

General Game Heuristic Prediction Based on Ludeme Descriptions

Matthew Stephenson, Dennis J. N. J. Soemers, Éric Piette, Cameron Browne

Department of Data Science and Knowledge Engineering

Maastricht University

Maastricht, the Netherlands

{matthew.stephenson,dennis.soemers,eric.piette,cameron.browne}@maastrichtuniversity.nl

Abstract—This paper investigates the performance of different general-game-playing heuristics for games in the Ludii general game system. Based on these results, we train several regression learning models to predict the performance of these heuristics based on each game’s description file. We also provide a condensed analysis of the games available in Ludii, and the different ludemes that define them.

Index Terms—Ludii, Heuristics, Ludemes, General Game Playing, Data Mining, Supervised Learning

I. INTRODUCTION

Ludii is a general game system focused on traditional board games [1]. It is being developed as part of the Digital Ludeme Project [2] but also provides opportunities for stand-alone AI research in the areas of agent development, content generation, game design, player analysis and education. Games within Ludii are described using ludemes, which are unique tokens or keywords that typically correspond to high-level game concepts. In this paper, we investigate if the performance of different game-playing heuristics can be estimated based on this ludemic game description format. If successful, this will allow us to predict the best heuristic for any new game, without the need for lengthy performance comparisons.

Many game-playing agents rely on heuristics to determine the best action to perform at a given time. In Chess, for example, one heuristic may encourage an agent to gain a material advantage, while another a positional advantage. Techniques that aim to predict the best choice from a collection of heuristics are commonly called hyper-heuristic approaches [3]. Such approaches have been proposed for different games and research frameworks, including the general video game AI (GVGAI) framework [4], [5], the general game playing (GGP) framework [6], Starcraft [7], Angry Birds [8], Pac-Man [9], Jawbreaker [10], FreeCell [11], and theoretical games [12].

The remainder of this paper is structured as follows. Section II describes the games and their ludemic format within Ludii. Section III describes the game-playing heuristics. Section IV describes the heuristic prediction methodology, experiments and results. Section V provides a short discussion of these results and possibilities for future work.

This research is funded by the European Research Council as part of the Digital Ludeme Project (ERC Consolidator Grant #771292) led by Cameron Browne at Maastricht University’s Department of Data Science and Knowledge Engineering.

II. LUDII GAMES

The Ludii general game system currently includes over 750 fully playable games.¹ The types of games within Ludii includes, but is not limited to:

- Deterministic / Stochastic Games
- Complete / Hidden Information Games
- Single-Player / Multi-Player Games
- Alternating / Simultaneous Move Games

For this paper, we will focus only on games that require two or more players, have an alternating move format, and are fully observable (no hidden information). This subset includes a total of 695 valid games.

A. Ludemes

Games within Ludii are described in terms of distinct ludemes. The number of ludemes used within each description can range from a few dozen for simple games (e.g. Tic-Tac-Toe) to several thousand for larger games (e.g. Taikyoku Shogi). Each ludeme represents a fundamental aspect of play, allowing for high-level structured descriptions. For example, the ludemic description for Tic-Tac-Toe is as follows:

```
(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)))
  )
)
```

Keywords such as Square, Piece, Line, etc. represent ludemes. Values defined by Strings, Numbers and Booleans (e.g. "Disc", 3, and True) are not considered ludemes.

The set of ludemes within the Ludii Language is continuously expanding but currently contains 547 ludemes. Some are used commonly, e.g. *Board* and *Players* are present in almost every game, while others are far less frequent, e.g. *Liberties* and *Enclose* are mostly found in Go-like games. By recording which ludemes are present within each game’s description, we can produce a complete ludeme dataset.

¹All values presented in this paper were obtained using Ludii v1.1.17.



Fig. 1. Example game clusters based on our ludeme dataset. Points were reduced to two dimensions using t-SNE.

B. Ludii Game Clusters

To get a better idea of the overall distribution of the games available in Ludii, we applied t-distributed stochastic neighbor embedding (t-SNE) [13] to reduce our ludeme dataset to two-dimensions, see Figure 1. From this, we can see 3 distinct clusters of games. The orange cluster contains 88 games, the green cluster contains 85 games, and the red cluster contains 522 games. Using a decision tree classifier, we identified which ludemes could be used to define this clustering.

The orange cluster contains games which have the *Sow* ludeme, a ludeme which is specific to the “Mancala” family of games. These games have a unique play-style, as they do not rely on traditional ideas of piece movement, placement and capturing in the same way that many other games do. Instead, players take turns “sowing” seeds along a ring of holes, often with many seeds located on a single space.

