# Improving Game Modeling for the Quoridor Game State Using Graph Databases

Daniel Sanchez(✉) and Hector Florez

Universidad Distrital Francisco Jose de Caldas, Bogotá, Colombia
danielssj88@gmail.com, haflorezf@udistrital.edu.co

**Abstract.** Artificial intelligence has gained great importance in the last decades because based on its techniques, it is possible to make autonomous systems. In addition, it is possible to make those systems able to learn based on the previous interactions with users. This paper presents one proposal for an agent to play the Quoridor game based on some improvements in the graph model of the board. It is done by using artificial intelligence techniques to provide the capacity to learn through games played against users. Thus, learning is achieved through the use of game trees, where some of the nodes are going to be stored using a graph database. Since graph databases are one of the subgroup of the noSQL databases, which focuses in the relation representation between nodes, such databases are suitable for this kind of approaches.

**Keywords:** Game tree · Quoridor · Graph databases · Minimax

## 1 Introduction

The Quoridor game was designed by Mirko Marchesi and published by Gigamic Games in 1997. It received the *Mensa Mind Game* award in 1997 and the *Game Of The Year* in USA, France, Canada and Belgium. It has gained attention in the research community since several studies are being made to mastering this game. The reason is that despite its rules are simple, it has complex strategies to study. Furthermore, the game has a big branch factor and game tree order, which is similar to the chess game.

In the last thirty years, the studies around the game theory has increased in order to make proposals that improve the way as an artificial agent can deal with games with a certain complexity. Of course, the use of a strong hardware is a really important aspect. For instance, Deep Blue was able to review explore 200.000.000 of positions per second, which means that it was able to make a deeper search in the game tree in less time [1]. In this paper, we focus on the model of the game state since there is no a lot of research in defining its heuristics.

Graph databases are in the group of noSQL databases, which are focused on the relations between the data stored in the database, among other facilities like the semantic of the model, which can be shared directly to an expert (in this case a game player).

Á. Rocha and T. Guarda (eds.), *Proceedings of the International Conference on Information Technology & Systems (ICITS 2018)*, Advances in Intelligent Systems and Computing 721,
https://doi.org/10.1007/978-3-319-73450-7_32

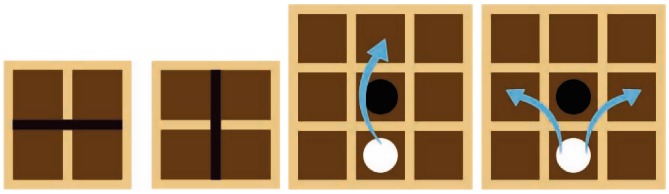
The rest of the paper is structured as follows: Sect. 2 - explanation of the game, Sect. 3 - an explanation of the features of graph databases, Cypher and Neo4J, Sect. 4 - representation of the board as a graph, Sect. 5 - future work and, finally Sect. 6 - conclusions.

## 2 Quoridor Game

### 2.1 Game Rules

The Quoridor consists of a board of 9x9 spaces and can be played by 2 or 4 players; however, in this work we just focus on 2 players, the COM and the human player. Each player has one pawn and ten fences that can be placed horizontally or vertically as shown in Fig. 1a. A player wins the game when he puts its pawn in the row where the opponent pawn starts.

On the player's turn, it is possible to either move his pawn or place one fence. Fences can be placed anywhere on the board between the squares that the pawns can move; however, a new fence cannot intersect an existing fence. In addition, a fence cannot block completely a player's access to his goal. Furthermore, the pawn cannot move through a fence. If pawns are adjacent to each other with no fence between them, a pawn may jump over the opponent's pawn. This jump must be in a straight line as shown in Fig. 1b, unless a fence is behind the other pawn, in which case the jump must be diagonal as presented in Fig. 1c.



(a) Fence positions (b) Jump over a pawn (c) Jump over a pawn with fence behind

**Fig. 1.** Quoridor Rules

### 2.2 Game Notation

For representing the game we are using the notation proposed in [2], which uses an approximation for the chess algebraic notation, explained as follows [2]:

- Each square on the board is identified by a letter-number combination.
- The columns are labeled with letters *a* to *i* from left to right. The rows are labeled with numbers *1* to *9* from top to bottom (see Fig. 2).
- The black pawn begins at square *e9*, while the white pawn at square *e1*.

– A pawn move is identified by the square that the pawn is moved to.
– A fence move is identified by the fence's *northwest square*. This square identification is followed by $h$ or $v$ to identify whether the fence is placed horizontally or vertically.
– A sequence of moves should be identified by the turn number, where a turn is ended after both players have moved, first black then white.

