by nodes in a given state, and dominance, defined as the fraction of unique unoccupied nodes, source nodes in a given state and target nodes occupied by the opponent.

`getAverageNodeReach`$[\mathcal{N}, \mathcal{E}]$
Function returning the number of unique nodes targeted by edges emanating from nodes in a given state, normalized by the total number of nodes in that state.

`getOffensiveness`$[\mathcal{N}, \mathcal{E}]$
`getDefensiveness`$[\mathcal{N}, \mathcal{P}]$
Functions which, given a positional chess graph defined by $\mathcal{N}$ and $\mathcal{E}$, return the number of nodes in a given state which target occupied nodes in the opposite state (offensiveness) and the number of nodes in a given state which yield potential edges to nodes in the same state (defensiveness), respectively.

2.2. Software functionalities

ChessY's primary focus is on providing tools for the generation of positional chess graphs and their associated data objects for visualization and subsequent analysis from PGN chess game records. The construction of positional chess graphs from these records starts with the conversion of the file-rank square identification *fr* commonly utilized in chess records into a numerical node identifier $i \in [1, 64]$ using the function

$$i(f, r) = 8 * (r - 1) + (f - 1) + 1,$$

were the board's files are successively mapped to numbers, i.e. $a \rightarrow 1, \ldots, h \rightarrow 8$. Each node i can have one of three states s_i, depending on whether the corresponding square is occupied by a white or black piece ($s_i = -1$ and $s_i = 1$, respectively), or empty ($s_i = 0$). Each occupied node i serves then as source for the graph's edges to target nodes j, depending on the legal moves a piece occupying the source square can potentially perform, thus populating an adjacency matrix with elements $a_{ij} = s_i$. For potential capture moves, with the exception of *en passant*, the state of the target node is $s_j = -s_i$. For castling moves, two source and target nodes exist with $s_{i_1} = s_{i_2}$ and $s_{j_1} = s_{j_2} = 0$, while for all other legal moves, including *en passant* and non-capture promotions, $s_j = 0$. Special moves (castling and *en passant*) require a reduced record of earlier game positions, carried by the association list *info* in $\mathcal{P}$. Finally, if a given position yields an edge to a node in state s_i and occupied by the king (check), then the set of potential edges emanating from nodes in the same state s_i is solely restricted to elements which remove this node as potential target in the next position. If this set is empty, the given graph marks the end of a chess game (checkmate). With this, a complete description of the graph representing a given chess position is achieved.

The generated positional chess graphs are fully described by their nodes $\mathcal{N}$ and associated edges $\mathcal{E}$, and, together with the positions $\mathcal{P}$ obtained manually or by parsing valid PGN game records, are available for visualization and analysis using tools provided in *ChessY*, or the explorative potential of the embedding platform *Mathematica*. Specifically, both the list of edges $\mathcal{E}$ as well as the adjacency matrix a_{ij} returned by *ChessY*'s respective `getEdgesFromPosition` and `getAdjacencyMatrix` functions (see Section 2.1) provide data objects which serve as arguments for the construction of graph objects utilizing *Mathematica*'s native graph construction and representation library.

2.3. Performance evaluation

In order to assess the performance of *ChessY*, a PGN database containing 2000 records of chess games was analyzed on a 2013 model Mac Pro (3.5 GHz 6-Core Intel Xeon E5, 64 GB of 1866 MHz DDR3 ECC) running on OSX 10.11 (El Capitan). The average CPU time for parsing a single PGN chess game record and generating the associated game positions (function `getPositionFromGamePGN`) was 1.95 s with a standard deviation (SD) of 0.77 s, the CPU time for constructing positional chess graphs for each position during an average game (functions `getNodesFromPosition` and `getEdgesFromPosition`) was 6.03 s (SD of 2.34 s). The comparably large error is these evaluations is a direct consequence of the inherent variability of chess games, with an average number of positions of 74 (SD of 40) in the investigated sample dataset. Finally, with nodes and edges generated, the analysis of the generated positional chess graphs during a game with the complete set of analysis functions listed in Section 2.1 took an average CPU time of only 0.41 s (SD of 0.19 s), a direct consequence of the comparably small size of positional chess graphs (64 nodes and a sparse connectivity of, at average, 1.5%).