The green Cluster contains games which have both the *Track* and *Dice* ludemes. This is a common combination for the “Race” family of games, which includes popular examples such as Snakes & Ladders and Backgammon. The *Dice* ludeme means that dice are used as part of the game’s rules, and would indicate that there is some aspect of luck to the game. The *Track* ludeme indicates that there are fixed paths on the board which pieces must follow, often limiting the number of possible moves that can be made. Both of these ludemes together would suggest that the game has a limited degree of interaction, with the outcome of a player’s turn often being determined more by chance than skill.

The red cluster contains all other games which did not fit into either of the previous clusters, including games that have a track but no dice (e.g. Ja-Jeon-Geo-Gonu) and games with dice but no track (e.g. Dice Chess). It is likely that there other additional, albeit less clearly separated, clusters within this large set of games, but this was not investigated as part of this preliminary analysis.

III. LUDII AI HEURISTICS

Ludii contains several heuristics that may be used in heuristic state evaluation functions for game-playing algorithms that require them, such as $\alpha\beta$ -search [14]. The selection of heuristics implemented in Ludii is primarily based on the advisors that were found to be generally applicable and useful (not just for a specific game) in the earlier Ludi general game system [15]. The first two columns of Table I list names and short descriptions of each of them. For every heuristic, we include a positive and a negative variant, with positive and negative weights, respectively. For example, Material with a positive weight prefers game states where a player owns more pieces (in comparison to the opponents), whereas Material with a negative weight prefers game states where the player owns fewer pieces.

A. Heuristic Performance

For each of the 695 games, we aim to get an estimate of the “usefulness” of each heuristic for that game. This is done by measuring the average win-rates of $\alpha\beta$ -search agents using each heuristic against combinations of all other heuristics. To make this study feasible across many games, we restricted the agents to search depths of 2. Furthermore, we exclude heuristics from games that they are not applicable in, and instead estimate what their performance would have been by the performance of the “Null” heuristic, which always returns a constant value of 0. For example, the “Score” heuristic is inapplicable in games that do not use scores, “RegionProximity” is inapplicable in games that do not have any defined regions, etc.

More concretely, given an n -player game with a pool of k applicable heuristics, for every heuristic h , we generate all combinations of $n - 1$ heuristics out of the other $k - 1$ as opponents. If there are more than 10 such combinations (in games with many players and many applicable heuristics), we randomly sample only 10 combinations to retain. We play a minimum of 10 evaluation games for every such heuristic h against every combination of opponents, with a minimum of 100 evaluation games per heuristic h (plus more games when h itself is in the pool of candidate opponents for other heuristics).

The average win-rate of each heuristic across all 695 games is shown in Table I, for both the positive and negative variants. The number shown in brackets is the number of times this heuristic had the exclusive highest win-rate on a game (i.e. the number of games where it outperformed all other heuristics). Some heuristics (e.g. Material, Mobility, Influence) with positive weights have relatively high average win-rates, which suggests these heuristics may be good candidates for “default” heuristics in new games with zero domain knowledge. Almost all heuristics – including several with average win-rates around 50% – are still the top performers in some games, which suggests they are valuable to retain in a portfolio of general game heuristics. When using negative weights, all heuristics appear to have a poor level of performance on average, but most of them are still top performers in some games.

TABLE I

HEURISTICS IMPLEMENTED IN LUDII. THE SECOND-TO-LAST AND LAST COLUMNS SHOW THE WIN-RATES OF THE HEURISTICS AVERAGED OVER ALL GAMES (AND NUMBER OF GAMES IN WHICH A HEURISTIC IS THE SOLE TOP PERFORMER), FOR CASES WHERE THE HEURISTICS ARE ASSIGNED A POSITIVE OR NEGATIVE WEIGHT, RESPECTIVELY.

Heuristic	Description	Avg. Win % (# top performances)	
		Positive	Negative
Material	Sum of owned pieces.	62.09% (181)	39.88% (12)
Mobility	Number of legal moves.	57.75% (65)	43.58% (13)
Influence	Number of legal moves with distinct destination positions.	57.59% (53)	44.70% (6)
CornerProximity	Sum of owned pieces, weighted by proximity to nearest corner.	57.27% (44)	40.92% (9)
SidesProximity	Sum of owned pieces, weighted by proximity to nearest side.	56.36% (31)	43.86% (14)
LineCompletion	Measure of potential to complete line(s) of owned pieces.	54.86% (73)	44.28% (3)
CentreProximity	Sum of owned pieces, weighted by proximity to centre.	54.53% (37)	44.59% (13)
RegionProximity	Sum of owned pieces, weighted by proximity to a predefined region.	53.07% (21)	47.60% (7)
OwnRegionsCount	Sum of (piece) counts in owned regions.	50.35% (19)	47.66% (10)
PlayerRegionsProximity	Sum of owned pieces, weighted by proximity to owned region(s).	50.01% (2)	49.50% (2)
PlayerSiteMapCount	Sum of (piece) counts in sites mapped to by player ID.	49.99% (18)	47.86% (2)
Score	Score variable of game state corresponding to player.	49.83% (17)	47.39% (0)
ComponentValues	Sum of values of sites occupied by owned pieces.	48.68% (0)	48.39% (0)
Null	Always returns a value of 0.	48.52% (0)	-