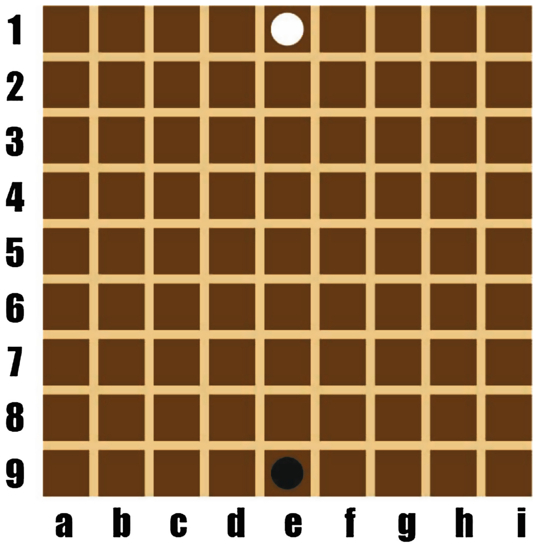


**Fig. 2.** Board positions notation

## 2.3 Game Complexity

According to Glendenning et al., [2], based on game theory, Quoridor complexity is similar to chess and checkers and can be described as a deterministic, sequential, two-player, zero-sumgame of perfect information with a restricted outcome. In addition, according to Mertens [1] there is two ways to measure the complexity of the game:

1. The state-space complexity: It refers to the different positions that may be raised in the game. For the Quoridor, it corresponds to the possible amount of positions for the pawns multiplied by the available positions to place the fences. Since the amount of illegal positions are hard to calculate; then, the upper bound is calculated. The possible pawns movements is presented in Eq. 1:

$$S(p) = 81 * 80 = 6480 \tag{1}$$

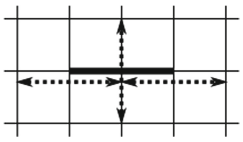


**Fig. 3.** Positions occupied by a fence [1]

Where *81* is the amount of possible pawns movements and 80 is the possible movements of the other pawn minus the position occupied for the first pawn. With this in mind, when a player places a fence on the board, both players are enabled to place another fence in four spaces. Thus, it uses the required space to put other 4 fences as shown in Fig. 3

Then, it is possible to put 8 fences in 8 columns horizontally, having 64 possible positions, and 8 fences in 8 rows vertically. So the possible movements of the fences are presented in Eq. 2. Therefore, the total value of the state-space is presented in Eq. 3.

$$S(f) = \sum_{i=0}^{20} \prod_{j=0}^{i} (128 - 4i) = 6.1582 * 10^{38} \tag{2}$$

**Table 1.** Complexity comparison

| GAME | log (state-space) | *log* (game-tree) | Average (branching factor) | Source |
|---|---|---|---|---|
| Tic tac toe | 3 | 5 | 4 | |
| Nine Men's Morris | 10 | 50 | 10 | [3] |
| Awari | 12 | 32 | 3.5 | [3] |
| Pentominoes | 12 | 18 | 75 | [4,5] |
| Connect Four | 13 | 21 | 4 | [3,6] |
| Checkers | 20 or 18 | 31 | 2.8 | [3,7] |
| Lines of Action | 23 | 64 | 29 | [8] |
| Othello | 28 | 58 | 10 | [3] |
| Xiangqi | 40 | 150 | 38 | [3,9,10] |
| **Quoridor** | **42** | **162** | **60** | [2] |
| Arimaa | 43 | 402 | 17281 | [11–13] |
| Chess | 47 | 123 | 35 | [14] |
| Shogi | 71 | 226 | 92 | [9,15] |
| Go (19 × 19) | 170 | 360 | 250 | [3,16,17] |
| Connect6 | 172 | 140 | 46000 | [18] |

$$S = S(p)S(f) = (6480)(6.1582 * 10^{38}) = 3.9905 * 10^{42} \tag{3}$$

2. The game-tree complexity: The total amount of games that can be played. This is estimated by raising the average branching factor (the number of possible game state nodes that can be derived from a game state node). According to [2] this is *60.4* and the average game length is *91.1*, so the game-tree complexity is estimated with the Eq. 4

$$G = 60.4^{91.1} = 1.788410^{162} \tag{4}$$

A little comparison between the complexities of some games similar to Quoridor is presented Table 1.

