Maastricht University

Group Project 7:

Final Report
Game Distance Metric

created by

Shcherbakov, Kirill - i6248484

Abraham, Frederic Marvin - i6262598

Giannilias, Theodoros - i6255101

Penroz Valenzuela, Diego - i6261944

Franzoni Darnois, Guillaume - i6256477

Supervisors:	Cameron Browne, Walter Crist and Matthew Stephenson
Coordinators	Kurt Driessens and Gijs Schoenmakers

University of Maastricht

Data Science & Knowledge Engineering

Maastricht, January 20, 2021

Abstract

In this research project report, we propose different metrics to find game distance estimation on a phenotype basis, i.e., focusing on how a game is played. The metrics are developed as an extension to the Digital Ludeme Project in order to help with the modeling, reconstruction, and mapping objectives for its games database. One of the phenotype approaches utilizes the Tree Edit Distance Metric, which compares games by calculating the distance between generated game trees of the two games. Various techniques have been used to generate differentiating trees using different aspects of the games. The second concept utilizes Play Trace Distance Metric, which compares differently created play outs.

In order to compare the phonetic structures, further processing needs to be done. The proposed Labelling algorithms, applied in both mentioned approaches, enable comparisons by storing essential game-related information inside the generated structures. We show that the developed phenotype approaches are independently able to calculate distances between games that match the prior classification done by the Ludii General Game System Team. This project shows that the concept of utilizing phonetic structures of the games is a viable approach to compare games and that future work with experts in the field needs to be done to refine the methods.

Contents

Abbreviations and terms

Abbreviations

GDL Game Description Language

TED Tree Edit Distance

PTS Play Traces Similarity

GGP General Game Playing

TF-IDF Term frequency–inverse document frequency

MCTS Monte Carlo Tree Search

UCT Upper Confidence Trees

PCA Principal Component Analysis

Terms

Unified move label A string representation of a move played in a game that is unified over two different games so that they match.

1 Introduction

1.1 Context and Motivation

Games have been part of the human being and his interaction for thousands of years. They are part of our life, used for different purposes regardless of our culture or age. On the other hand, there are several differences between them. Many games have been evolving, "traveled between cultures", some become extinct, and new ones have been created.

Important features or characteristics are the rules. Sometimes more complicated and sometimes more straightforward, but with enough remarkable relevance to define a game almost entirely. The issue is about the number of players, the board‘s shape, objectives. Precisely, the rules imply differences, which in turn indicate that we can differentiate the games, classify them and, in some way, compare them. If we are given two games, we can determine how similar or dissimilar they are or analyze them. However, we are talking about a considerable amount of data, so several techniques have been developed in recent years for classification, clustering games into meaningful categories.

It is here when we can better understand one of the main parts of the context of our research, the Digital Ludeme Project, which investigates the development of traditional strategy games with modern methods and that is being developed in Maastricht, Netherlands. Following its purposes, our team wants to contribute to this far-reaching program with the Games Distance Metrics Project. The objective is to develop a capable tool in order to distinguish games on a behavioral basis. The reason why this topic is being explored is, on the one hand, to run a check on newly and automatically generated games to assess their originality. On the other hand, it can be used as a method to prevent plagiarism.

Moreover, understanding the distance between two games makes it possible to create and update clusters with more accurate and precise information, which can be related to both the genotype and the phenotype.

1.2 The Digital Ludeme Project

The Digital Ludeme Project (Browne, 2019) is a five-year project ranging from 2018 to 2023 hosted by Maastricht University and funded by the European Research Council (ERC) Consolidator Grant. "The key objective is to study Traditional Strategy Games, presented in the public domain that rely on mental acuity and stem from different cultures and periods starting at 3500 B.C." (Heinze et al., 2020). The general aim of this project is to improve our understanding of traditional strategy games, using modern Artificial Intelligence techniques, to chart their historical development, and explore their role in the development of human culture and the spread of mathematical ideas.[1]

Some of the project's key objectives are :

- **Model** the full range of traditional strategy games in a single playable database.
- **Reconstruct** missing knowledge about traditional games with an unprecedented degree of accuracy.
- **Map** the transmission of games and associated mathematical ideas across history and culture.

1.3 Problem Statement

This research focuses exclusively on games studied as part of the Digital Ludeme project. The Ludii General Game System [2]defines a range of games in a custom game description language based on the ludemes that make them up. These are traditional strategy games, i.e., games in the public domain without an owner, based on logical decision-making and were invented before 1875. (Browne et al., 2019).

More formally, a game can be defined as:

$$\text{Game} : G = \{Equipment, Rules, Mode\}$$

Where the 3-tuples provide information on the form (genotype: rules and equipment) and the functionality (phenotype: behavior during the play) of a specific game. Each

[1]The website of the project: http://ludii.games.

conceptual unit of game-related information that is used to specify a game is called a ludeme.

This project aims to define measures or algorithms that quantify how similar two games are (potentially, in terms of the underlying structure of rule sets and moves), based on the form or functionality of the games, thus allowing us to estimate the distance between games.

The problem statement then is the following:
Given a set of n games $G = \{g_1, \ldots, g_n\}$ in their ludemic description and three games $g_i, g_j, g_k \in G$, where $i, j, k \in \{1, \ldots, n\} \wedge i \neq j \neq k$. Find a distance measure $dist(g_i, g_j)$ such that it is likely that if $dist(g_i, g_j) > dist(g_i, g_k)$ then the behaviour and structure of g_i is closer to that of g_k than g_j. And test whether or not it is the case.

1.4 Research Questions

The research questions are the following:

- Which aspects of games allow the functional distance between two games to be reliably measured?
- Which of the existing technologies is best suited to achieve the project goal?
- How will we evaluate the reliability of the obtained results?

More approach-oriented research questions (based on labeled moves from playouts):

- Is it possible to generate unified move labels[3] given a game?
- Which selection techniques are we going to use to generate a labeled game tree out of a given game?
- Having labeled game trees, can a reliable distance metric between two games be computed?
- Is it plausible to average the computed results that have been extracted from different distance measurements?

[3] *unified move labels*: A string representation of a move played in a game that is unified over two different games so that they match.

1.5 Structure of the paper

In this subsection, we briefly explain the structure of our paper by summarizing the contents of the paper as well as introducing our major approaches for this research project.

Firstly, we give a brief and general introduction of the topic so it can be understandable by those who read it as well as introducing our purpose and contribution to the field. Secondly, with provide the reader with a general overview of the previous related work that has been done in the field. Moreover, we mention the social impact of similarity measures between games and the cultural aspect that can be distinguished by that similarities.

Afterwards, we describe our methodologies and approaches by defining our labeling algorithm in more detail. Additionally, we present our distance metric approaches by defining the differences in each of them. Furthermore, we state the obtained results extracted from our approaches by using visualized graphs, and in the next section, we discuss the inference of them by comparing all our methods.

Concluding, we are providing our ideas for methods and uninvestigated approaches that can possibly be investigated and therefore implemented in future researches related to this work.

2 Previous and related work

In this section, the current existing technologies related to the main goal of the project will be discussed in detail. We will primarily describe the general tools and techniques that we will exploit to calculate a reliable and accurate distance between two or more games written in the Ludii format. Secondly, the previous attempt to generate phylogenetic similarities between games is presented to provide the reader with a good overview of the whole topic.

2.1 Existing Engines

2.1.1 General Game Playing and Game Description Language

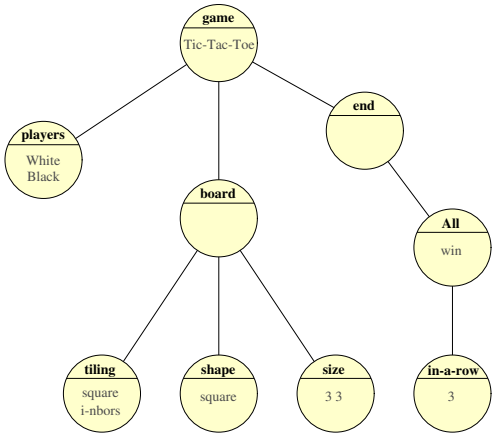
In 2005, the Stanford Logic Group came up with the Game Description Language (GDL), used to develop games in a way that it enables the use of an agent to perform efficiently in many arbitrary games, General Game Playing (GGP). The basis for GDL is a conceptualization of games in terms of entities, actions, propositions, and players. In 2017 the Ludii platform (Stephenson et al., 2019), a complete general game system, was launched. This new system improves the previous framework of GDL by following a more class grammar approach based on ludemes, and therefore simplifying the language. Every ludeme is correlated to a specific type of information and characteristics of games such as equipment, rules, end-conditions, etc. The pictures below illustrate the different descriptions of Tic-Tac-Toe between these two frameworks:

```
(role white) (role black)
(init (cell 1 1 b)) (init (cell 1 2 b)) (init (cell 1 3 b))
(init (cell 2 1 b)) (init (cell 2 2 b)) (init (cell 2 3 b))
(init (cell 3 1 b)) (init (cell 3 2 b)) (init (cell 3 3 b))
(init (control white))
(<= (legal ?w (mark ?x ?y)) (true (cell ?x ?y b))
(true (control ?w)))
(<= (legal white noop) (true (control black)))
(<= (legal black noop) (true (control white)))
(<= (next (cell ?m ?n x)) (does white (mark ?m ?n))
(true (cell ?m ?n b)))
(<= (next (cell ?m ?n o)) (does black (mark ?m ?n))
(true (cell ?m ?n b)))
(<= (next (cell ?m ?n ?w)) (true (cell ?m ?n ?w))
(distinct ?w b))
(<= (next (cell ?m ?n b)) (does ?w (mark ?j ?k))
(true (cell ?m ?n b)) (or (distinct ?m ?j)
(distinct ?n ?k)))
(<= (next (control white)) (true (control black)))
(<= (next (control black)) (true (control white)))
(<= (row ?m ?x) (true (cell ?m 1 ?x))
(true (cell ?m 2 ?x)) (true (cell ?m 3 ?x)))
(<= (column ?n ?x) (true (cell 1 ?n ?x))
(true (cell 2 ?n ?x)) (true (cell 3 ?n ?x)))
(<= (diagonal ?x) (true (cell 1 1 ?x))
(true (cell 2 2 ?x)) (true (cell 3 3 ?x)))
(<= (diagonal ?x) (true (cell 1 3 ?x))
(true (cell 2 2 ?x)) (true (cell 3 1 ?x)))
(<= (line ?x) (row ?m ?x))
(<= (line ?x) (column ?m ?x))
(<= (line ?x) (diagonal ?x))
(<= open (true (cell ?m ?n b))) (<= (goal white 100) (line x))
(<= (goal white 50) (not open) (not (line x)) (not (line o)))
(<= (goal white 0) open (not (line x)))
(<= (goal black 100) (line o))
(<= (goal black 50) (not open) (not (line x)) (not (line o)))
(<= (goal black 0) open (not (line o)))
(<= terminal (line x))
(<= terminal (line o))
(<= terminal (not open))
```

