General Game Playing with a Portfolio Agent using Machine Learning and GGP add-ons

Simona Vychitilova, Aaron Schapira, Ismail Gündüz, Elliot Doe, Meike Thijsen
Department of Advanced Computing Sciences
Maastricht University
Maastricht, The Netherlands

January 2023

Abstract

General Game Playing requires an agent to play all types of games optimally. In the Kilothon competition, an agent has to play over a thousand games, each with only one minute of thinking time. For this purpose, a portfolio agent is created, picking the best agent per game. Which agent should be selected for which game is determined by a machine learning model. This model was trained on a subset of the games as training on all games is not feasible. This subset was created by clustering games on their concepts and selecting representative samples from each cluster. After training this model, the portfolio agent achieves an average utility of 0.161 on all games in the Kilothon competition. In addition to this, general game playing features are added: a uniform time distribution and picking heuristics based on the game ending increase the utility further up to 0.2195, therefore beating all previous Kilothon participants.

1 Introduction

1.1 Background

General Game Playing (GGP) is a subsection of AI which aims to develop agents that can play all types of games. An agent is required to be able to play any arbitrary game based only on the game description. This means that the agent has to decide which heuristics to use in real time and cannot rely on features like opening books or set evaluation features.

Many algorithms have been developed for game playing each with varying performance for different games. Adversarial search is a technique where agents compete with conflicting goals. This is a very large sub field of AI and a lot of research has been done to develop algorithms that optimise game play.

Ludii [11] [14] is a digital general game system. It was developed by the Digital Ludeme Program (DLP) [4] team at Maastricht University to model, play and evaluate games. This project aims to improve our understanding of traditional games using modern AI techniques. Games in the system can range from card to board or even puzzles. All games are turn based but have a varying number of players and can sometimes also include stochastic elements.

The Ludii AI Competition [12] has been held by the Ludii team every year since 2020. The Kilothon is one of three possible tracks that involves participants testing a single agent against the inbuilt UCT agent for 1119 different games that satisfy the Kilothon conditions in the Ludii database. For each game it is given a utility between -1 and 1 based on its performance. At the end of the run the average utility is given as the score of the agent. In 2022 Cyprien Michel–Deletie [5] won the Kilothon. The goal of this project was to build an agent that would then be entered into this competition. The agent is limited to one minute of thinking time per game. After the thinking time is used up, the agent will be forced to play randomly until the end of the game. Additionally, after each player has played 500 turns the game concludes in a draw. This is slightly different to GGP as the list of games is predetermined and therefore preprocessing steps can be applied. All these games in Ludii also have a fixed concepts that can be accessed during play.

1.2 Approach

Since playing these 1119 games takes very long, the project aimed to create a subset that is representative of the entire game set. This subset could then be used in order to conduct experiments much quicker, but would still give meaningful results. Subsets like this have been created using various clustering techniques. Each subset was then validated to evaluate how the score of the subset represents the true score of the Kilothon. Based on these scores a single subset has been selected that could be used for testing purposes. The subset was also be validated by doing explainability to find which features are most important and seeing if these line up with what is expected. Experiments were then done on the subset in order to determine which agents perform best on each game. From these results a classification model was created that could predict which agent is best for each game based on its concepts. This resulted in a final agent called a portfolio agent, which contains a list of preestablished agents that can be deployed to play certain games based on the classification model.

Enhancements were also be added to the agents so that the agent will perform optimally in the Kilothon. This included generating an automatic evaluation function based on the given information of a game. The given algorithms were then able to evaluate states with more precision. Additionally, an algorithm was created to better distribute the allotted thinking time. The goal was to decrease how often the agent would be forced to play randomly.

1.3 Research Questions

1. Can games be clustered based on their concepts in a way that a valid subset can be created of the games in the Kilothon, and is an agents performance on this subset representative of how it would perform on the entire Kilothon?
2. Can an effective algorithm be developed to classify games by the AI that would be most effective at playing it?
3. What is the performance of the created algorithm in the Kilothon competition?
4. Does the created algorithm outperform the winner of the previous Kilothon?
5. Can the performance of the AI be increased by adding GGP enhancements to the single portfolio agents?

