Game Playing Hyper-Agents

By

Nicholas Thompson

Thesis
Submitted to Flinders University
for the degree of

Bachelor of Engineering (Software) (Honours)

College of Science and Engineering
22/10/2024

TABLE OF CONTENTS

ABSTRACT

This research explored the viability and effectiveness of hyper-agent approaches to general game playing within the Ludii game system, a general game framework that defines games in terms of conceptual units called ludemes. Hyper-agents are defined here as game playing agents that use existing agents or algorithms to optimise game performance.

The study used agent and heuristic win rate data sourced from the Ludii database, along with game parameters (ludemes, concepts), to produce machine learning models designed to predict optimal agents and heuristics for novel games. Following this, three hyper-agents were implemented within the Ludii system, utilising the models. A portfolio agent designed to select the best predicted sub-agent at the beginning of a game, an ensemble agent that required each of its sub-agents to select a move, before making the most popular, and a weighted ensemble agent, which leveraged the machine learning models to weight the votes of its sub-agents, selecting the move with the highest vote weight each turn.

The study ran an experiment that had each of the hyper-agents and sub-agents play a representative set of 30 games, playing each 50 times for a total of 1500 playouts. The experiment results showed a statistically significant improvement in the performance of the portfolio agent over each of the sub-agents. Non-parallel implementations of the ensemble agent and weighted ensemble agent did not produce a statistically significant improvement over the best performing sub-agent, though their parallel counterparts did perform best overall, with the parallel weighted ensemble producing a statistically significant performance improvement over every other agent in the experiment.

DECLARATION

I certify that this thesis:

1. does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university
2. and the research within will not be submitted for any other future degree or diploma without the permission of Flinders University; and
3. to the best of my knowledge and belief, does not contain any material previously published or written by another person except where due reference is made in the text.

Signature of student: Nicholas Thompson

Print name of student: Nicholas Thompson

Date: 22/10/2024

ACKNOWLEDGEMENTS

I would like to thank my academic supervisor, Dr. Matthew Stephenson, for his invaluable guidance and support throughout my research. His expertise and insights were crucial in developing my understanding of the field of games and AI.

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

INTRODUCTION

Background

Game Playing Agents

The development of artificial intelligence (AI) agents for game-playing is an area of research whose broad goal is to explore the capabilities of AI in complex strategic decision-making and long-term planning[1]. With well-defined parameters and objectives, games provide excellent environments both for benchmarking agent performance and developing novel algorithms[2]. Techniques developed for game-playing agents can be adapted to real-world domains, and real-world environments can be simulated with specific game parameters.

Specialized agents developed for mastery of a single game have seen impressive results against expert level human players. IMB's Deep Blue, a chess playing agent, exemplified this with its' defeat of chess grandmaster and reigning world champion Garry Kaspirov in 1997[3]. Google DeepMind's AlphaGo and its successors have defeated professional Go players, and the most recent iteration, MuZero, has mastered Go, chess, shogi, and Atari[4, 5]. Despite their impressive performance, specialized agents find much of their success in domain or game specific knowledge, hindering their performance in games outside of their training domain[6].

The Ludii System

Started in 2018, the Digital Ludeme Project aspired to create a database of traditional strategy board games. The Ludii game system provides a general board game system that describes its games in terms of elements called ludemes. A ludeme is a “conceptual unit of game related information”[7] and each Ludii game is represented as a 3-tuple of ludemes: Players, Equipment, and Rules.

The conceptual nature of ludemes makes understanding the differences between games and descriptions of their rules straightforward, with the potential to provide insight into game playing agents' decision pathways[8]. With a large repository of games, and a series of game playing agents, the Ludii game system provides the ideal environment for the development of a general game playing agent.

Research Terms and Scope

This research explored the techniques used in the implementation of hyper-agents, with the goal of developing and evaluating general board game playing hyper-agents in the Ludii game system.

Ludeme

A ludeme is a term that describes a single game element, as defined in the Ludii Game Description Language, and can be combined to describe a vast set of different games. Ludemes used in this manor generally fall into three categories: players, equipment, and rules.

Game Concept

A game concept within the Ludii system is a more abstract game element, often arising from distinct combinations of ludemes, allowing for clearer conceptual distinction between games. For example, the way an individual ludeme behaves within the context of a game will vary based on the other ludemes present within the game description.