IV. HEURISTIC PERFORMANCE PREDICTION

To determine if a game’s ludemes can be used to predict heuristic performance, we experimented with several regression learning algorithms. The names of these algorithms can be seen in Table II. We also tested a naive approach, which always predicts the mean of all label values. For each regression algorithm, a different model was independently trained for each heuristic. Each of these algorithm-heuristic models was trained across all 695 games, using the heuristic’s win-rate as the labels and the ludemes as the input features.

All results for this section are shown in Table II. All model training was performed in Python (v3.8.5) using Scikit-learn (v0.24.1) [16], with all regression learning algorithms trained using default hyperparameter values. All of the data and code used in these experiments is open-source and available online.²

A. Accuracy

To determine the accuracy of each trained model, we performed Leave-One-Out Cross-Validation and recorded the Mean Average Error (MAE) of each model. Due to space limitations, we could not show the MAE of each separate algorithm-heuristic model, but instead show the average MAE of each regression algorithm across all heuristics. RandomForestRegressor performs best overall, with an MAE of 7.48.

B. Expected Win-Rate

To get a better idea of how these models would compare if used by a portfolio agent, we also calculated the average expected win-rate of each regression algorithm. The expected win-rate of a regression algorithm on a given game, is equal to the true win-rate of its most preferred heuristic (i.e. the heuristic with the highest predicted win-rate) for that game. The average expected win-rate is computed by averaging over

all games. RandomForestRegressor once again performs best, with an average expected win-rate of 71.03%.

C. Regret

For a single game, the regret of a regression algorithm is defined as the expected reduction in win-rate as a result of picking its most preferred heuristic over whichever heuristic truly performs best. An increase in regret is inversely proportional to that of expected win-rate, but shows how close each regression algorithm is to perfect heuristic prediction. More formally, the regret $\mathcal{R}(h)$ for a regression algorithm that picks a heuristic h with a win-rate $w(h)$, in a game in which the best heuristic $h^* = \operatorname{argmax}_{h'} w(h')$ has a win-rate $w(h^*) \geq w(h)$, is given by $\mathcal{R}(h) = w(h^*) - w(h)$. The average regret is computed by averaging over all games. RandomForestRegressor has an average regret of 7.68.

V. DISCUSSION

Based on the results shown in Table II, the best performing regression algorithm (RandomForestRegressor) had less than half as much regret as the naive approach, representing a significant performance improvement. While not as large, the reduction in MAE was also substantial. This result confirms that the ludemes within a game’s description can be used to predict the performance of our heuristics, and that these predicted win-rates can be used effectively to select the best performing heuristic on new games.

There is also a very high correlation between the MAE and Regret results, with a lower MAE almost always correlating with a lower Regret. However, the improvement against the naive approach was much greater for regret than for accuracy (54% improvement versus 26% improvement). This would suggest that our prediction models are generally more accurate for games where the heuristics perform very differently, compared to cases where the performance is similar. In other

²https://github.com/Ludeme/HeuristicPrediction/tree/CoG_2021

TABLE II
HEURISTIC PERFORMANCE PREDICTION RESULTS. MAE VALUES FOR EACH ALGORITHM ARE AVERAGED ACROSS ALL HEURISTICS.

Regression Algorithm	MAE (stdev)	Win-Rate	Regret
RandomForestRegressor	7.48 (1.28)	71.03%	7.68
GradientBoostingRegressor	7.87 (1.54)	70.83%	7.88
KNeighborsRegressor	8.05 (1.39)	70.46%	8.25
BayesianRidge	8.29 (1.50)	70.26%	8.45
LinearSVR	8.56 (1.31)	69.97%	8.74
MLPRegressor	8.90 (1.40)	70.15%	8.56
ElasticNet	9.13 (1.83)	67.92%	10.79
Lasso	9.26 (1.81)	67.63%	11.08
Ridge	9.50 (1.76)	68.86%	9.85
DecisionTreeRegressor	9.70 (1.61)	67.95%	10.76
Naive	10.07 (2.58)	62.09%	16.62

words, if a game has low variance in terms of heuristic win-rate, then having a large MAE is typically going to have much less of an impact on our regret than if the game had high variance in heuristic win-rate.