2 Methods

2.1 Clustering

In order to create a relevant subset of the games, clustering techniques are used. Clustering is a technique of grouping similar data points together into clusters without prior knowledge of their classifications. Clustering algorithms use various methods like distance measures to determine the similarity between data points and create clusters based on that similarity.

2.1.1 Data Preparation

To cluster the games, the correct data from the Ludii database had to be retrieved. The tables that were kept from the whole database are the following: *concepts, gamerulesets, games, rulsesetconcept, rulsesetconceptuct, rulesetconceptab*. By merging those different tables into a dataframe, a proper dataset suitable for clustering was created. Preprocessing was done in order to remove visualisation concepts as well as those that have identical values to others. A feature selection algorithm was implemented in order to reduce the dimensionality of the data.
First, Principal Component Analysis (PCA) [15] was applied. PCA is a technique for dimensionality reduction, it is a linear method that transforms the original data into a new coordinate system where

the first axis represents the direction of maximum variance, the second axis represents the direction of second maximum variance and so on. By using PCA, it is possible to reduce the number of dimensions in the data while still preserving as much of the original variation in the data as possible. Using this, the data went from 669 to 150 features, representing the concepts of the games. 150 was was selected as this represented 90% of the variance of the original data.

Since clustering is unsupervised, feature selection had to be done by comparing how well a model without a given feature can predict this feature. This method uses parallel processing to calculate the feature importance scores for each feature in the data. After computing the feature importance scores for every feature, they are saved. Based on this the best 75 performing features were taken as the data set.
Another dataset was created that contained three types of concepts: *Equipment, Rules.play, Rules.end.* Based on expert opinions, those three concepts are the most important ones for this kind of problem. PCA was also applied to this dataset.
With these techniques there are now four datasets for the clustering.

- All concepts
- All concepts with PCA and Feature Selection
- Three main concepts
- Three main concepts with PCA and Feature Selection

2.1.2 Clustering Techniques

There are various ways to apply clustering, with each having their own strengths and weaknesses. To select the best clustering technique on the game concepts, the following methods are explored: Agglomerative clustering, K-Means and BIRCH.

Agglomerative Clustering Agglomerative Clustering is a hierarchical clustering algorithm that starts by treating each point as an individual cluster, then iteratively merges the closest clusters into larger clusters until all points are in one single cluster or a stopping criterion is met. The algorithm can be used to create any number of clusters as it does not require a predefined value. The results of this technique will also be used in future clustering methods. The resulting dendogram is shown in Figure 1.

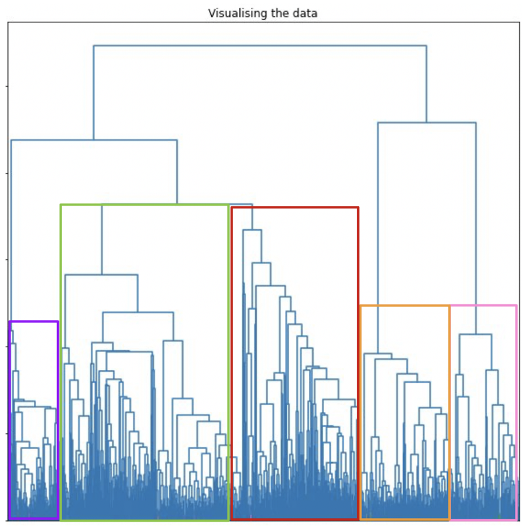


Figure 1: Hierarchical Clustering

From Figure 1, there are 5 clearly identifiable clusters. This number will be used as a benchmark for the number of clusters to define in the other clustering techniques presented in this report.