Figure 2.1: Tic-Tac-Toe description in GDL. (Browne, 2020).

```
(game "Tic-Tac-Toe"
    (players 2)
    (equipment {
        (board (square 3))
        (piece "Disc" P1)
        (piece "Cross" P2)
    })
    (rules
        (play (move
                  Add (to (sites Empty))
              )
        )
        (end
            (if
               (is Line 3)
               (result Mover Win)
            )
        )
    )
)
```

(a) Description in Ludii



(b) Tree description in Ludii

Figure 2.2: Representation of the Tic-Tac-Toe description in Ludii as a tree (Browne, 2020).

2.2 Major Approaches

2.2.1 Genotype

In terms of biology, a genotype is defined as the collection of traits of the individual that is inherited from the parents. For instance, DNA sequences can be considered as genotype. However, in terms of this research project and the Ludi Platform, genotype will refer to the ludemes that are used to describe games. Ludemes basically correspond to the encoding of the rules and the characteristics of games(Heinze et al., 2020). One example of genotype in the Ludi Platform will be to construct labeled trees based on the rules of the games and compare those written rules with the use of Levenshtein distance.

2.2.2 Phenotype

On the other hand, phenotype is a genetical term that is being used for composite observable characteristics or traits of an organism. In this research topic, the phenotype corresponds to the actual behaviour of a game when being played, dictated by the rules of it(Heinze et al., 2020). To distinguish between different games, several quality measures are being used, such as the average game length or the sequence of player's strategic decisions from a sample of playouts.

2.3 Methodologies

2.3.1 Tree Edit Distance Metrics

A lot of work has been done by previous research, especially on the computation of the distance between labeled trees, which will be on aspect that this project will focus on. Bille, 2005 researched the topic widely with specific emphasis on the tree edit distance method.

2.3.2 Play Traces Distance Metric

Another similar approach that has already been exploited is the comparison between play traces of games to distinguish dissimilarities between them. The significant difference in this method lies in the type of content highlighted. In particular, this

technique of comparing two games focuses on the semantic contents so that the rating calculated is more representative of how humans perceive the feature's differences (J. Osborn et al., 2014). Gamalyzer is an example of a tool that is capable of computing such a metric. However, it focuses only on goal-oriented games by comparing input files that encode to play traces. The tool, concerning the calculation of distance metric, is available on GitHub[1].

2.4 Previous attempts

To the best of our knowledge, the only existing attempt is the one made by the previous year's group. (Heinze et al., 2020) The approach used in that project focused mainly on the phylogenetic aspects of the trees trying to reconstruct the evolutionary history of the games included in Ludii, especially the ones belonging to the Mancala family. In that case, a simple Bag of Words approach is based on keywords included in the Ludii games description.

Initial processing of the .lud files was necessary to remove all the unnecessary information. Following that, a basic frequency distribution of the keywords over the files was done without considering the relationship or the order between the words.

An additional approach was used from the previous year's group by trying to implement the Graph Similarity algorithm. The group tried to extract the board by using an algorithm for receiving an input for Graph Similarity Algorithm. Unfortunately, even though this method was quite practical on mancala family games since, for every variation of them, the board is quite similar, the results proved that this technique is an NP-Hard problem and therefore not feasible for computing accurate distances(in dissimilar boards) for trees that contain more than 12 nodes.

A more sophisticated approach then was used, the Term frequency–inverse document frequency (TF-IDF) which also a genotype approach and basically improves the previous framework "bag of words" by reflecting how important some words that are encountered inside a game are. For this development, what the previous year's group did was to distribute weights to the words that were encountered less time during the files analysis.

It is, therefore, necessary to point out the differences between our project and the previous one. Our main target is the development of a tool capable of distinguishing

[1]GitHub repository: https://github.com/JoeOsborn/gamalyzer

games based mainly on their behaviour (Phenotype approach). This way, the result obtained not only will serve as a tool to cluster games into meaningful categories but also has the potential to help the creation of new and different games by checking the distance from the existing ones. Instead, from a technical perspective, the previous year's students tried to produce a phylogenetic classification of games (Genotype approach).

2.5 Previous Literature

Even though all these years the primary goal of building an evolutionary tree of ancient games has been explored with the use of advanced computational methods, it still remains a relatively unexplored topic.

As far as we know, not much work has been done in classifying the difference of phenotype between different board games. For instance, Genre discovery engines and methods of assigning video games to existing genres have been developed, but expect the game properties to be already assigned. Furthermore, several arbitrary board games that do not contain the factor of chance, luck, the quality measures of depth, clarity, and drama, etc. have been suggested(Heinze et al., 2020).

However, the lack of analytical tools to viable distinguish board games phenotype has been criticized over the years. For that reason, in this research project, our major contribution will be concerning mostly the phenotype approach by focusing on identifying reliable distance metrics from several methods that will encompass a set of elements regarding the behavior and the characteristics of the games when being played.

2.6 Social Impact

To discover the origin of a game, three main sources are of use: Literary sources, archaeological findings, and the game itself.

By focusing on the game aspects and trying to identify cultural similarities or differences in how the games are built and played together with an accurate and reliable distance metric can be used to create a family tree of clustered games (Heinze et al., 2020).

In terms of phenotype and characteristics of the games, one element that can be distinguished would be the way of thinking from each nation since the way the

game is played can be considered as a possible factor that reflects the mindset of different nations and people.

Another possible beneficial extraction of that would be to identify key aspects of several games and possibly combine them to create new complex games, which will inherit several different characteristics and points of view.

This will be a huge contribution to the evolution of the games and society in general since it will allow people to know and understand multiple cultural aspects just by playing these complex games.

Finally, this will also help understand the evolution of human history and the illustration of the common cultural heritage of humanity, especially in cases where there are light records.(Heinze et al., 2020)

3 Concepts, Approach and Methodology

When comparing games, there are two major ways to approach this problem. The first being the genotype approach, which focuses the distance metric on the items that describe the game on a direct level. One example of a genotype approach would be to compare the written rules with the Levenshtein distance. In the perspective of Ludii defined games, which describe the games in a tree structure (Stephenson et al., 2019) resulted in using the Tree Edit Distance (TED) in order to calculate a distance metric. Another genotype approach is the comparison of the game boards. As the game boards in Ludii are defined as a graph, one can use graph edit distance Stephenson et al., 2019 to compare the two boards.

The second major approach is the phenotype approach. Which, in contrast, focuses on the way the game is played. The way the game is played can be captured in play traces. A play trace of a game is considered to be a list of the moves made in one play out. A game tree can be regarded as the combination of multiple play traces. For example, the root of the tree can be the beginning of the game, and each possible move from the respective position is taken as the leaves.

Within the project, we are focusing our attention on the phenotype approach. We investigate different methods to generate different play traces and game trees.

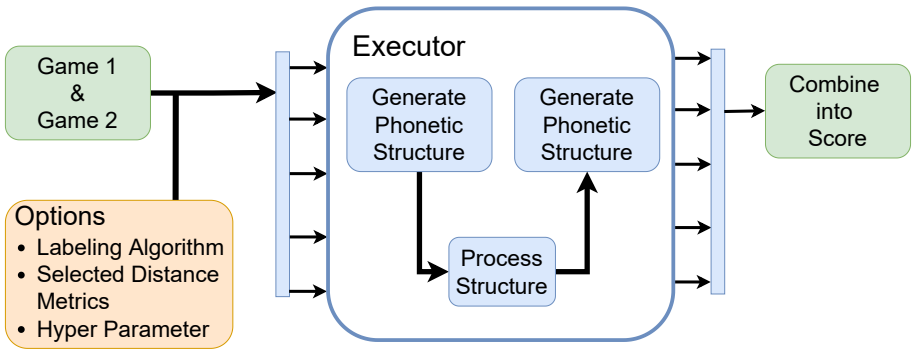


Figure 3.1: General structure for computing distance metrics between two games.

As visualized in Figure 3.1, the input to our algorithm contains the two games to compare and a distance option object containing the used distance metrics, hyper parameter, and which labeling algorithm should be used. The option object can be constructed with a builder pattern, which allows for easy customization for each distance metric. For each selected distance metric, the phonetic structures need to be generated, then prepossessed and followingly compared. As this is a lengthy process this part is parallelized for each selected distance metric with a work sharing thread pool. After each calculation is done the score of each successful metric is combined into on score.

The distance metric calculated is a value between 0 and 1, where 0 means no similarity and 1 means similar. Therefore for some of the distance metric we have to computed $1-$result where the result has to be between 0 and 1.

3.1 Labelling Algorithm

In this section the main concepts related to the labeling algorithm are going to be discussed in order to make clear specific steps found in the upcoming sections about TED Metric and Play Traces Similarity (PTS) Metric.