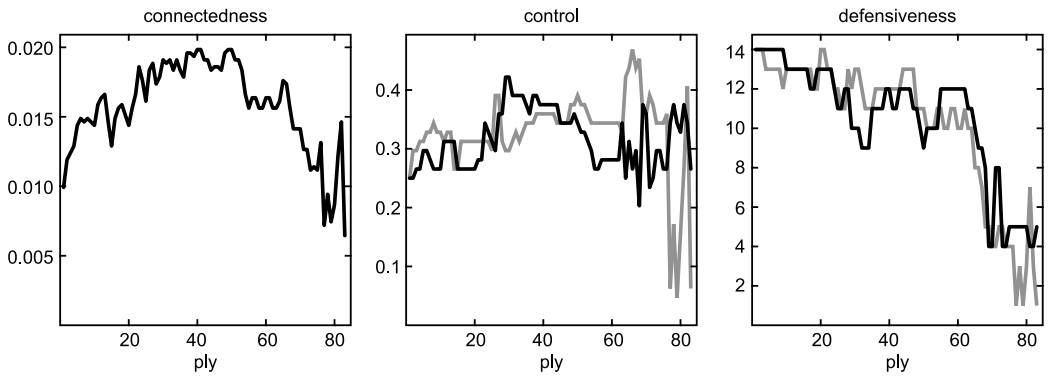

Fig. 2. Visualization of the results obtained in Example 2 for the complete game between Spassky (white) and Fischer (black) during the 8th World Championship in 1972. Shown are the total connectedness, defined as the ratio between the number of actual and possible edges, control, defined as the fraction of unique unoccupied nodes and nodes occupied by the opponent targeted by nodes in a given color state, and defensiveness, defined as the number of nodes in a given color state which yield potential edges to nodes in the same state, of white and black pieces as a function of the position during the game.

3. Illustrative examples

Example 1. To illustrate the use of *ChessY*, the first example presents the generation of the chess graph for the infamous position 23 (move 11, black) of Round 3 between Spassky (white) and Fischer (black) during the 8th World Championship (July 16, 1972, Reykjavik), with the output shown in Fig. 1.

```
position = position23SpasskyFischer1972; (* defined in
    ChessY.m *)
nodes = getNodesFromPosition[position];
edges = getEdgesFromPosition[position];
g1 = showChessPosition[position];
g2 = showPositionalChessGraph[nodes,edges];
g3 = showPositionalChessGraph[nodes,edges,
   NodeLayout->"LinearV",EdgeLayout->"Chessboard",
   ShowFrame->True];
GraphicsGrid[{{g1,g2,g3}}]
```

Example 2. In the second example, the complete game between Spassky and Fischer is processed. After extraction of the ordered list of moves from the PGN game record, a list of valid chess positions is generated by parsing all moves, and lists of nodes and edges associated with each position are generated. Finally, the position, node and edge data objects are used to analyze the game graph-theoretically, specifically the connectedness, control and defensiveness of white and black pieces are assessed as a function of the ply during the game. The results of this analysis are shown in Fig. 2.

```
(* ... load PGN file and extract the list of moves ...
    *)
(* get list of game positions, nodes and edges *)
positions = getPositionsFromGamePGN[moves];
For[i=1,i<=Length[positions],i++,
  nodes =
    Append[nodes,getNodesFromPosition[positions[[i]]]];
  edges =
    Append[edges,getEdgesFromPosition[positions[[i]]]
];
(* analyze all positions of game *)
For[i=1,i<=Length[positions],i++,
  ne = Append[ne,getNumberOfEdges[edges[[i]],
       State->"Simple"]];
  co = Append[co,getConnectedness[ne[[i]]]];
  c = Append[c,getControl[nodes[[i]],edges[[i]]]];
  d = Append[d,getDefensiveness[nodes[[i]],
       positions[[i]]]];
];
(* ... display results ... *)
```