K-Means K-Means [10] is an unsupervised machine learning algorithm. It aims to partition a set of points into K (in this case 5) clusters, where each point belongs to the cluster with the nearest mean. The algorithm iteratively assigns each point to the cluster with the closest mean, and then updates the cluster means based on the newly assigned points, until the cluster assignments no longer change. K-Means requires the number of clusters to be decided beforehand. The results from the Agglomerative clustering are used to determine the number of clusters for K-Means.

BIRCH BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a density-based, hierarchical clustering algorithm. It builds a tree-like data structure called a CF-Tree (Clustering Feature Tree) to represent the clusters and their sub-clusters. BIRCH starts by creating a global cluster that contains all the points in the dataset. Then it iteratively splits the global cluster into smaller clusters by using a feature, which is a summary of the points in a cluster.

2.1.3 Subset Creation

One of the main goals of the study is to create a representative subset of the games, which will be done using two different techniques.
The first important part of the subset creation is to know how many games from each clusters will appear in the subset. In order to decide this, the following formula was used:

$$X = \frac{C}{threshold}$$

Where:

- X is the resulting number of games in the subset from the given cluster
- C is the size of the cluster
- $threshold$ is decided to be able to control the size of the subset

Random A random sampling approach was employed to generate a subset of the data. This was done by randomly selecting a specific number of games from each cluster. This can result in a diverse subset of the data.

Distance Based In the context of K-means clustering, a centroid is a representative point of a cluster, defined as the mean of all the points in the cluster.
An alternative strategy for generating a subset of the data is based on the distance of each point within a cluster from its centroid. Euclidean distance is used as the measure of distance. The subset is created using the closest point as well as a single far point. This approach ensures diversity by including an outlier point and good representation by selecting instances that are central to the cluster.

2.1.4 Final Subset

In order to determine the optimal combination of dataset and clustering technique for generating a representative and diverse subset, a series of experiments were conducted. The results of these experiments are presented and analyzed in Section 3.1.

2.2 Agent Methods

This section describes the agent that this paper proposes to enter into the Kilothon competition. First the portfolio agent will be described quite generally in subsection 2.2.1, then a more detailed explanation of the selection processes is provided in subsection 2.2.2. Finally, this section will describe two techniques that were used to make the agent more effective at general game playing in subsection 2.3.

2.2.1 Portfolio Agent

The agent that this paper proposes for general game playing is a portfolio agent. This portfolio agent contains a portfolio of other well-established game playing agents. Whenever asked to play a game, the portfolio agent needs to select from the portfolio how to play the game. The difficulty here is that a general game playing agent needs to be able to play games it has not seen before. There are various approaches that can be taken to decide on an agent. Two such approaches are described in subsection 2.2.2. The portfolio of this proposed agent contains five game playing agents: Alpha-Beta Search, PN-MCTS [7], UCT [9], MAST [2] and GRAVE [6].

All these agents are already implemented in Ludii and have different strengths and weaknesses. Alpha-Beta can use heuristics which can vary results. These are explored in Section 2.3.1. UCT is the opponent agent, and can thus be seen as a kind of 'safe' option, as playing an agent against itself typically results in a win rate around 50%. PN-MCTS is an agent that combines UCT with Proof-Number Search[7]. It was shown to be able to overcome some weaknesses of UCT in certain domains. A portfolio agent specifically tries to find agents that can take over when others are weak, so PN-MCTS could serve as a solution in some domains. Finally, MAST and GRAVE are both enhanced UCT agents as well. These enhancements could give an edge over the original UCT agent opponent that they are built on.

2.2.2 Agent Prediction

There are two strategies for agent selections that are implemented in this project. They are described in the following sections.

Clustering based The initial strategy for selecting the optimal AI for a given game is based on the clustering analysis conducted in Section 2.1. Given a game, the agent will classify it into one of the identified clusters and subsequently utilize the AI model that performed best on that cluster. This approach aims to ensure that the selected AI model is most suitable for the game's characteristics.