Agent

Game playing agents are artificial intelligence systems designed to play games. In the context of this research, agents will generally refer to game playing algorithms developed for the Ludii system and used as sub-agents for hyper-agent implementations.

Heuristic

A heuristic is a tool used by agents simplify the evaluation of different pathways through a game tree. Heuristics and heuristic functions define the key considerations for assigning value to game states. An example heuristic might be positive material, which evaluates the position of a player using the sum of their pieces. Using this heuristic, moves that result in more pieces would be assigned a greater value than those that reduce the total piece count.

Hyper-agent

Hyper-agents are game playing agents that employ sub-agents to solve sets of complex problems. This research explores two hyper-agent approaches, portfolio agents and ensemble agents. A portfolio agent employs a selection mechanism to change its sub-agent based on the game it is presented with. Ensemble agents utilise an ensemble of sub-agents, each casting a vote for their preferred move before the ensemble selects the most popular.

Research Significance

General game playing agents are developed with the goal of matching and exceeding human performance across a range of games, without prior knowledge of the game environments. Progress in general game playing is likely to provide insight into the broader challenge of artificial general intelligence[9]. This research will build on existing data within the Ludii database, and follow directly from a study that showed optimal heuristics be reliably predicted using a game's ludeme descriptions[10].

Research Problem and Objectives

Game results sourced from the Ludii database indicate differential performance for agents across the wide variety of games in the Ludii system. If a model could be developed describing the connection between an agent's performance and a game's parameters, optimal agents could be selected to play novel games, with the potential for improved general game playing performance.

Thus, this research had 3 key objectives:

1. To develop a machine learning model that could predict the best agent and heuristic to play a game using the game's parameters
2. To implement hyper-agents in the Ludii game system that utilize this model to optimize general game playing performance
3. To evaluate the hyper-agents' performance in the context of general game playing using the Ludii game system

Thesis Overview

The thesis will begin with a review of the relevant literature and an outline of the research context. Next, the methodology will document techniques used in the development of the implemented hyper-agents and the evaluation of their performance. Experiment results will be presented, followed by a discussion of the implications. Finally, the conclusion will summarise the research findings, and outline suggestions for further research.

LITERATURE REVIEW

General Game Frameworks

General game frameworks aim to provide a universal set of tools that can be used to describe a large range of games in a structured way. Developed in 2005, the Game Description Language (GDL) has been the most prominent over the past few decades for academic research[11]. GDL represents its games as state machines, with rules to specify state transitions, and distinctions made between initial states, goal states, and terminal states. More recent general game frameworks such as the Regular Boardgames language, and the Ludii system, were shown in a 2019 paper to provide more efficient playouts, and simplified game definitions[12].

Game Search Algorithms

Monte Carlo Tree Search

Algorithms developed for general game playing primarily focus on decision making. The most prevelant is the Mone Carlo tree search (MCTS), which uses a tree structure to represent a network of states(nodes) connected by actions(edges)[13]. Originally developed to play Go, MCTS was used to great success in AlphaGo, in combination with deep learning. MCTS explores the game tree by randomly sampling segments, using Monte Carlo simulations, called rollouts, to evaluate game states[14]. Typically implemented with the Upper confidence bound applied to Trees (UCT) to optimise the exploitation-exploration trade-off, the probabilistic nature of MCTS makes it well suited to games with large search spaces and stochastic components[15].

Min-Max Search

The min-max search algorithm is an alternative approach to searching a game tree, applicable to alternating move, adversarial games with two players[16]. The algorithm evaluates and assigns values to leaf nodes at the terminating depth of the search. Defining a minimising and maximising player, each node in the tree is assigned a value from one of its child nodes, determined by active player – a minimising player selects the node with the lowest value, while the maximising player selects the highest. The advantage of the min-max search algorithm is the full coverage of the game, though it suffers from an exponential growth in search space as depth increases and is limited by the effectiveness of its position evaluation metrics.

The alpha-beta pruning algorithm optimises the min-max search by pruning branches that are known not to be optimal decisions for the parent node’s active player, reducing the search space and allowing for deeper, more efficient searches of the game tree. Research into applications of min-max pruning to games with more than two players found that the explored pruning implementation did not effectively reduce the search space[17].