3.1.1 Concepts

Labeling represents a key concept for developing methods that can efficiently and accurately describe similarities or differences between games. The reason that leads to the formulation of the previous sentence is fairly simple: algorithms used in the TED or PTS compute metrics by comparing information stored inside the predefined data structure. This means that such information must capture the essence of the games.

First of all, it is then necessary to find what data seems to represent the game in a meaningful way. While doing so, a particular constraint must be taken into account: the goal is to calculate distances with attention to the phenotype. To do that we identified such data in the Moves that are played, which bring the game from one state to another, highlighting how the rules (genotype) can be used/applied to influence the game playing (phenotype).

Different approaches were tried throughout the development of this project, although all of them were based on the concepts of **Moves** and **Actions**. Both derive directly from objects found inside the Ludii project: the first conceptually contains many

instances of the second, meaning that the Actions are atoms that can be combined to make up a single Move.

3.1.2 Approaches

In this section, all the approaches will be explained and discussed to give a good overview and make the following sections more easily understandable. The description will be structured as follows: first, the label generation process will be explained, then the strengths and weaknesses of both generation and resulting labels will be discussed. Moreover, the approaches will be presented chronologically, starting with a simple idea and refining it into more sophisticated methods.

In the section dedicated to the results the actual robustness and accuracy of every method will be analyzed.

3.1.2.1 First Approach

• **Generation**:
This first approach could be considered the naive one. It relies on a straightforward method that is commonly used in Java: *toString*. In fact, the process obtains from the simulated game the last move played, and from it the constituent actions. Those actions are store inside a list, which is turned into a String by calling the previously specified method. Hashing is performed on this last object to obtain an Integer, which serves as a key to access a HashMap where all the previously obtained labels are stored. Finally, the label is returned by either getting it from the dictionary, if already present, or creating it by saving the size of HashMap as a value of type String.

Figure 3.2 visually explains this process.

Figure 3.2: Label Generation Method 1

• **Strength & Weaknesses**:
This method is not to be considered a lousy approach because of its simplicity. It can provide reasonable results when applied to the Tree Edit Distance Metric, as we will see in the tests section. There are although some problems associated with it. The data used to generate the labels is exceptionally detailed for many

games where pieces are moved. This results in specific information related to the cells from which the piece is moved to be the main discriminant that makes labels different.

In this instance, the issue is the absence of common labels even inside the same game play out and thus high distance even when comparing a game to itself. Overall, though extremely low (in terms of similarity between games), the values obtained seem to capture structural differences in the generated trees.

3.1.2.2 Second Approach

• **Generation**:
The first approach has the defect of producing a very high number of labels due to the level of detail provided by the data used to generate the labels. For this second approach, the goal is mainly to reduce that number by generalizing over the possible moves played during a game.

In this case, instead of turning a move into a String object, we proceed by recursively obtain every action that makes up the move. All the actions are then turned into a String object that uniquely identifies them. Following that, the same approach used for the first method is applied. All the Strings are hashed and then finally XORed together to obtain a single value. This final value is used to access the HashMap and returning the correct label.

Figure 3.3 describes with a scheme the process explained above.

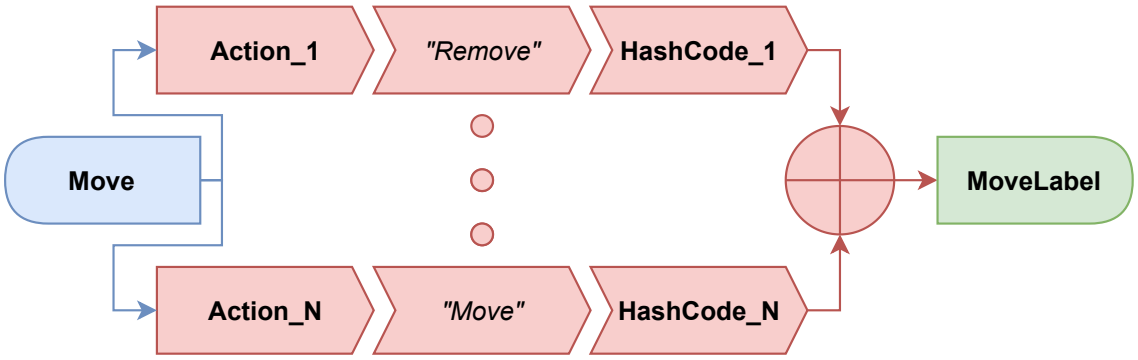


Figure 3.3: Label Generation Method 2

• **Strength & Weaknesses**:
This method improves a lot on the previous one. The number of labels generated is lower than the previous approach has: from approximately 600, it drops to 5 for the game of Chess. Moreover, as previously explained, such labels are more general, which means that only essential information is kept to distinguish

them producing more meaningful results that work well with both TED and PTS.

The main problem that mine this approach is opposite to the one presented in the first approach: generalization. In this case, specific information about the type of movement made by pieces is left out, leading to difficulties in distinguishing games with moving pieces since all the motion moves are considered the same.

3.1.2.3 Third Approach

• **Generation**:
This third approach tries to come up with a solution for the excessive generality of the previous method. To do so, it relies on what we defined as Walks, meaning the sequence of steps taken by a piece when it is moved.

For instance, the walk of a piece moving diagonally on a squared board will be described as follows: *{F, R, ..., F, R}*. In this example the letters *F* and *R* correspond respectively to *Forward* and *Right*. Since this method is developed starting from the previous one, part of the process remains the same. In fact, every single step is equal, except a new one is added. After XORing together all the hashed derived from the actions name, a similar process is applied for the walk with a little twist. If a piece has been moved, the walk is calculated and then turned into a sequence of Strings, each of them representing a step. This is then checked against all the values already stored in a dedicated HashMap and then used to obtain an existing hash value or create a new one. Once this is done, the value is XORed together with the label previously obtained to create a more specific one.

In Figure 3.4 the entire process is visually described.

• **Strength & Weaknesses**:
This attempt, as we will discuss in the results chapter, did not seem to produce a more accurate result. It presents, in fact, two main flaws.

The first one is a lack of generalization over the type of moves played. It does not currently exist in Ludii, nor is it easy to implement a method capable of generalizing from Walks. This makes our method too specific to be able to accurately capture the similarity between pieces slides.

The second flaw groups together these first three approaches. The label produced is described as a single value. Having such a simple data structure make further meaningful similarity comparisons impossible. Despite these two defects, it still improves

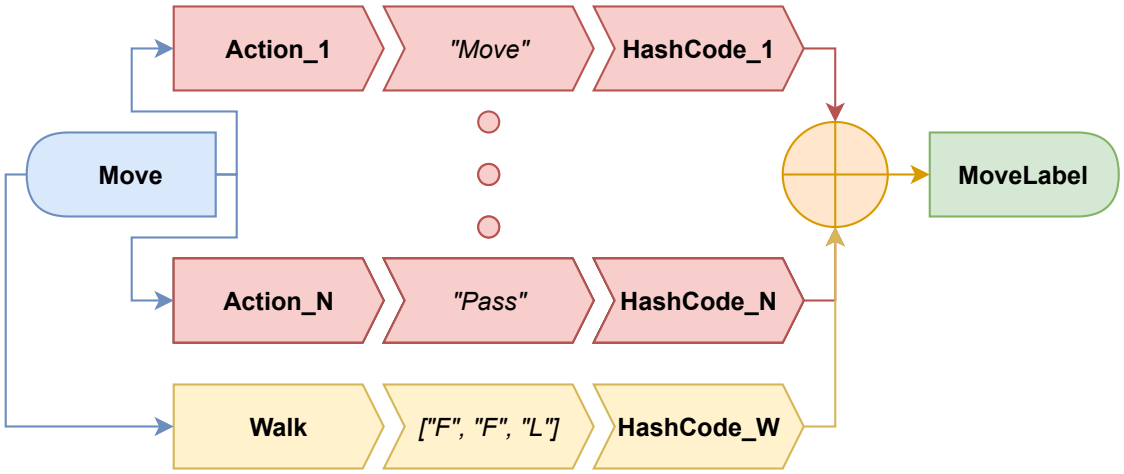


Figure 3.4: Label Generation Method 3

the results obtained with the first approach, and we believe that it would provide more accurate results with the right movement generalization.

3.1.2.4 Fourth Approach

- **Generation**:

In this last approach, the goal is to overcome the issue presented by the generation of labels in a single value format. To do that, we edited the second method explained in Section 3.1.2.2.

From the Move object, all the actions are obtained and again turned into their String representation. Instead of being hashed and XORed together, these values are simply collected in a list that is return as a label. This highly simplifies the whole process.

Figure 3.5 shows this generation process.

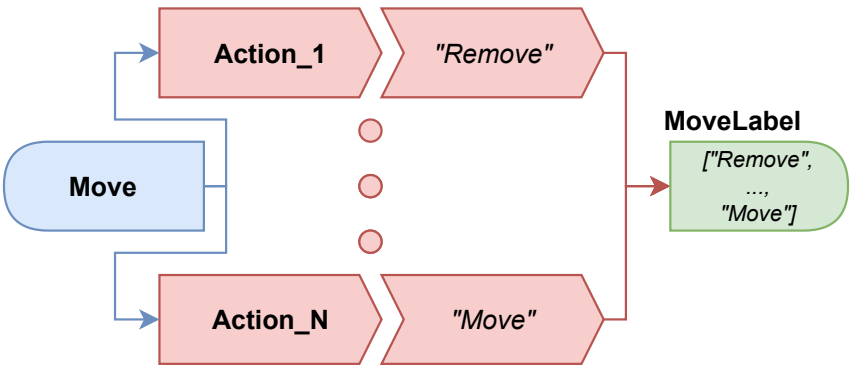


Figure 3.5: Label Generation Method 4

• **Strength & Weaknesses**:
Later on, the results obtained with this last method will be discussed more thoroughly. For now, it will only be mentioned that the value obtained leads to believe that out of the four implemented methods, this is the one providing the best performance. This algorithm's main strength is producing multi-value labels that can be used to compute fractional similarity, especially when used together with Play Traces. This calculation allows capturing similarity in cases where two moves have one or more common actions.