Machine Learning The second approach for selecting the appropriate AI for a given game uses a supervised machine learning model. Experiments were done on the subset in order to get an indication of which agent is best for each game. A Random Forest Classifier [3] was then trained on the subset to predict AIs based on game concepts. This was then applied to all games to get a prediction for the best AI.

2.3 GGP Features

Extra GGP features can be developed to potentially increase the utilities of the portfolio agent on the Kilothon. In this paper, automatic heuristics and new ways of distributing thinking time are explored.

2.3.1 Automatic Heuristics

Algorithms like Alpha-Beta rely heavily on their evaluation function. An AI for game playing is often only applied to one or one type of game. In these cases, an evaluation function can be handcrafted based on expert knowledge and time can be spent on tuning this. In General Game Playing, these options are not available, and thus generic evaluation functions need to be generated. This is the field of hyper-heuristics, where heuristics are generated, selected or combined to fit a subproblem [8].

Whether to select, generate or combine heuristics for a problem depends on the type of game and how much time there is for learning.

One way to approach hyper heuristics is generating an evaluation function based on meta-information of the game. When there is quite some time to play the game, multiple generated evaluation functions can be tested against one another and the best evaluation function can then be picked. One issue is that this testing needs time, and in the Kilothon competition an agent only has one minute per game.

Offline learning for generalized heuristic functions is therefore the go-to approach regarding the Kilothon. Stephenson *et al.* [13] used this type of learning to create a machine learning model to predict the best single heuristic per game. This was trained by running experiments on all games with evaluation functions consisting of one heuristic each time. This research only includes one heuristic per evaluation function, as otherwise the weights between these heuristics also has to be determined. Note that heuristics are split up by whether they are negative or positive, e.g. MaterialNeg and MaterialPos.

To continue from the research by Stephenson *et al.* these best single heuristics per game can be extracted, after which multiple evaluation functions can be tested by combining these n-best heuristics and the weights between them. It was chosen to do this training grouped by the endgame concept of a game as these show high correlation.

2.3.2 Time Distribution

Under the rules of the Kilothon each agent is allowed one minute of thinking time per game. It can distribute this over all the turns in any way. The opponent uses 0.5 seconds per move and then switches to random when time is run out. Some strategies were developed in order to measure the change performance. The DurationTurn concept in Ludii represents the average number of turns in a game for a specific agent. Since in the Kilothon the agent plays against UCT, this was the length of the game that was selected. Three strategies were developed for the agent to distribute its time.

The first strategy was a simple uniform time distribution. The time was calculated by dividing the minute by the length of the game. This time interval was then set as the maximum thinking time per move.

Another strategy would be to start with a lower thinking time and then increase it as the games go on. This could improve the model as it may result in the opponent running out of time and the agent would then have a lot of thinking time and be able to take advantage of the random player. For this a linear increase with a starting time of 0.1 seconds was selected. The rate of increase was then calculated so that a total thinking time of one minute would be reached when a game ends.

The final strategy was a linearly decreasing thinking time model. This means that an agent could use more thinking time in the beginning of the game and get an advantage. The initial starting time and rate of decrease are calculated so that at the expected end of the game the agent thinks for 0.1 seconds and has used its entire minute of thinking time.

3 Experiments

In this section, the experiments done to create and train the models are discussed as well as getting the final results of the portfolio agent.

3.1 Subset Validation

The main goal of the subset is to be able to run agents and get results that would give an indication as to how the agent would perform in the Kilothon. To validate which of the created subsets were best experiments were run with different pre-existing agents on the Kilothon. For each agent the score on the Kilothon and each of the subsets created was calculated. From this the correlation of each subset scores to the actual scores was calculated. To do this Pearson's correlation coefficient [1] was used. It is calculated using the following formula.

$$Correlation(x, y) = \frac{Covariance(x, y)}{\sigma_x \times \sigma_y}$$

This gives a value between -1 and 1. Correlations of -1 and 1 indicate perfect negative and positive correlation and 0 indicates no correlation.