Hyper-Agents

Portfolio

Portfolio agents have been implemented for video game playing with some compelling results. In 2017 a portfolio agent was developed using the Angry Birds AI competition following the observation that agents from previous years had varied performances across different levels[18]. The highest score from each agent at each level was recorded after a series of level runs, along with the key features of each level, to provide the hyper-agent with a set of training data. The portfolio approach allowed the hyper-agent to outperform each of the sub-agents overall, across 80 levels.

An exploration of portfolio search optimization for general strategy game playing found that standard portfolio approaches to Stratega, a general strategy games framework, could be improved[19]. The analysis reviewed portfolio improvements to decision-making in real-time scenarios in which the portfolio search modelled increasingly optimized moves for both agent and opponent units across the 6 potential actions each unit could take. This algorithm is noted as a common implementation for real-time strategy unit micromanagement[20, 21].

An evolutionary approach was also explored, in which scripts (unit/state action pairs) were periodically mutated, with the “fittest” scripts propagating[22]. Notably, the best performing algorithm was observed to prioritize the actions of support/specialist units, rather than direct attacks. While shown to have measurable improvements in real-time and turn based strategy games with incomplete information, these optimizations may not provide significant improvement to portfolio searches in turn-based board games with complete information.

Ensemble

Ensemble agents employ a slightly different approach to utilizing sub-agents. Rather than selecting the best agent for a particular scenario, ensemble agents poll their sub-agents and proceed with the most popular action. A 2019 study looked at using an ensemble decision system for general video game playing as part of the General Video Game AI competition[23]. The hyper-agent allowed each sub-agent to voice their opinions before the arbitrator made the final decision. Interestingly, this study observed that ensemble agents performed best on certain levels when a specific sub-agent was present, though that same sub-agent was not able to complete the level on its own. The games in which certain sub-agents excel saw those sub-agents consistently outperform the ensemble agent, even when they were also part of the ensemble. This is likely due to the nature of the ensemble agent’s polling system, and a portfolio approach may perform better in this scenario.

A 2022 study describes a weighted ensemble agent created to play Werewolf[24]. Werewolf is a social communication game in which players are assigned the role of a villager or a werewolf. Each group must identify and eliminate the other - a game of incomplete information, deception, and deduction. This study identified an issue with standard ensemble agents that reduces the effective contribution of high performing sub-agents because of equal weighted polling. To address this, each sub-agent was assigned a weight based on its performance during training, such that the importance of its vote when polled would be adjusted. The ensemble agent performed very well in games with 15 players when compared to games with 5 players[24], however the results from this study make it difficult to draw conclusions about the effectiveness of a weighted ensemble approach, since the top sub-agent outperformed the ensemble agent in both 5 and 15 player games. A better comparison might have been made with a non-weighted ensemble agent to assess the impact that the weighting had on the agent's performance.

Research Context

In 2021 a study was conducted to investigate general game heuristic prediction based on ludeme descriptions[10]. The study documented the performance of several heuristics across 695 games in the Ludii system. The results indicated that a game's ludemes could be used to predict the performance of different heuristics, and suggested further research could be conducted into the development of a portfolio agent[10]. Thus, the aim of this research was to develop a general game playing portfolio agent, leveraging a novel game's ludemes and concepts to predict the optimal agent and heuristic for playing it. Building on this, an ensemble agent would be developed as an alternative hyper-agent approach to general game playing, and a point of comparison for the portfolio agent. Finally, a combination of the two hyper-agents would be implemented, using the portfolio agent's model to modify the weight of votes for each sub-agent in the ensemble.

METHODOLOGY

Portfolio Models

Training Data

Data was sourced from the Ludii database, available at https://ludii.games/download.php. Data for agent and heuristic win rates across a set of games in the Ludii game system was used as targets for the models, while features were comprised of the ludemes and concepts that define each game. The full data set used is available at the repository provided in Appendix A. Game ludemes were represented as binary values, indicating whether a ludeme was present in the corresponding game. Table 1 provides an example of the ludeme data structure.