While being a promising approach, it suffers the same problems described for the second method. It uses very general information to generate labels leading to many duplicate labels to be produced and making it challenging to capture differences between games that present similar play out but different types of movements of the piece.

3.1.3 Label Comparison

Finally, in this section, the comparison between labels is introduced to make the understanding of later concepts easier. More importantly, it explains what is referred to as *Fractional Similarity* in the previous paragraphs. This concept is especially useful in combination with the PTS approach and, even though not yet implemented, it lends itself to the development of custom cost functions that can be used in TED. The similarity measure used is the Jaccard's Similarity, which simply consists of the ratio between the intersection over the union of two given sets A and B:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.1}$$

Since the data structure on which this algorithm applies is a List of Strings, the list has to be turned into a set, which leads to obtaining the ratio of common actions over all the actions in both labels. There are, although some exceptional cases where this is not enough. In fact, some games present moves where the same action is performed more than once. The basic similarity then has been updated to handle these cases by renaming repeated labels and thus preserving them when casting the list to a set.

3.2 Tree Edit Distance Metric

Trees are among the most common and well-studied combinatorial structures in computer science (Bille, 2005). In order to compare trees, the TED is utilized as a distance metric between labeled trees. The algorithm compares two trees by computing the minimal amount of actions required to transform the first tree into the second. In particular, those actions are the deletion of a node, the insertion of a node as a leaf or between two nodes, and the renaming of a node.

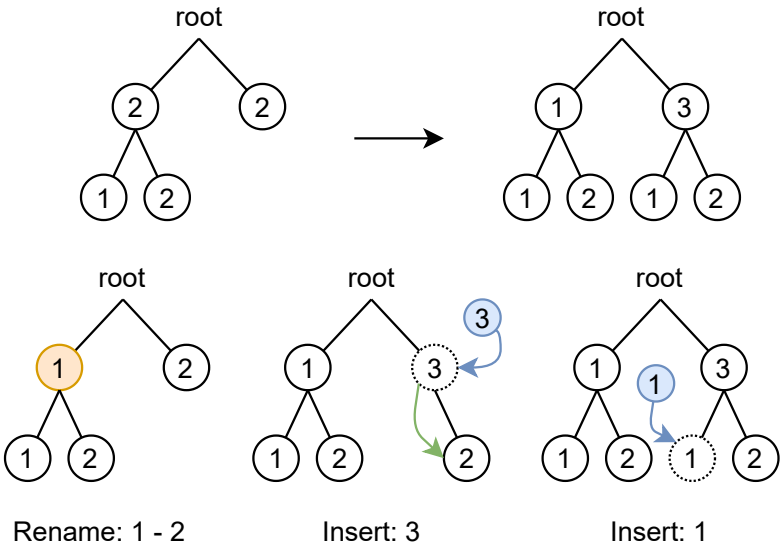


Figure 3.6: Visualization of the Tree Edit Distance

Figure 3.6 describes exemplary the actions required to calculate the tree edit distance. The distance in this instance is three as it requires one rename and two insert actions.

Therefore comparing games by comparing generated game trees utilizing the tree edit distance is one of the phenotype approaches we are investigating. A game tree in the sense of game theory is a directed graph whose edges are moves of the game and whose nodes are the resulting positions.

The tree edit distance in itself is not a good indicator for the distance between two games as the number of required actions usually rises proportionally with the size of the trees. Consequently, the distance between bigger games would be higher than between smaller games. In order to get a comparable value, a normalization of the tree edit distance is used.

$$NTED(T_1, T_2) = \frac{TED(T_1, T_2)}{max\left(|T_1|, |T_2|\right)} \tag{3.2}$$

With T_1 and T_2 being two trees, and $|T|$ being the amount of nodes in the tree, equation 3.2 is a commonly accepted approach for normalization (Li & Chenguang, 2011).

The larger the editing distance of the tree, the more dissimilar the trees are.

3.2.1 Generation of Trees

As we try to capture the way the game is played in different structures, we also look into different approaches on how to capture differentiating aspects of the game in the game trees. Four different techniques are to generate different trees were developed. They can be grouped to either take into account the location of the state of the game or if an AI was used in the generation process.

First, we look at the more straightforward generation methods in which the tree's location in the full game tree is taken into account.

Secondly, an AI is involved to prune parts of the tree that seem like a bad option. We use the game Tic-Tac-Toe to show the different ways to capture aspects of the game, as it is a small game.

3.2.1.1 Full Game Tree and End Game Tree

One aspect of a game regarding the location of the tree is how it is played in the beginning. This can be captured by the full game tree, where the root node is the beginning of the game.

Figure 3.7 displays Tic-Tac-Toe's full game tree where representatively for the second turn only on the move is continued. The first row contains every possible move. The opening player can play. As two players play tic-Tac-Toe, the second row is the opponent player's turn. Contrary to the game tree in Figure 3.7 the full game tree would have the displayed second row for every move in the first row regarding the first move played.

Like the full game tree, the end game tree tries to capture away the game is played at a specific position. Contrary it is not the beginning but, as the name implies, the end of the game. The end game tree requires, as the name suggests, the end of the game. Therefore, we generate a random play out and store the last n nodes. The required amount of nodes depends on the given tree depth provided as a hyperparameter.

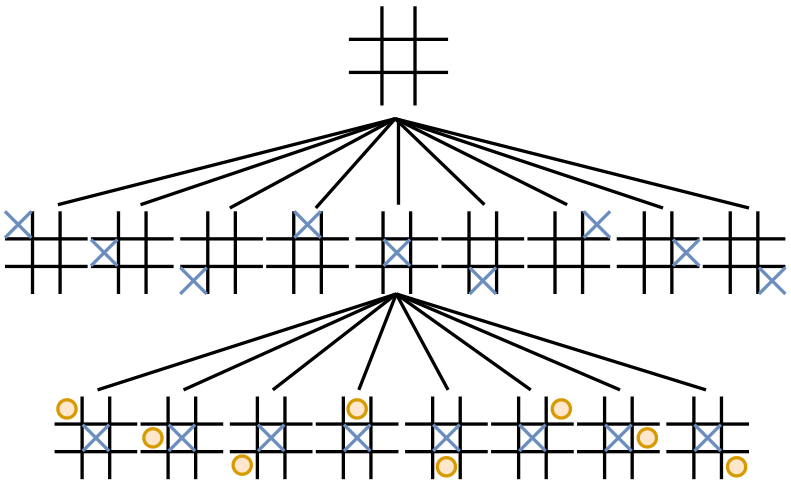


Figure 3.7: Full Game Tree example that is not actually a full game tree because that wouldn't fit

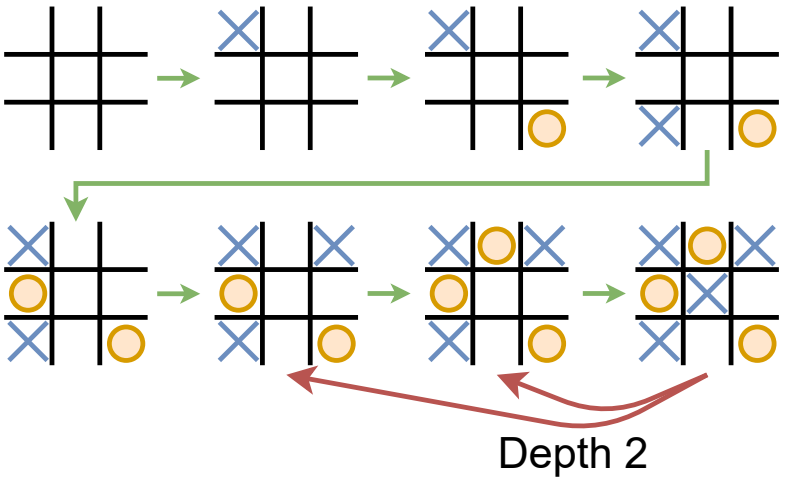


Figure 3.8: Play trace of Tic-Tac-Toe with the selection of the root node of the game end tree.

Figure 3.8 visualizes a play out of Tic-Tac-Toe. The example depth of the tree is two. Therefore, only the last two game states need to be stored in order to go backward. When we have selected the root node by going n nodes back, the process of generating the end game tree is the same as generating the full game tree. Usually, in the end, the options for moves are limited, and consequently, the shape of the tree isn't as uniform as at the beginning of the game.

As there are multiple possible ends to a game, multiple end game trees are generated. Following, they are randomly matched to the end game trees of the comparing game, and the respective tree edit distances are averaged over each pair.

3.2.1.2 Alpha-Beta Tree

In order to find the optimal move to make in a specific situation, the Minimax search algorithm can be applied. The algorithm searches the game tree and designates each terminal node either a positive value for a win, a negative value for a loss or a zero in case of a draw out of the root node players perspective. The leaf node values are propagated upwards by selecting either the highest or lowest value depending on who is choosing in the respective node. The beginning player tries to maximize and the opponent to minimize. The best move is selected by choosing the root node move with the highest value. This algorithm is initially intended for two-player games but can be generalized to work for multiple players.

“A technique called "alpha-beta pruning" is generally used to speed up such search processes without the loss of information.” (Knuth & Moore, 1975). Figuratively speaking, this speed-up is achieved by cutting off parts of the tree that do not yield to the search for the optimal move.