If an agent were to win all games in the Kilothon then it would also win every game in the subsets. Similarly, if it was to lose all games. These values were also added into the scores of subsets in order to enforce a positive correlation. The subset with the highest positive correlation was then chosen to be the final result.

These experiments were done on different AI's in order to ensure that the subset was valid for agents with different strengths.

3.2 Subset Explainability

To see if there is are ways to explain why games are clustered in a certain way the main concepts for each cluster were calculated. Since each cluster is defined by its center point the distances between each concept per cluster can be calculated. A concepts importance in a cluster was defined as the average distance to this concept in the other clusters. This method will be able to identify outliers as the concepts of each centroid.

3.3 Heuristics

The heuristic that works best in each situation is dependent on the ending condition of the game, which is why defining heuristics per ending condition is an intuitive approach. Therefore, for automatic heuristic generation, all games were grouped by their EndGameConcept, this is a group of concepts in Ludii that describes the ending condition of a game. An example EndGameConcept is EliminatePiecesWin and EliminatePiecesLoss. Per EndGameConcept the average best single heuristics are selected based on the research by Stephenson *et al.*, as described in Section 2.3.1. The best two heuristic terms are selected and four variations are created: equal weights for both (1 and 1 respectively), first bigger weight (100 and 1), second bigger weights (1 and 0.001) and only the first weight (1 and 0). These four heuristics plus the basic heuristic are then run 10 times per game per EndGameConcept using an Alpha-Beta depth 2 agent against the basic heuristic (also Alpha-Beta depth 2). The best heuristic is then selected based on the mean utility. These new heuristics are added to the portfolio agent, where when Alpha-Beta is chosen as the agent, the EndGameConcept of a game are checked whether non-basic heuristic can be applied.

3.4 Time Distribution

To test which of the time distribution algorithms is best, each of them were tested on the subset using the portfolio agent that uses the classification model for agent prediction. These scores were then compared and the highest performing strategy was taken as the best.

4 Results

4.1 Subset Validation

To decide which subset will be used, the correlation between Kilothon scores and cluster scores was calculated. The results of all the experiments on subsets can be found in Appendix **??**. Best performing subset was K Means on filtered PCA data with distance selection and is displayed in Table 1.

The correlation of these columns is 0.999. This is clear as many of the scores are similar over all the different agents. Due to time restraints of the project each of these agents were only run once, which can cause a lot of variance in the data. This means the results can vary with runs and in order to get more stable results more experimentation would be required.

	Full Kilothon	K Means on filtered PCA data with distance selection
Alpha Bets	0.237	0.275
UCT	0.041	0.106
PN-MCTS	-0.093	-0.088
GRAVE	0.012	-0.04
MAST	0.111	0.110

Table 1: Performance of Different Agents on Kilothon and Subset

4.2 Subset Explainability

The three most important features for each cluster were found by calculating the average distance between each concept of the centroids. The following list displays these results.

- Cluster 0
  - Dice_AB
  - Num Dice_AB
  - Roll Frequency_UCT
- Cluster 1
  - Threat_AB
  - Forward Direction_AB
  - Can Not Move
- Cluster 2
  - Step Decision To Empty Frequency
  - Step Decision Frequency
  - Step Decision To Empty Frequency_UCT
- Cluster 3
  - Sow Origin First_AB

  - Sow Frequency_UCT
  - Sow Frequency
- Cluster 4
  - Add Decision Frequency
  - Add Decision Frequency_UCT
  - Add Decision Frequency_AB

It is clear that these clusters are able to separate types of games based on the concepts. Cluster 0 is basing it on die indicating that it contains stochastic games. Looking into this cluster it contains game like Backgammon. Cluster 1 has concepts that apply to chess variants and contains many variants of Chess and Shogi. In Cluster 2 it is clear that games with a step decision are involved. This means it contains games where each step is a move, for example Tower of Hanoi. Contrary to this Cluster 4 uses Add Decision Frequency indicating that it contains game where you add pieces to the board. This can be for example Go and Ultimate Tic-Tac-Toe. Cluster 3 has Sow concepts indicating the games are Manacala game like Kalah.