Table 1 - Data structure for ludeme game descriptions

Game Name	Ludeme 1	Ludeme 2	Ludeme 3	…	Ludeme n
Game 1	1	0	0	…	0
Game 2	1	1	0	…	0
Game 3	0	1	0	…	1
…	…	…	…	…	…
Game n	1	0	1	…	1

Game concepts were structured similarly, though the concepts included a mix of data types. Table 2 provides an example of the game concepts data structure.

Table 2 - Data structure for game concepts

Game Name	Concept 1	Concept 2	Concept 3	…	Concept n
Game 1	1	8	-2	…	0
Game 2	1	2	1	…	1
Game 3	0	4	4	…	1
…	…	…	…	…	…
Game n	1	3	3	…	1

Win rates for agents and heuristics were represented as a score between 0 and 100. Table X provides an example of the data structure.

Table 3 – Data structure for Agent/Heuristic win rate results

Game Name	Agent/Heuristic 1	Agent/Heuristic 2	Agent/Heuristic 3	…	Agent/Heuristic n
Game 1	0	100	54	…	12
Game 2	12	31	21	…	36
…	…	…	…	…	…
Game n	95	78	54		32

An analysis of the agent win rate data indicated that several agents had win rates identical to the random agent for most games in the set. Agents whose win rates were identical to the random agent were removed from the final training set. Table 4 lists the final set of agents used in the training of the models, with a brief description and an indication of whether that agent requires heuristics. The random agent was removed from the dataset when exporting the final models for hyper-agent implementation.

Table 4 - Ludii agent descriptions and heuristic requirements

Agent Name	Description	Heuristics
Alpha-Beta	Alpha-beta pruning algorithm, an optimisation of the min-max search[16].	Yes
MAST	Move-Average Sampling Technique, a playout strategy for MCTS[25].	No
MC-GRAVE	Application of the Rapid Action Value Estimation heuristic for MCTS[26].	No
Progressive History	Application of progressive bias using the history heuristic for MCTS[27].	No
UCT	Standard application of Upper Confidence Bounds applied to Trees with MCTS[15].	No
Random	An agent that selects moves at random.	No

Model Training

Regression and classification models were trained in Python, utilizing scikit-learn machine learning libraries. Classification models used the agent or heuristic with the highest win rate as the target for each game, while regression models predicted a win rate for each target, selecting the target with the highest predicted win rate as the best candidate for the corresponding set of features.

Model Validation

To ensure trained models were reliable and not overfitted, an initial shuffle was performed, followed by 10-fold-cross-validation, which split the data into 10 subsets, using each as a test set once, with remaining subsets used as the training set.

The performance of each model was evaluated by calculating the average difference between the win rates of the predicted best target and the actual best target, referred to here as regret.

$$Regret = Actual_{WR} - Predicted_{WR}$$

The regret metric was selected over a measure of accuracy to better reflect the actual cost of making a suboptimal prediction within the game system, since the selected suboptimal agent or heuristic will still produce a win rate across a set of games.

In addition to the candidate models, naïve regressors and classifiers were also trained and evaluated, providing a baseline point of comparison for model performance. These naïve models used the highest average method, predicting the agent or heuristic with the highest average win rate. The model that produced the lowest average regret for each set of targets (agents, heuristics) was selected for the development of Hyper-Agents within the Ludii system.

Hyper-Agent Implementation

Ludii AI

Ludii, written in Java and open source (https://github.com/Ludeme/Ludii), provides an abstract AI class that can be extended to create a game playing agent. Each hyper-agent extended this class, implementing the initAI and selectAction methods required to play games. The initAI method is called on each participating agent when a new game is initialised, providing the agents with the game object, and their player id. The selectAction method is called each time an agent is required to make a move, providing the moving agent with the game context and the allowed thinking time before a move must be returned. Additionally, the aiVisualisationData method was implemented, allowing visualization of an agent's search process within the Ludii UI.

Portfolio Models

The trained portfolio models were exported from Python to the Predictive Model Markup Language (PMML) format using a pipeline provided by the sklearn2pmml Python library https://github.com/jpmml/sklearn2pmml. This allowed the models to be imported directly into Java utilizing an XML parser, and then evaluated using the JPMML Evaluator, also provided by the JPMML team https://github.com/jpmml/jpmml-evaluator.