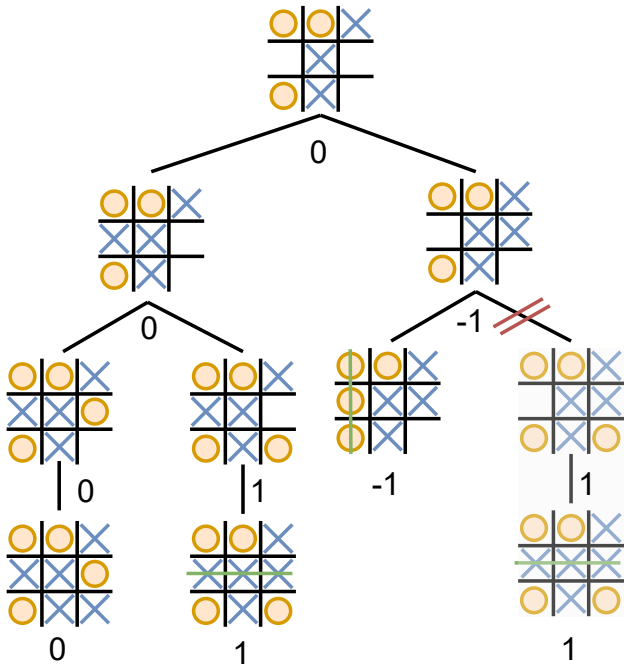


Figure 3.9: Play trace of Tic Tac Toe with the selection of the root node of the game end tree.

Source: Example taken from: https://materiaalit.github.io/intro-to-ai-17/part2/

An example of how the alpha-beta pruning technique is used is shown in figure 3.9. From the cross player’s perspective, by following the first move and placing the cross on the middle left, she can archive at best a draw under when the circle player plays

optimally. When playing on the right side, the circle player can win in one move. As the cross player knows that she can archive a draw by playing the first move, there is no merit in investigating the branch where she loses further. Therefore the remaining branches of that path can be pruned.

By utilizing that technique, differently shaped trees can be generated in which, for the player, uninteresting parts are pruned off. Similar games might have the same strategy and, therefore, similar shaped trees.

3.2.1.3 UCT Algorithm

A different approach can utilize artificial intelligence to find the best move is the Monte Carlo Tree Search (MCTS) algorithm. "MCTS [...] is based on randomized explorations of the search space. Using the results of previous explorations, the algorithm gradually grows a game tree [...], and successively becomes better at accurately estimating the values of the most promising moves"(Chaslot et al., 2008).

In order to grow the game tree, the algorithm utilizes four strategic phases that are repeated as long as there is time left.

- **Selection** - The phase in which a new leaf node is selected randomly that is not yet added to the tree.
- **Expansion** - Adding the selected leaf node to the tree.
- **Simulation** - Doing a role out / play out to find the win / loss ratio.
- **Backpropagation** - Propagating the result of the simulation upwards to the root node updating the win / loss ration of each node.

For the task of comparing games, based on the generated tree, the win/loss ratio isn't necessarily required as this approach would not be much different from creating a random game tree. Considering the win/loss ratio in the selection phase would result in a more representing tree that holds more information about the game. The Upper Confidence Trees (UCT) algorithm tries to balance the exploration and exploitation in the selection by taking into account the win/loss ratio and the number of times a specific node has been visited. The child node j that maximizes the formula given in equation 3.3 is selected.

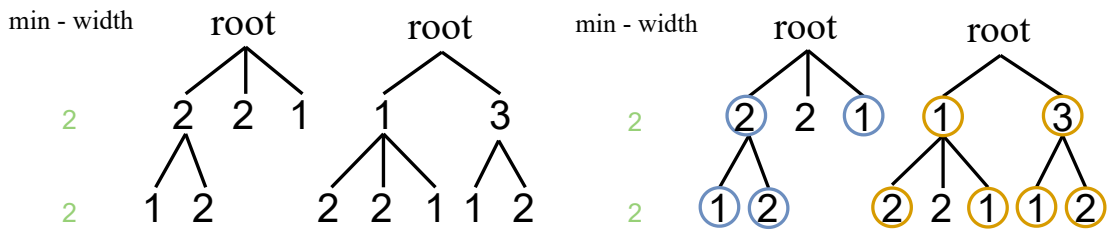
$$UCT_j = X_j + C \cdot \sqrt{\frac{\ln(n)}{n_j}} \tag{3.3}$$

Where X_j is the win loss ratio of the child, n is the number of times the parent has been visited, n_j is the number of times the child has been visited and C is a constant to adjust the amount of exploration.

3.2.2 Processing the trees

The Tree Edit Distance is an NP-hard problem. (Bille, 2005) Put another way, the distance metric's calculation time grows exponentially with the size of the trees. Therefore, the distance metric becomes unfeasible for bigger game trees. Bigger games with more possible moves result in bigger trees and generally need to be constrained in their height.

Another approach to limit the size of the tree is by limiting the branching factor of the tree. Matching the trees in minimal branch or max width per layer was one approach to speed up the calculation of the distance metric without losing too much information. Minimal branch means that firstly we sample branches of the tree to get the minimum width of both trees, and the max width parameter is a given hyperparameter.



(a) Finding the minimal width per height of a tree

(b) Selecting different nodes.

Figure 3.10: Representation of the Tic-Tac-Toe description in Ludii as a tree (Browne, 2020).

Exemplary in Figure 3.10 (a) the min branching width is two for each layer. After the minimal branching factor is found, the selection of the different labels per layer is done. Only less than the determined min width of that layer can be chosen, or if that value is bigger, then less than the given max width parameter.

A drawback of this approach is that in bigger trees, the tree's unique shape is lost, and the width of the tree becomes uniform. If not time but accuracy is of the essence with the given hyperparameter object, the ability to turn off the pruning is given to the user of the distance metric calculation. Further more depending on the expressiveness of the used labeling algorithm, pruning down the trees could remove

the differentiating features of the trees. If the labeling algorithm designates the game state the same labels then the trees become the same and the resulting score would be 1.

The tree edit distance is order dependent. Consequently, if the children of a node are not in the same order as in the comparing tree, the distance metric results in a higher value. For comparing games, there is no apparent order in the possible moves of a state. Accordingly, the higher distance value is not correct as the distance metric should be order independent in this case.

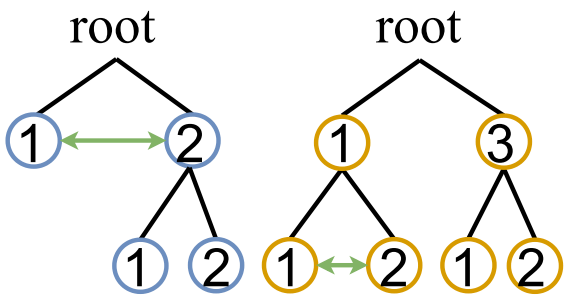


Figure 3.11: Sorting the labels.

The simple solution to this problem is the ordering of the children by the given label. Figure 3.11 shows that process.

3.3 Play Trace Distance Metric

Based on the behavior during the play, which we can get through simulation (Phenotype approach), a Play Trace similarity method was developed. Two games playouts are compared, and output that describing the play trace difference is returned. There already exist other metrics defined for play traces similarity measures, but with their properties relying on the cost of changing one play trace into another (J. Osborn et al., 2014) or simply based on the visualization of play traces (J. C. Osborn et al., 2018). These approaches have some limitations of size and variety of games that we want to avoid after our first Tree Edit Distance metric approach.

3.3.1 Concepts

3.3.1.1 Play Trace

A gameplay trace consists of a sequence of moves played in that game, that is, a number of actions that occur in a particular order. This basic definition is needed to understand our goal of finding patterns between different play traces games. Having two play traces, we can find similarities in sequences of moves. As the Label algorithm is used, a gameplay trace has the following form in Figure 3.12.

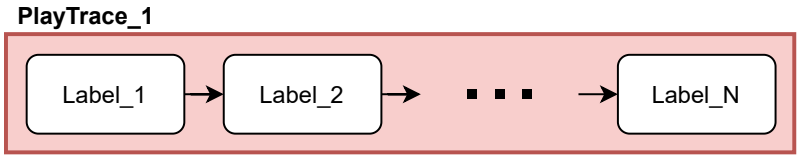


Figure 3.12: Generic form of a Play Trace

3.3.1.2 N-gram Algorithm

Nowadays, n-gram models have been shown to be very effective in the modeling language. However, n-grams can also be used for sequences of practically any type of data. In our case, an N-gram based model is used in order to be able to inspect similarity by comparing 'n' moves-chunks each by each. A full play trace can be split in smaller grams, even in grams of size one, which for example, are interesting when comparing individual moves. More formally, we define the n-gram play traces as follows in Figure 3.13.

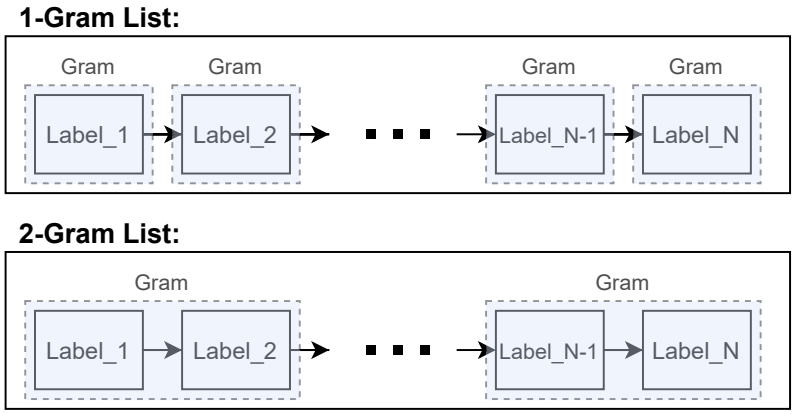


Figure 3.13: Generic form of a 1-Gram and a 2-Gram lists

3.3.2 Approach and Methodology

3.3.2.1 Play Trace Generator

The play traces for a game are generated specifying how many number of different play traces are needed. The reason behind this is that the more play traces, the more moves and actions are analyzed. In section 4.1, the more in-depth explanation is given.

We take into account two ways of generating the play traces, the Random generation and the AI generation.We assume at this point that the first one allows us to explore actions that the AI rarely would choose. As the proper name says, the play traces are composed of random moves. On the other hand, AI generation has to be taken into account because the moves also describe a game that a human would possibly do.