4.3 AI tests

To identify the best model for predicting the agent for each game, the portfolio agent strategies were played on the Kilothon. The cluster model used a single AI per cluster that was trained on the subset. A random forest classification model was also built and trained on the subset in order to predict which AI is best for each game based on its concepts. The results are displayed in Table 2

	Cluster Model	Classification Model
Score	0.1243	0.1614

Table 2: Performance of Portfolio Agents on the Kilothon

This shows that the classification model outperforms the clustering model on the Kilothon. This is likely due the model aiming to find concepts that are similar within games that have the same AI predictions. This is different as clustering finds similar games by concepts however has no indication of whether this means that the same strategies apply and same agents will perform similarly. They both however give a relatively good score.

4.4 Heuristics

Examples of new heuristics per EndGameConcept can be seen in Table **??**. This shows three end concepts where new heuristics are chosen if these apply to a game. In the NoMovesLoss concept, for example, the new heuristics give a negative weight to the material instead of the basic heuristics positive one. These three example trained heuristics all gave a higher utility than the basic heuristic, and were therefore implemented in the portfolio agent. One limitation of this research is that there was not enough time to do many experiments. An example of this is that preferably more experiments were run on all endgameconcepts so that the results are more stable. If more time was available, it would be possible to run more experiments and then based on this, train a machine learning model (e.g. RandomForest) on the output of the best weights between the heuristics. This would likely give better results on the final Kilothon score.

Table 4 displays the utility on the Kilothon before and after the automatic heuristics is added. It is observed that the utility increases after adding the extra heuristic features. The model predicts for 180 games that Alpha-Beta should be used.

	No Heuristics	Heuristics
Score overall	0.1614	0.1864
Winrate only AB	0.67	0.77

Table 4: Performance of the Portfolio Agent with and without Heuristic Prediction on Kilothon

4.5 Time Distribution

The three time distribution strategies were all tested on the subset in order to get an indication of which would be best to apply to the final agent. The results of this are displayed in Table 5.

Table 3: Example End Concepts and their best heuristics and the best weights between those heuristics.

	Heuristic 1	Heuristic 2	Weights
LineWin	RegionProximity5	LineCompletionHeuristic	Equal
NoMovesLoss	UnthreatenedMaterialNeg	MaterialNeg	FirstHigher
NoOwnPieces	MobilitySimpleNeg	InfluenceNeg	Equal

	0.5	Uniform	Increasing	Decreasing
Score	0.1614	0.1823	0.1725	0.1792

Table 5: Performance of the Portfolio Agent with Different Time Strategies on Kilothon Subset

This shows that overall a uniform time distribution was the best. This was the strategy that the final agent used on every game. It is however likely that the performance of these strategies highly differs between games. Future work could be done in order to build a model that would be able to predict the strategy based on the game. Unfortunately due to time constraints of this project it was not feasible.

4.6 Final Agent

The final agent was comprised of the ML portfolio agent and used a uniform time distribution and automatic heuristic prediction. On the full run of the Kilothon it scored 0.2195. In comparison when running Cyprien Michel–Deletie's agent a score of 0.189.

5 Conclusion

In this study, a portfolio agent was developed to achieve a high accuracy in general game playing. To achieve this, clustering was applied on all game concepts to create a representative subset of them. This was used to answer the question: Can games be clustered based on their concepts in a way that a valid subset can be created of the games in the Kilothon, and is an agents performance on this subset representative of how it would perform on the entire Kilothon? Results show that clustering can be applied after using PCA and Feature Selection, showing logical splits in the data. When using non-random sampling, a representative subset is found, which is proved to have relatively similar results as the Kilothon. The final agent had a utility of as 0.1944 on the subset and 0.2195 on the Kilothon.