## 3 Quoridor Game Using Graph Databases

Normally, this kind of games are treated as an adversarial search problem. In this proposal, we use an approach to model the game trees to generate one leaf that represents each of the states of the game. We have the *MIN* and *MAX* players, who take turns to play, until the game finishes. Then, at the end of the game, it makes one evaluation of the movements, in order to know which is the best movement. In this moment, we are not focusing our attention in the algorithm or the heuristic function to evaluate the best movement, but instead we are focusing on representing the states of the game. It is important to mention that this approach requires a lot of processing for generating movements and visiting them. Then, in order to reduce the impact of high processing, we improve the approach by using a graph database in order to store efficiently all states of the game. Thus, we present graph databases and their useful characteristics for this kind of problems.

In this work, we used *Neo4j*[1], which is a graph database engine. In addition, we use *Cypher*[2], which is a declarative language, inspired on SQL to manage the required operations in Graph Databases. Some features of *Cypher* are: (1) matches nodes and relations in the graph database in order to extract information or modify data, (2) allows to create, update and delete nodes, relations and properties, and (3) manages indexes and constraints.

Furthermore, we used *Python* for connecting to *Neo4j* sending *Cypher* queries through the *Bolt protocol*[3], which is a connection-oriented protocol that uses a compact binary encoding over TCP or web sockets for higher throughput and lower latency[4].

### 3.1 Relational Vs Graph Databases

According to Hunger et al., [19], a Graph database is one technology which centers its attention in the representation of the data using nodes and relationships between those nodes, forming a graph structure.

[1] https://neo4j.com/.
[2] https://neo4j.com/developer/cypher-query-language/.
[3] https://github.com/neo4j/neo4j-python-driver.
[4] https://neo4j.com/blog/neo4j-3-0-milestone-1-release/.

Currently, data is increasing in volume, velocity and variety. As a result, data relationships are gaining more importance than the data itself. Traditional relational databases are not designed to capture this rich relationships. In this case, when storing tree nodes in a relational database; then, series of *JOINS* are require, which impacts considerably the system performance. On the contrary, graph databases have better response for this kind of queries.

There are a lot of different database options, including a set of *NoSQL* data stores, but none of them are explicitly designed to handle and store relationships like Graph databases.

Nowadays, organizations use graph database technology in a diversity of ways, including these six most common use cases: (1) fraud detection, (2) real-time recommendation engines, (3) master data management (MDM), (4) network and IT operations, (5) identity and access management (IAM), and (6) graph-based search.

Graph databases give the ease of a well-done, normalized entity-relationship diagram, which can quickly share with other domain experts with a representation almost similar than the real domain. For these reasons, we selected this type of store system to save the Quoridor game trees. This database has a knowledge base of all the movement possibilities of the previous games played by the application. Consequently, the application will make the queries in order to select the best next movement, so the effort is centered to explode the potential semantic given by the Graph databases in the relationships in order to label them with the correct heuristic value. Thus, the load processing is target to query the node that represents the best next movement based on the relation with the best heuristic that the application needs. This allows showing a better semantic representation of data and also a better semantic query over data.

## 4 Representation of the Quoridor Board

Several works such as [2,20,21] have evaluated the mechanisms to represent the state of the board, concluding that the best one is to model it with an undirected graph. Nevertheless, we consider that some of the connections between the spaces could be directed, this might offer a better representation because according to our evaluation, there are some situations in which some directed connections are useful. For instance, consider the situation in which a pawn needs to jump the opponent's pawn with a fence behind him.

Thus, we have create a bidirected graph using *Neo4J*. Figure 4 presents the initial state of the game from the perspective of the black player. In this state, the graph indicates the possible movements towards their neighbors direction with the possibility to return to the current position. In this initial graph the only unidirectional relation is towards the edges in the objective position.

In our approach, each node has two relations because of the nature of the Quoridor. Thus, we store in memory the digraph using *Python*, saving the two directions of the path in a dictionary structure in the case of a bidirectional relation.

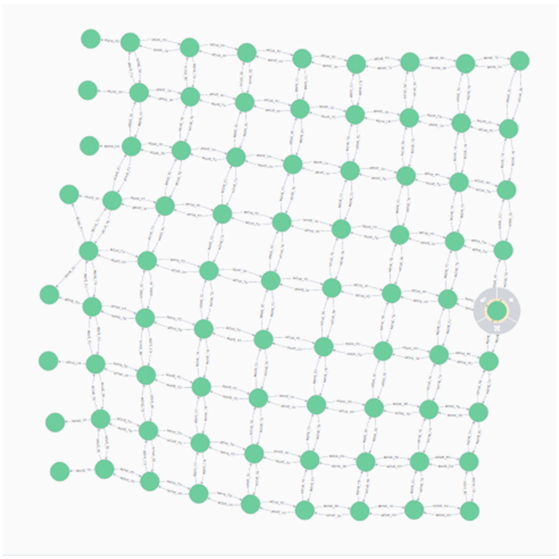


**Fig. 4.** Initial graph representation of the board.

Another example is the board state presented in Fig. 5, which is representing the perspective of the black pawn in the graph of the Fig. 6. Since it is not possible to place the black pawn in the white position, then that pawn is not modeled in the graph representation.