The two methods differ in how the next move that is going to be stored in the play trace list is selected. In the first one, the move is selected randomly. In the second approach, an AI agent is used as it follows the game working context for selecting the next move.

3.3.2.2 Play Trace Comparison

Once we know how the labeling algorithm is applied and how play traces are generated, we compare two play traces by comparing two lists of lists of game labels.

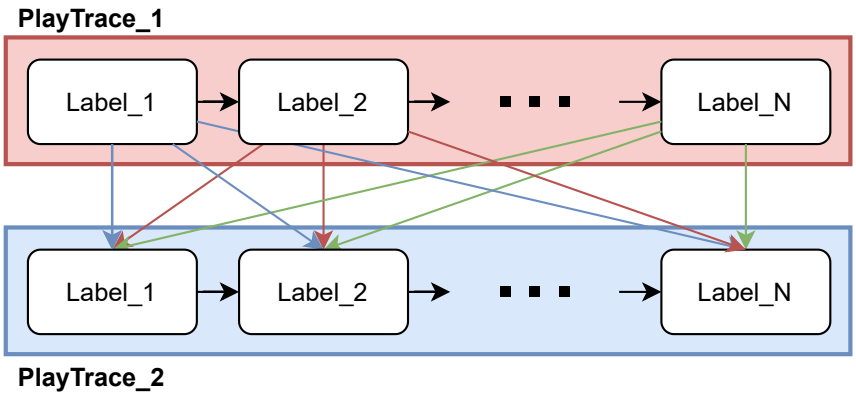


Figure 3.14: Visualization of Play Traces comparison

These two lists of lists and the maximum number of n-grams we want to consider are the first elements taken into account for the final similarity result. Every individual

number of n-gram, together with both lists of lists, is analyzed, creating the n-grams lists from the game labels for every play trace (i.e., list of play traces). The results for every n-gram, as it depends on both n-grams sizes, are normalized by dividing over the multiplication of both n-grams sizes. The total result of these steps is averaged over the number of n-grams comparisons done.

The inspection on the lists of n-grams is done by analyzing each n-gram, which is a list of game labels, and each game label can be compared using the approach explained in Section 3.1.3 and showed in Figure 3.15. Since we can have a different number of actions per game label, the result of this comparison also reflects this size difference. This is possible as the fourth labeling approach is used, so a fractional result is outputted.

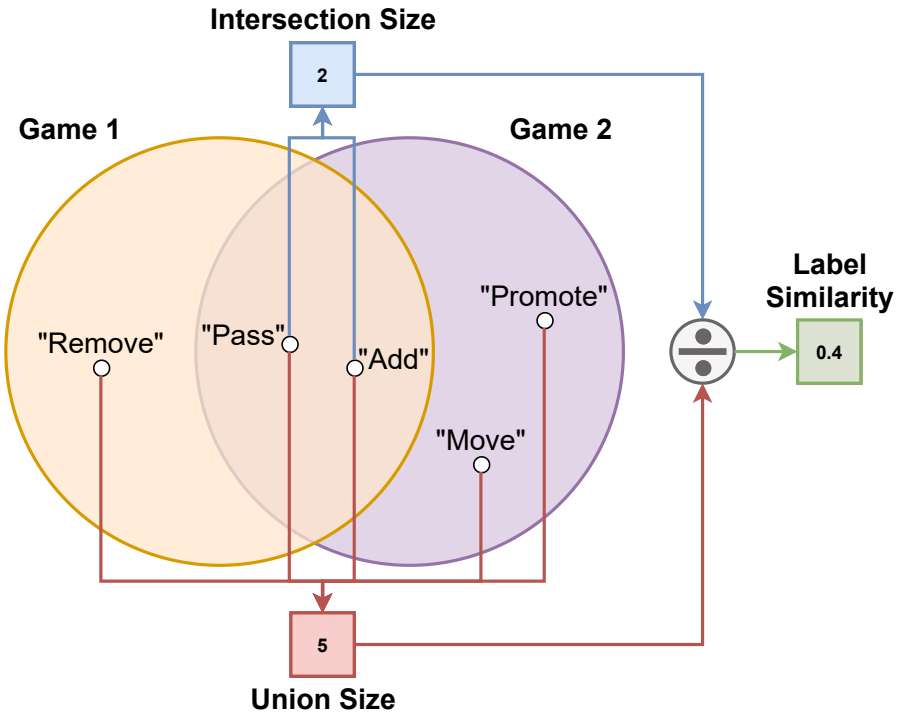


Figure 3.15: Visualization of Labels comparison

4 Results

4.1 Number of play traces

To determine the optimal number of play traces, we pick two games, e.g., Half Chess and Mini Shogi, and run several repeats of game comparisons with a different number of play traces. After we have collected the data, we compute the average difference between the same number of play traces to see how many we should use to find the most robust number of play traces.

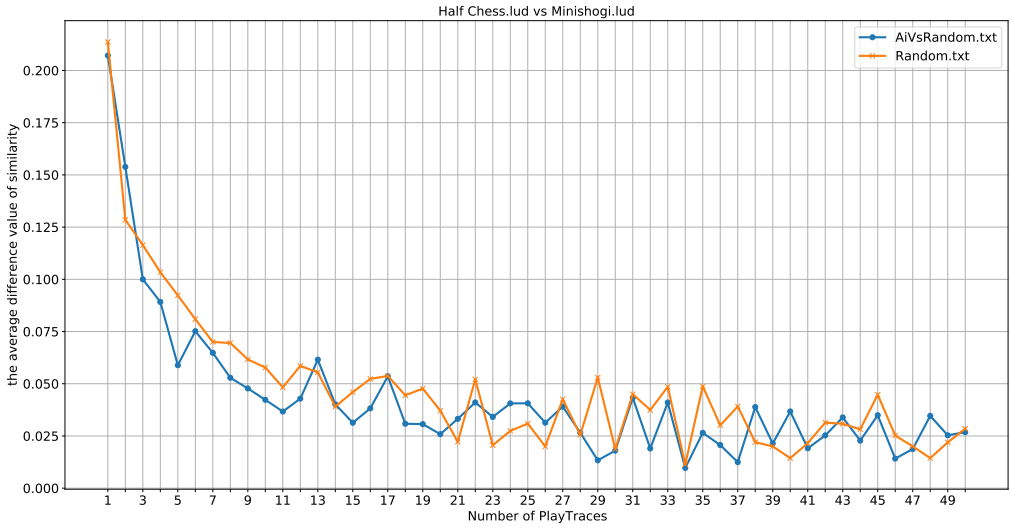


Figure 4.1: Comparison of the average difference value of similarity for each number of play traces.

As with most hyperparameters, the amount of to be used play traces is game dependent. Also, it corresponds with the amount of time the AI has to select a move. The assumption is that when utilizing the AI, the results obtained using the play trace method would be more robust. The results of the experiments are shown in Figure 4.1. With bigger games, it seems the convergence is slower, and the use of more play traces is reasonable. With Half Chess and Mini Shogi, there is no big difference between the AiVsRandom and only Random method can be seen.

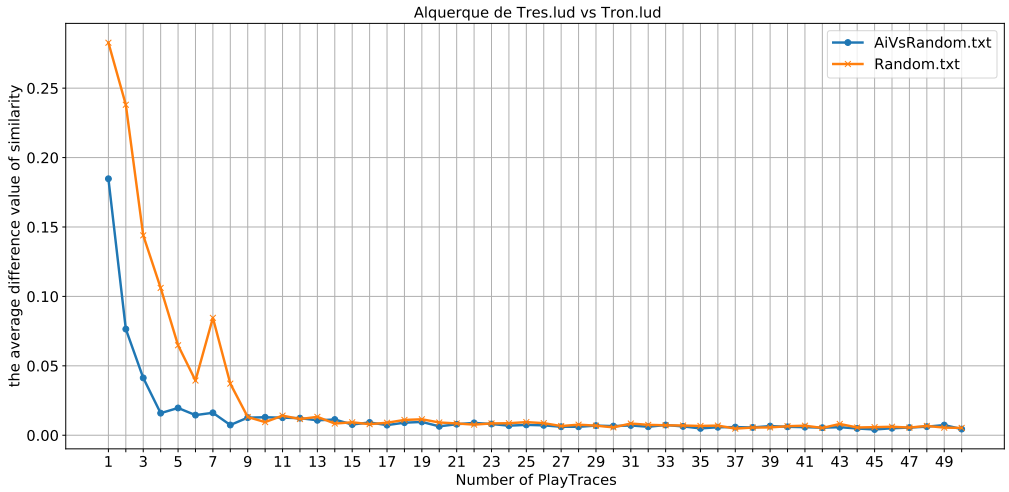


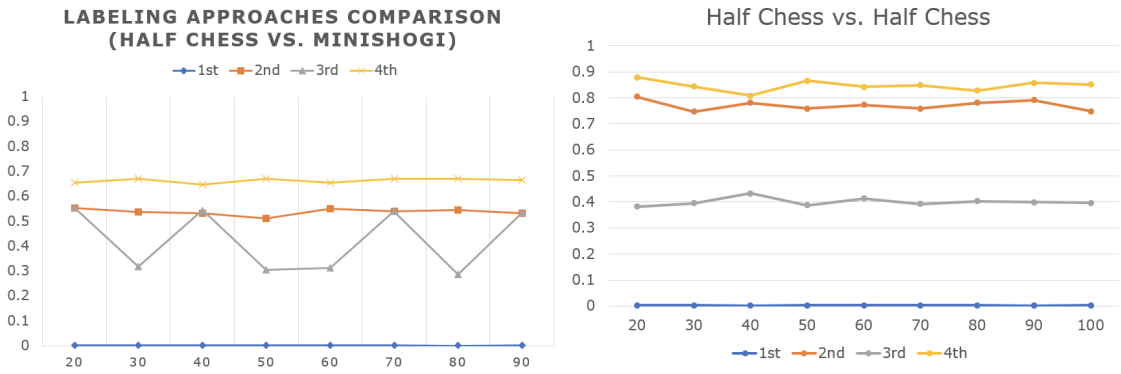
Figure 4.2: Comparison of the average difference value of similarity for each number of play traces.