This subset was used for experiments to train a machine learning method to answer the question: Can an effective algorithm be developed to classify games by the AI that would be most effective at playing it, and what is the performance of the created algorithm in the Kilothon competetion? A RandomForestRegressor was used to create this model, and proved to work well as score of the agent on the Kilothon achieved a utility of 0.161.

To increase the utility of the agent, time distribution methods and automatic heuristics were explored to answer whether the performance of the agent can be increased by adding general game playing enhancements. Both the time distribution and automatic heuristics show evidence of increasing the score drastically, increasing the utility to 0.2195. This score outperforms the winner of the Kilothon in 2022 as this agent scored 0.189.

The portfolio agent created in this study and its high performance is a step towards development of general artificial intelligence, where one portfolio agent has to pick the right AI for the right task.

References

[1] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[2] Y. Björnsson and H. Finnsson. CadiaPlayer: A simulation-based General Game Player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):4–15, 2009.

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] Browne C. The digital ludeme project. *Board Game Studies Colloquium (BGS 2018)*, April 2018.

[5] Michel–Deletie C. Heuristics-based ais for general game playing. 2022.

[6] Tristan Cazenave. Generalized rapid action value estimation. 01 2015.

[7] E. Doe, M. H. M. Winands, D. J. N. J. Soemers, and C. Browne. Combining monte-carlo tree search with proof-number search. In *2022 IEEE Conference on Games (CoG)*, pages 206–212, 2022.

[8] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. Recent advances in selection hyper-heuristics. *European Journal of Operational Research*, 285(2):405–428, 2020.

[9] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, pages 282–293, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[10] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[11] E. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne. Ludii – the ludemic general game system, 2019.

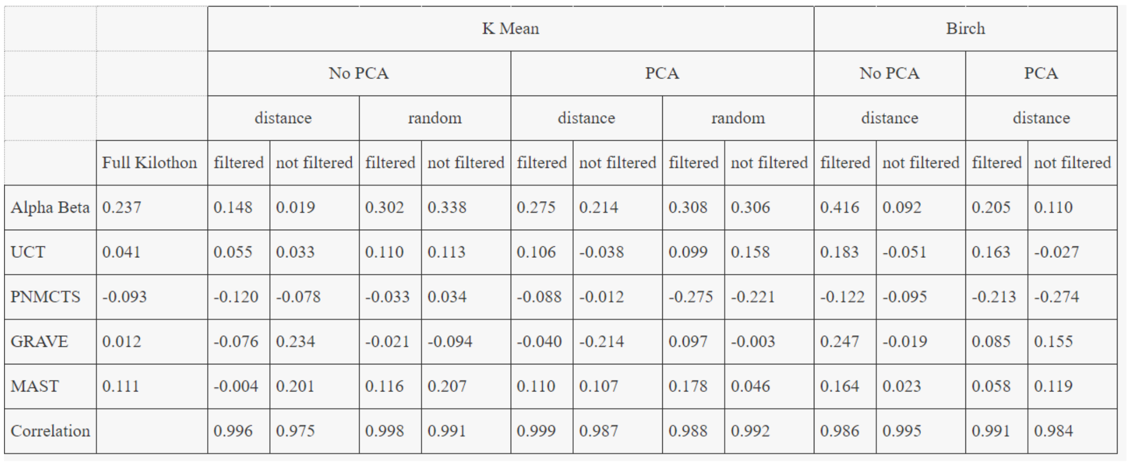
[12] GitHub Digital Ludeme Project. Ludii ai competition, 2022.

[13] M. Stephenson, D. J. N. J. Soemers, E. Piette, and C. Browne. General game heuristic prediction based on ludeme descriptions. In *2021 IEEE Conference on Games (CoG)*, pages 1–4, 2021.