4. Impact

Although the game of chess with its finite and discrete spatio-temporal makeup can be viewed as a prototypical example of a network in graph theory, systematic studies of the vast number of available chess game records within the powerful confines of graph theory remains, to date, scarce, mainly due to the inherent difficulties associated with the construction of chess graphs from PGN game records, and the peculiar nature of positional chess graphs. The *ChessY* toolbox attempts to bridge this gap, by providing a basic set of data objects and functions for both the efficient parsing of PGN records and the construction of positional chess graphs associated with these records using *Mathematica*.

The computer algebra environment *Mathematica* was chosen as its explorative, sandbox-like nature allows for an easy extension of the tools provided in *ChessY*, as well as an almost limitless potential for further examination of chess graphs by employing its numerical and algebraic capabilities. To our best of knowledge, *ChessY* is the first serious attempt to provide a coherent set of functions and data structures for the construction, visualization and analysis of chess graphs in *Mathematica*, and was already successfully utilized in the analysis of chess games between Grandmasters and computer players aimed at identifying and characterizing strategical approaches used by humans and computer players [10].

The goal of *ChessY* is to provide an easy-to-use and expandable toolbox for researchers studying chess games in a graph-theoretical context, and, thus, to facilitate the understanding of the dynamical complexity inherently associated with this intriguing boardgame by making chess game records accessible to the mathematical field of network analysis. With the choice of *Mathematica* as embedding platform, *ChessY* will hopefully stimulate the consideration of new approaches for studying chess games beyond a mere numerical analysis, towards more formal algebraic and operator graph-theoretical avenues. Finally, by opening the door to systematic graph-theoretical investigations, *ChessY* might bring us closer to the ambitious goal of understanding the cognitive processes at the heart of strategical thinking and decision making, and the conception of more human-like computer chess

algorithms beyond linear evaluation functions or sophisticated adaptive searches in complex decision trees forming the conceptual basis of most contemporary chess programs.

5. Conclusions

The software package *ChessY* provides an easily accessible and coherent set of functions and data structures for parsing PGN records of chess games, as well as the construction, visualization and analysis of positional chess graphs for the *Mathematica* environment [9]. Together with the unlimited explorative potential of the latter, *ChessY* is a venue for the systematic numerical and formal-algebraic study of chess games within the powerful confines of graph theory.

Acknowledgments

The author wishes to thank Audrey "Hepburn" Le Reun for valuable comments and stimulating discussions surrounding the subject presented here. Moreover, the author wishes to thank JAG Willow, CO Cain, S Hower and LS Dee for their valuable contributions to this study. Research supported in part by CNRS.

Conflict of interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

References

[1] Watkins JJ. Across the Board: The Mathematics of Chessboard Problems. Princeton University Press; 2004.
[2] Benjamin A, Chartrand G, Zhang P. The Fascinating World of Graph Theory. Princeton University Press; 2015.
[3] Johnson K. On the domination chain of m by n chess graphs. (Master thesis), Murray State University; 2018.
[4] Lu W-L, Wang Y-S, Lin W-C. Chess evolution visualization. IEEE Trans Vis Comput Graphics 2014;20:702–13.
[5] World Chess Federation. Fédération Internationale des Échecs, FIDE, Handbook. http://www.fide.com/fide/handbook.html?id=207&view=article.
[6] Levy DNL. Computer Chess Compendium. Springer; 2009.
[7] Shannon C. Programming a computer for playing chess. Phil Mag 1950;41:256–75.
[8] Newman MEJ. Networks: An Introduction. Oxford Univ. Press; 2010.
[9] Wolfram Mathematica. Wolfram research Inc.; 2016.
[10] Rudolph-Lilith M. How about a nice game of chess? IEEE Transactions on Games 2018. submitted for publication.