The results of the smaller game experiment are shown in Figure 4.2. In smaller games, the amount of needed play traces actually converges to an amount where they have a negligible difference.

4.2 Labels comparison

The results of comparing algorithms on the example of comparing two different games are shown in Figure 4.3a. As one can see, the ActionWalkLabeling algorithm (third gray graph) shows low reliability compared to other algorithms that are more robust. It can be seen that the MoveStringLabeling algorithm (first blue graph) is inefficient for PlayTraces since no matter what games are compared, the result is always close to 0. Comparing the PlainActionLabeling and the ActionXORLabeling (fourth yellow and second orange graphs, respectively) algorithms, the PlainAction-Labeling algorithm performs better. It captures more game similarities due to the fractional similarity between labels.

The results of comparing algorithms on the example of comparing two similar games are shown in Figure 4.3b. It should be noted that the same playing distance is affected by the fact that the play-outs are random. That makes the method less accurate but faster and captures the game's sequence, that is, how similar matches of the same game are.



(a) Labeling algorithms comparison on different games.

(b) Labeling algorithms comparison on the same games.

Figure 4.3: Labeling algorithms comparison.

4.3 Game Comparison

As the calculation of the game distance takes time, we reduced our selection of games to shorter ones with the assumption that they also have a smaller branching factor. In order to filter the games, we go through every single game and generate ten play traces. Following, we averaged the play traces' length and selected the one with less than 25 moves. The list of games we computed the avg of can be seen in appendix A.

Figure 4.4 shows a comparison of the obtained average results of the metrics of games from the same folder and games from different folders. It can be noticed that for each metric, the average result is always higher for games from the same folder, which shows the correctness of the metrics and their ability to distinguish between games from the same folder (which means that the games are similar) and games from different folders.

Figure 4.6 shows the robustness of each metric. Each pair of Game X and Game Y games is compared, and then a comparison of Game Y and Game X is made to show how different results are. The difference between obtained values is calculated, and all results are averaged over all pairs for each metric. One can notice that endTreeDistance and uctTreeDistance algorithms show low robustness due to the randomness in the creation process. On the contrary, random play traces are more robust because the play trace generation process is computationally effective, and we generate a huge number of play traces. Hence, this method shows better robustness. Another interesting fact that was discovered was that AI random play traces have low robustness, which was unintuitive. It seems the used UCT Algorithm to generate the AI play traces is not the best choice and needs to be investigated further. Also,

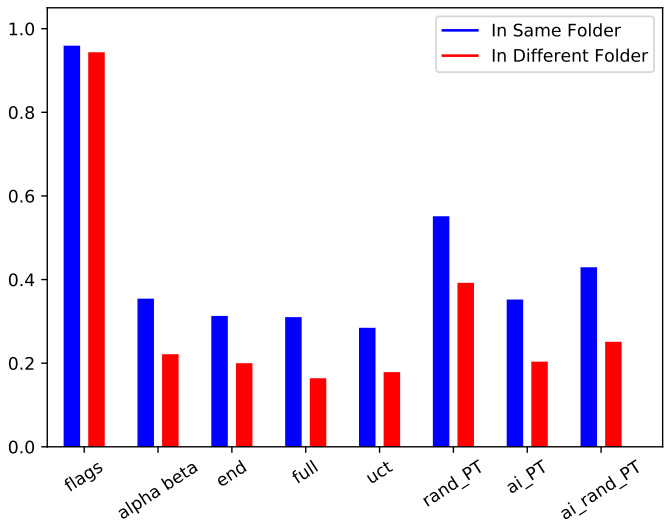


Figure 4.4: Comparison of average scores of algorithms of games in the same folder and games from different folders.

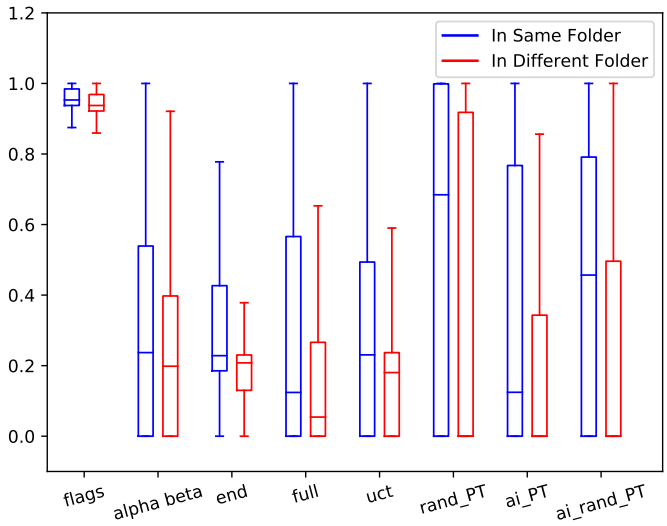


Figure 4.5: Comparison of scores of games in the same folder and games from different folders.

in future work, one can extend the AI play trace generator to create not only one play trace.

Figure 4.7 and Figure 4.8 show the relative games comparison inside the Shogi folder and the Territory folder, respectively. A comparison of each game inside the folder with each other is performed. Each game has n variables (n is the number of games inside the folder), where each value represents the relative similarity measure with

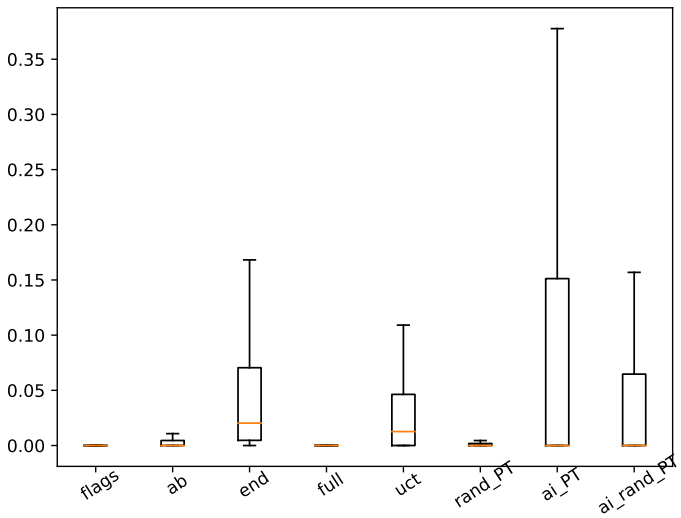


Figure 4.6: Comparison of the robustness of algorithms.

the corresponding game. Having this data, we apply the PCA technique in order to be able to visualize it and see how each game is relatively placed with respect to other games from the same folder. It was found that two principal components have 90- 95% variance in total (i.e., two principal components explain 90-95% of the data). One can see on Figure 4.7 that Dice shogi is quite distant from the cluster of other games. It should be said that this is just relative distance (because, for instance, the average score for Dice shogi is 0.5, whereas, for other games in this folder, it is around 0.8) that gives us an intuition of how games are placed with respect to each other. On Figure 4.8 showing how the games are located in the Territory folder, we can see that two main clusters appear.

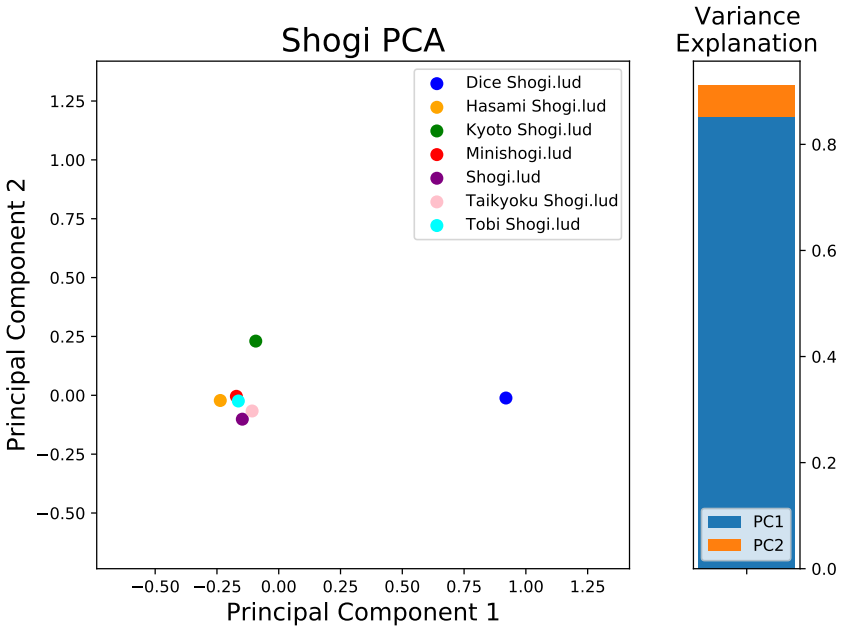


Figure 4.7: Shogi folder games comparison.

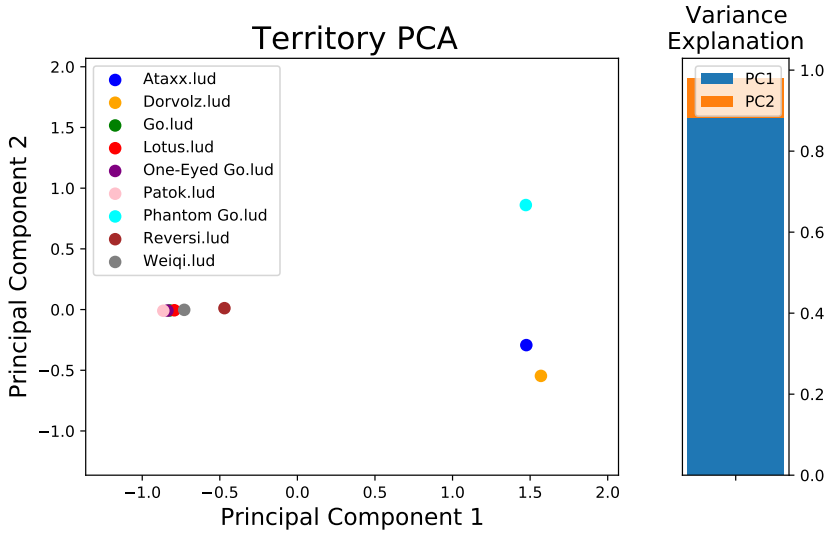


Figure 4.8: Territory folder games comparison.