[14] M. Stephenson, D. J. N. J. Soemers, E. Piette, and C. Browne. Ludii language reference. Dec 2020.

[15] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

A Subset Validations



		K Mean								Birch			
		No PCA				PCA				No PCA		PCA	
		distance		random		distance		random		distance		distance	
	Full Kilothon	filtered	not filtered	filtered	not filtered	filtered	not filtered	filtered	not filtered	filtered	not filtered	filtered	not filtered
Alpha Beta	0.237	0.148	0.019	0.302	0.338	0.275	0.214	0.308	0.306	0.416	0.092	0.205	0.110
UCT	0.041	0.055	0.033	0.110	0.113	0.106	-0.038	0.099	0.158	0.183	-0.051	0.163	-0.027
PNMCTS	-0.093	-0.120	-0.078	-0.033	0.034	-0.088	-0.012	-0.275	-0.221	-0.122	-0.095	-0.213	-0.274
GRAVE	0.012	-0.076	0.234	-0.021	-0.094	-0.040	-0.214	0.097	-0.003	0.247	-0.019	0.085	0.155
MAST	0.111	-0.004	0.201	0.116	0.207	0.110	0.107	0.178	0.046	0.164	0.023	0.058	0.119
Correlation		0.996	0.975	0.998	0.991	0.999	0.987	0.988	0.992	0.986	0.995	0.991	0.984

Figure 2: Subset Validation Results

B Subset

- Cluster 0
  - Los Escaques
  - Tayam Sonalu
  - Petol
  - Siga (Sri Lanka)
  - Pahada Keliya
  - Nebakuthana
  - Los Palos
  - Tawula
  - Kawade Kelia
  - Asi Keliya
  - Verquere
  - Pachih
- Cluster 1
  - Saxun
  - Breakthrough Chess
  - Rumi Shatranj
  - Lombard Chess
  - Ouk Chatrang
  - Persian Chess with a Queen
  - Cittabhramanrpasya Khelanam
  - Welschschach
  - Safe Passage
  - Chaturanga (12x12)
  - Sarvatobhadra
  - Shodra
  - Ploy
  - Taikyoku Shogi
  - Currierspiel
- Cluster 2
  - Tuknanavuhpi
  - Bam Blang Beh Khla
  - Coyote
  - Janes Soppi (Symmetrical)
  - Tides

- Mysore Tiger Game
- Musinaykahwhanmetowaywin
- Hund efter Hare (Vendsyssel)
- Merimueng-rimueng-do
- Pulijudamu
- N Puzzles
- Demala Diviyan Keliya
- Haretavl
- Juroku Musashi
- Thermenmuseum
- Mysore Tiger (Two Tigers)
- La Yagua
- Huli-Mane Ata
- Janes Soppi
- Mao Naga Tiger Game
- Yeung Luk Sz' Kon Tseung Kwan
- Orissa Tiger (One Tiger)
- Merimueng-rimueng
- T'uk T'uk
- Shui Yen Ho-Shang
- Komikan
- Jeu de Renard (Two Foxes)
- Ram Tir
- Hyvn aetter Hare
- Dam (Singapore)
- Orissa Tiger (Four Tigers)
- Bouge Shodra
- Yaguarete Kora
- Adugo
- Shiva
- Refskak
- Bagh Guti
- Gurvan Xudag
- Wolf und Schaaf
- Mysore Tiger (Three Tigers)

- Cluster 3
  - Hufesay
  - Andada
  - Tchela
  - Kanji Guti
  - Dabuda
  - Koro
  - Gamacha
  - Gabata (Ansaba)
  - Gabata (Ghinda)
  - Katro
  - Gabata (Wuqro)
  - Awagagae
  - Gabata (Oromo)
  - Motiq
  - French Wari
  - Four-Player Chess
  - Gabata (Adegrat)
- Cluster 4
  - Shisen-Sho
  - Hamiltonian Maze
  - Latin Square
  - Tic-Tac-Mo
  - Quantum Leap