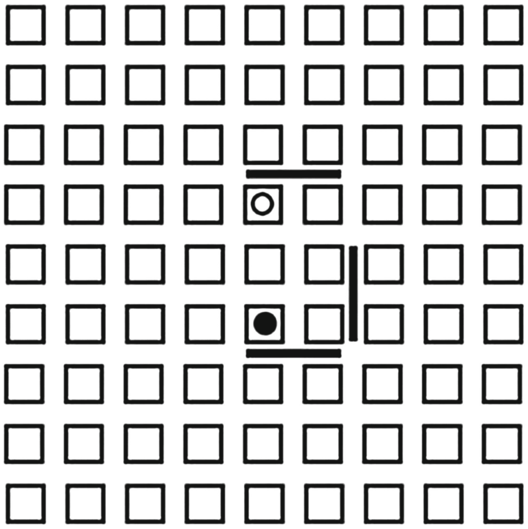


**Fig. 5.** Board state when an enemy pawn has one fence behind him

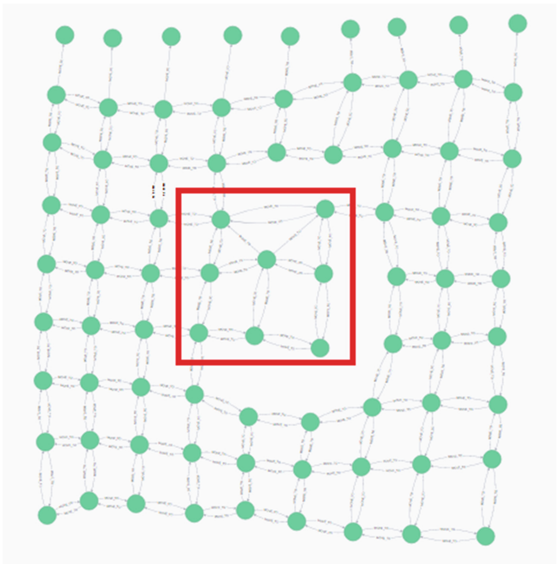


**Fig. 6.** Generated graph when an enemy pawn has one fence behind him

Figure 7 presents a zoom of the graph focusing on the center. The graph shows one connection from e5 to d4 (Jump1). However, from d4 is not possible to jump to e5 because reaching the d4 position; then, the pawn can jump the

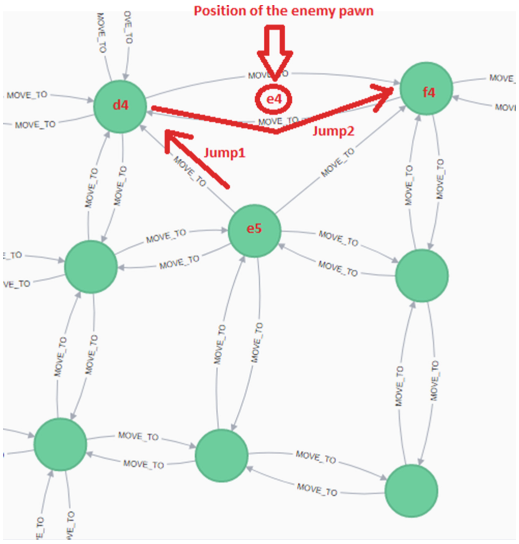


**Fig. 7.** Detailed generated graph when an enemy pawn has one fence behind him

enemy through the pawn straight to the f4 position (Jump2). In this case, it is valuable having the representation as a directed relations in those specific edges.

In addition, albeit our own pawn is not in front of the opponent's pawn, the approach calculates the graph with the available jumps over the opponent's pawn because we hypothesized that this feature might improve the minimax calculation. Moreover, based on the graph observation, it is possible to lead some unexplored heuristics.

## 5 Future Work

There is a tradeoff regarding the depth explored in the game tree; thus, accuracy demands computational resources. We believe that graph databases might provide a great picture of the study of the Quoridor game. In addition, graph databases allow seeing the behavior of the search algorithms. Thus, one opportunity based on the results of this project is testing this graphs databases in business projects. Regarding to the minimax algorithm, in the last thirty years there have been a lot of research in this area. Nevertheless, we are evaluating the idea to help minimax algorithms using Graph Databases.