5 Conclusions

We have shown how to utilize different phenotype approaches in order to get an estimate on how similar games are.

Utilizing existing distance metrics on the phonetic structures generated between two games proved to be a viable method to calculate a distance metric.

Further, through the different labeling algorithms, we investigated the various aspects of games that allow the functional distance between two games to be reliably measured.

The reliability of our distance metric has been shown to be heavily dependent on the used hyperparameters for the distance metrics. As our calculated distance metrics of the selected games have shown that the results match with the prior made classification by the ludii team, a certain assurance to the reliability has been made. Further, prove can only be made with the help of experts in the field.

5.1 Future Work

We created the groundwork for using the phenotype structures to compare games. With the limited knowledge about games and the Ludii project, interpreting the resulting distance metrics is sometimes difficult. Therefore to get a deeper understanding of our algorithm's work (potential shortcomings and identification of weaknesses), it is necessary to analyze our data with an expert.

5.1.1 Hyperparameter tuning

We found that most of the distance metrics used are heavily dependent on the parameters used for them. Further, the hyperparameters are dependent on which game is tested. For example, the calculation of the TED of bigger games is not

feasible, and therefore, either tree pruning or only play trace similarity needs to be used.

For some types of games, it may be useful to require a dynamic change of labeling algorithm based on the games that are to be compared. There is, in fact, the chance that a certain type of labeling might not be the best solution for all the possible comparisons and that different types of games may require different labeling approaches. This represents a hard challenge and requires the collaboration of experts to individuate all the essential characteristics and types of game. A major drawback of a similar approach would be the need for constant updates every time a new game is added to the Ludii library and would not lend itself well to a generalized metric such as the one that has been researched for in this project. Nevertheless, it represents a valid and possible approach. Additional experiments are needed to explore different combinations of labeling algorithms with a large number of game types.

Future work includes the investigation into a mechanism that tunes the hyperparameters automatically depending on the comparing game. Further is the dynamic weighting of the individual distance metrics depending on the compared games to be investigated.

5.1.2 Association rules

Investigating further different approaches to analyze the play traces is also part of future work to make the distance metric more stable.

"A methodology known as association analysis [can be used] [...] for discovering interesting relationships hidden in a large data set. The uncovered relationships can be represented in the form of association rules or sets of frequent items." (Tan et al., 2005) It allows finding and determining the item sets of moves in games, which in turn can allow comparing the similarity of games by matching similar item sets of moves.

It could be the case that this approach may reveal different aspects of the games than the comparison of n-grams and give a different view of the games' similarity.

List of Figures

List of Tables

Bibliography

Bille, P. (2005). A survey on tree edit distance and related problems. *Theoretical Computer Science*, *337*(1), 217–239. https://doi.org/https://doi.org/10.1016/j.tcs.2004.12.030

Browne, C. (2020, September 29). *Game design*. Retrieved January 17, 2021, from https://canvas.maastrichtuniversity.nl/courses/3128/files/169570?module_item_id=59723

Chaslot, G. M. J. .-B., Winands, M. H. M., & van den Herik, H. J. (2008). Parallel monte-carlo tree search. In H. J. van den Herik, X. Xu, Z. Ma, & M. H. M. Winands (Eds.), *Computers and games* (pp. 60–71). Springer Berlin Heidelberg.

Heinze, E., Bapat, P., Wu, Z., Alferez, S. V., oria Ladeira, A. V., & Niebisch, M. (2020). *Technical report of measuring game distance* (tech. rep.) [Supervisor: Cameron Browne and Steven Kelk]. Game AI and Search Group at Maastricht University Netherlands.

Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial intelligence*, *6*(4), 293–326.

Li, Y., & Chenguang, Z. (2011). A metric normalization of tree edit distance. *Frontiers of Computer Science in China*, *5*(1), 119–125. https://doi.org/10.1007/s11704-011-9336-2

Osborn, J., Samuel, B., McCoy, J., & Mateas, M. (2014). Evaluating play trace (dis)similarity metrics. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *10*(1). https://ojs.aaai.org/index.php/AIIDE/article/view/12730

Osborn, J. C., Samuel, B., & Mateas, M. (2018). Visualizing the strategic landscape of arbitrary games. *Information Visualization*, *17*(3), 196–217. https://doi.org/10.1177/1473871617718377

Stephenson, M., Piette, É., Soemers, D. J. N. J., & Browne, C. (2019). An overview of the ludii general game system. *2019 IEEE Conference on Games (CoG)*, 1–2.

Tan, P.-N., Steinbach, M., & Kumar, V. (2005). *Introduction to data mining, (first edition)*. Addison-Wesley Longman Publishing Co., Inc.

A Tested game distances

- Coyote.lud
- Go with the Floe.lud
- Hat Diviyan Keliya.lud
- Huli-Mane Ata.lud
- Ko-app-pawna.lud
- Len Cua Kin Ngoa.lud
- Mysore Tiger Game.lud
- Neg Tugal Tuux.lud
- Pon Chochotl.lud
- Sher Bakr.lud
- Shi Liu Kan Tsiang Kun.lud
- Wolf und Schaaf.lud
- Aralzaa.lud
- EinStein Wurfelt Nicht.lud
- Gauntlet.lud
- Knightthrough.lud
- Tugi-Epfe.lud
- Alitev.lud
- Kalah.lud
- Kapana Bona.lud
- Pachgarhwa.lud
- Um el Tuweisat.lud
- Blue Nile.lud
- Cram.lud
- Do Guti.lud
- Domineering.lud
- Dorvon Cag.lud
- French Military Game.lud
- Madelinette.lud
- Oumoul Kono.lud
- Pong Hau K'i.lud
- Quantum Leap.lud
- Snowpaque.lud
- Spots.lud
- Susan.lud
- Temeen Tavag.lud
- Tron.lud
- Three-Player Hex.lud
- Trax.lud
- Feed the Ducks.lud
- Manalath.lud
- Akidada.lud
- Alquerque de Tres.lud
- Andantino.lud
- Bravalath.lud
- Connect Four.lud
- Djara-Badakh.lud
- Driesticken.lud
- Dris at-Talata.lud
- Engijn Zirge.lud
- Epelle.lud
- Fanorona Telo.lud
- Janes Soppi (Alignment).lud
- Les Pendus.lud
- Liu Tsi.lud
- Nine Holes.lud
- Niranchy.lud
- Pentalath.lud
- Picaria.lud
- Round Merels.lud
- San-Noku-Narabe.lud
- Score Four.lud
- Shisima.lud
- Spline.lud
- Squava.lud
- Sujjua.lud
- Symbol S.2.lud
- Tant Fant.lud
- Tapatan.lud
- Three Men's Morris.lud
- Tic-Tac-Chess.lud
- Tic-Tac-Die.lud
- Tic-Tac-Four.lud
- Tic-Tac-Mo.lud
- Tic-Tac-Toe-Mini.lud
- Tic-Tac-Toe.lud
- Tuk Tak.lud
- Ultimate Tic-Tac-Toe.lud
- Wure Dune.lud
- Yavalade.lud
- Yavalanchor.lud
- Yavalath.lud
- Upper Hand.lud
- Weiqi.lud
- Brazilian Draughts.lud
- Game of Solomon.lud
- Tre Guti.lud
- Ja-Jeon-Geo-Gonu.lud
- Press Ups.lud
- Toono.lud
- Triad.lud
- Vela.lud

B Source Data for PCA

Dice Shogi	Hasami Shogi	Kyoto Shogi	Minishogi	Shogi	Taikyoku Shogi	Tobi Shogi
1.0000	0.5280	0.4469	0.4720	0.4702	0.4782	0.5348
0.5280	1.0000	0.8783	0.9172	0.9329	0.8634	0.9236
0.4469	0.8783	1.0000	0.7960	0.7618	0.7693	0.8166
0.4720	0.9172	0.7960	1.0000	0.8092	0.8213	0.8536
0.4702	0.9329	0.7618	0.8092	1.0000	0.8354	0.8536
0.4782	0.8634	0.7693	0.8213	0.8354	1.0000	0.8182
0.5348	0.9236	0.8166	0.8536	0.8536	0.8182	1.0000

Table B.1: Source matrix used for PCA and Shogi folder similarity plotting

Ataxx	Dorvolz	Go	Lotus	One-Eyed Go	Patok	Phantom Go	Reversi	Weiqi
1.0000	0.6056	0.0000	0.0001	0.0000	0.0001	0.0000	0.0000	0.0001
0.6056	1.0000	0.0000	0.0001	0.0000	0.0001	0.0000	0.0000	0.0002
0.0000	0.0000	1.0000	0.9191	0.9540	0.9774	0.0004	0.7338	0.8867
0.0001	0.0001	0.9191	1.0000	0.9360	0.9570	0.0007	0.7207	0.8621
0.0000	0.0000	0.9540	0.9360	1.0000	0.9794	0.0004	0.7431	0.8957
0.0001	0.0001	0.9774	0.9570	0.9794	1.0000	0.0001	0.7578	0.9059
0.0000	0.0000	0.0004	0.0007	0.0004	0.0001	1.0000	0.0002	0.0013
0.0000	0.0000	0.7338	0.7207	0.7431	0.7578	0.0002	1.0000	0.6878
0.0001	0.0002	0.8867	0.8621	0.8957	0.9059	0.0013	0.6878	1.0000

Table B.2: Source matrix used for PCA and Territory folder similarity plotting