A. Future Work

In this section we briefly describe several preliminary ideas for future work, including both extensions to and applications of the research proposed in this paper.

1) *Game Concepts*: Rather than using only the ludemes within our game descriptions, including additional game features or concepts may provide better results. Several higher level features have been previously proposed for general video games [4], [5], but these are limited to a few dozen examples and are typically not applicable to board games. We have already begun work on creating a corpus of game concepts that are detectable within Ludii games [17]. Combining these concepts with our existing ludeme dataset may help to improve our heuristic prediction accuracy, and could allow us to extend our results to other general game systems.

2) *Combining Heuristics*: While the method described here identifies a single best predicted heuristic per game, better agent performance can typically be achieved by using a weighted combination of several heuristics. For example, Material and Mobility tend to combine well for many games to encourage game states in which a player has a material advantage over their opponent(s) and more movement options. Unfortunately, the large number of potential combinations of heuristics, and the fine tuning of their weights, makes the identification of optimal weight vectors a difficult task for hundreds of games.

3) *Portfolio Agent*: One of the primary applications of hyper-heuristic approaches is the development of a portfolio agent. These agents have access to a wide range of different heuristics or agents, and select the best one for a given game. For our case, the general performance of a basic portfolio agent which selects a heuristic based on our RandomForestRegressor can be estimated from our expected win-rate. However, there are several other aspects to portfolio agent development which can also be considered, such as selecting an alternative heuristic should the first choice perform poorly.

4) *Strategy Recommendation*: Each of our heuristics is simple enough that we can use their predicted win-rates to provide simple strategies or tutorials for human players to follow. For example, the game of Go has high predicted win-rates for the Material (positive) and Score (positive) heuristics. Certain models, such as decision trees, may also be able to provide which ludemes are responsible for these predictions. This information could then be combined into a single complete message that provides some strategic insight, such as “This game contains the Remove and AddScore ludemes, you should focus on gaining a Material and Score advantage”.

REFERENCES

- [1] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, and C. Browne, “Ludii – the ludemic general game system,” in *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, ser. Frontiers in Artificial Intelligence and Applications, G. D. Giacomo, A. Catala, B. Dilkina, M. Milano, S. Barro, A. Bugarin, and J. Lang, Eds., vol. 325. IOS Press, 2020, pp. 411–418.
- [2] C. Browne, “Modern techniques for ancient games,” in *IEEE Conf. Comput. Intell. Games*, 2018, pp. 490–497.
- [3] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, “Hyper-heuristics: a survey of the state of the art,” *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.
- [4] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, “Matching games and algorithms for general video game playing,” in *AIIDE*, 2016.
- [5] A. Mendes, J. Togelius, and A. Nealen, “Hyper-heuristic general video game playing,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 94–101.
- [6] M. Świechowski and J. Mańdziuk, “Self-adaptation of playing strategies in general game playing,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 367–381, 2014.
- [7] D. Churchill and M. Buro, “Portfolio greedy search and simulation for large-scale combat in starcraft,” in *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, 2013, pp. 1–8.
- [8] M. Stephenson and J. Renz, “Creating a hyper-agent for solving angry birds levels,” in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE’17, October 2017, pp. 234–240. [Online]. Available: <https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15828>
- [9] P. Rodgers, J. Levine, and D. Anderson, “Ensemble decision making in real-time games,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 2018, pp. 1–8.
- [10] S. Salcedo-Sanz, J. M. Matías-Román, S. Jiménez-Fernández, J. A. Portilla-Figueras, and L. Cuadra, “An evolutionary-based hyper-heuristic approach for the jawbreaker puzzle,” *Applied Intelligence*, vol. 40, pp. 404–414, 2013.
- [11] A. Elyasaf, A. Hauptman, and M. Sipper, “Evolutionary design of freecell solvers,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 270–281, 2012.
- [12] J. Li and G. Kendall, “A hyperheuristic methodology to generate adaptive strategies for games,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 9, no. 1, pp. 1–10, 2017.
- [13] L. van der Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: <http://jmlr.org/papers/v9/vandermaaten08a.html>
- [14] D. E. Knuth and R. W. Moore, “An analysis of alpha-beta pruning,” *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.
- [15] C. B. Browne, “Automatic Generation and Evaluation of Recombination Games,” PhD thesis, Faculty of Information Technology, Queensland University of Technology, Queensland, Australia, 2009.
- [16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [17] É. Piette, M. Stephenson, D. J. N. J. Soemers, and C. Browne, “General board game concepts,” *CoRR*, 2021.