Having in mind the big size of the game-tree of the Quoridor game, it is not possible to save all the possible movements in the Graph Database. Hence, we will work on saving just the eligible paths of the minimax evaluation for one node, and the minimax value calculated in some depth. Based on that, the following iteration could use those values to improve its evaluation podding by avoiding the unnecessary calculation of some nodes. In addition, it would be possible to evaluate the shortest path to the end of the game directly in the graph database [22]. Using the concepts of the network science applied in games, we plan to make a simpler representation of the moves in order to have an improvement of the space used. So far, we have focused this model in the fact that the Cypher queries can be more specific filtering the values of only one node at time and retrieve its related nodes.

## 6 Conclusions

Approaches based on trees are simple models applied in machine learning, but in this case, it has a great potential to represent semantically the states of the Quoridor game. Graph databases seems to be a good tool to apply in this kind of scenarios.

Python is a suitable language to apply in the data analysis problems because it has build-in several functionalities like the list comprehension to deal with data collections in a natural way like the mathematician used to do. In addition, the ascii-art syntax used for Cypher looks great and an easy way to create nodes and connection using Cypher into the database on Neo4j.

The bidirected graphs help us to calculate different paths from the current position of the pawn and the objective row, or even the enemy pawn in order to identify which one is in the better position in the state of the game. Moreover, graph databases help us to create dynamically data structures at the same time the data is added into the database, without having a static structure defined from the beginning like the relational databases.

## References

1. Mertens, P.J.: A quoridor-playing agent. Bachelor thesis, Department of Knowledge Engineering, Maastricht University (2006)
2. Glendenning, L., et al.: Mastering quoridor. Bachelor thesis, Department of Computer Science, The University of New Mexico (2005)
3. Allis, L.V., et al.: Searching for Solutions in Games and Artificial Intelligence. Wageningen, Ponsen & Looijen (1994)
4. Van Den Herik, H.J., Uiterwijk, J.W., Van Rijswijck, J.: Games solved: now and in the future. Artif. Intell. **134**(1–2), 277–311 (2002)
5. Orman, H.K.: Pentominoes: a first player win. In: Games of No Chance, vol. 29, pp. 339–344 (1996)
6. Tromp, J.: Johns connect four playground (1995)
7. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science **317**(5844), 1518–1522 (2007)
8. Winands, M.: Informed search in complex games. Universitaire Pers Maastricht (2004)
9. Chen, S.J.Y.J.C., Yang, T.N., Hsu, S.C.: Computer chinese chess. ICGA J. **27**(1), 3–18 (2004)
10. Park, D.: Space-state complexity of korean chess and chinese chess. arXiv preprint arXiv:1507.06401 (2015)
11. Cox, C.J.: Analysis and implementation of the game arimaa. M. Sc. diss., Universiteit Maastricht, The Netherlands (2006)
12. Wu, D.J.: Move Ranking and Evaluation in the game of Arimaa. Ph.D. thesis, Harvard University (2011)
13. Brian, H.: A look at the arimaa branching factor (2006)
14. Shannon, C.E.: Programming a computer for playing chess. In: Computer Chess Compendium, pp. 2–13. Springer (1988)
15. Iida, H., Sakuta, M., Rollason, J.: Computer shogi. Artif. Intell. **134**(1–2), 121–144 (2002)
16. Tromp, J., Farnebäck, G.: Combinatorics of go. In: International Conference on Computers and Games, pp. 84–99. Springer (2006)
17. Tromp, J.: The number of legal go positions. In: International Conference on Computers and Games, pp. 183–190. Springer (2016)
18. Xu, C.M., Ma, Z., Tao, J.J., Xu, X.H.: Enhancements of proof number search in connect6. In: Control and Decision Conference, CCDC 2009, pp. 4525–4529, Chinese. IEEE (2009)
19. Hunger, M., Boyd, R., Lyon, W.: The Definitive Guide to Graph Databases for the RDBMS Developer. Neo4j (2016)
20. Quinlan, J.R.: Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
21. Tsur, G., Segev, Y.: Quoridor agent. http://www.cs.huji.ac.il/ai/projects/2012/Quoridor/files/report.pdf
22. Castelltort, A., Laurent, A.: Fuzzy queries over NoSQL graph databases: perspectives for extending the cypher language. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, pp. 384–395. Springer (2014)