Portfolio Agent

The key function of the portfolio agent was the ability to switch its sub-agent based on the parameters (ludemes, concepts) of the game it was playing. This logic was implemented within the initAI method, extracting the game concepts and ludemes from the game object and querying the portfolio models for the best predicted agent and heuristic. Using these predictions, the portfolio agent initialised its selected sub-agent for the game. Once the portfolio agent was initialised, it passed all calls from the Ludii system on to its sub-agent, which then played out the game as normal. Listing 1 outlines the initialisation algorithm.

Listing 1 - Portfilio agent initialistaion algorithm

```
FUCNTION initAI
        Retrieve ludemes and concepts from the game object
        Format the ludemes and concepts to match model input format
        Query the heuristic model, storing best heuristic
        Query the agent model
        Initialize sub-agent variable
        Initialize best win rate variable to 0
        FOR each sub-agent in predictions
                IF predicted sub-agent win rate is greater than current best win rate
                        Update sub-agent variable with this sub-agent
                        Update best win rate variable with this win rate
                ENDIF
        ENDFOR
        Create new instance of sub-agent (with best predicted heuristic if applicable)
        CALL initAI on sub-agent
```

Ensemble Agent

The ensemble agent required the introduction of multiple sub-agents playing the same game, selecting a move to play each turn. The ensemble system utilized an ArrayList to store each of its sub-agents, instantiated when the initAI method was called by the Ludii system.

Listing 2 - Standard ensemble agent intialisation algorithm

```
FUCNTION initAI
        Retrieve ludemes and concepts from the game object
        Format the ludemes and concepts to match model input format
        Query the heuristic model, storing best heuristic
        Retrieve list of possible agents
        FOR each agent in agent list
                Create new instance of sub-agent (with best predicted heuristic if applicable)
                IF sub-agent supports game
                        Add sub-agent to ensemble sub-agent list
                        CALL initAI on sub-agent
                ENDIF
        ENDFOR
```

The ensemble agent's move selection algorithm was implemented by iterating through its ArrayList of sub-agents and calling the selectAction method on each. The total thinking time was split among the sub-agents, and vote counts for each selected move were tracked using a HashMap with moves as keys and votes as values. Once all agents had voted, the move with the highest vote count was played. Listing 3 outlines the move selection algorithm employed by the standard ensemble agent.

Listing 3 - Standard ensemble move selection algorithm

```
FUCNTION selectAction
        Initialise move map //moves as keys, votes a values
        Initialise best move
        Initialise highest vote count to 0
        FOR each agent in the ensemble
                CALL selectAction on agent
                Update move map with the selected move, incrementing the vote count
                IF selected move has a vote count higher than the current best move
                        Update the best move
                        Update the highest vote count
                ENDIF
        ENDFOR
        RETURN best move
```

Weighted Ensemble Agent

The ensemble agent was then extended with the introduction of weighted votes for each of its sub-agents. In the same manor as the portfolio agent, the ensemble agent would query the portfolio models prior to initialising its sub-agents, assigning each of them a weighting based on their predicted win rates. Listing 4 details the initAI implementation for the weighted ensemble agent.

Listing 4 - Weighted ensemble initialisation algorithm

```
FUCNTION initAI
    Retrieve ludemes and concepts from the game object
    Format the ludemes and concepts to match model input format
    Query the heuristic model, storing best heuristic
    Query the agent model
    Initialize total vote weight to 0
    FOR each agent in predictions
        Create new instance of sub-agent (with best predicted heuristic if applicable)
        IF sub-agent supports game
            Add sub-agent to ensemble sub-agents list
            Assign vote weight to sub-agent
            CALL initAI on sub-agent
            Update total vote weight
        ENDIF
    ENDFOR
```

The selectAction algorithm was modified, adding each sub-agent's vote weight to the total weight value for the move it selected. The ensemble agent would then return the move with the highest total vote weight. Listing 5 details the move selection algorithm for the weighted ensemble agent.

Listing 5 - Weighted ensemble move selection algorithm

```
FUCNTION selectAction
    Initialise move map //moves as keys, votes a values
    Initialise best move
    Initialise highest vote weight to 0
    FOR each agent in the ensemble
        CALL selectAction on agent
        Update move map with the selected move, adding the agent vote wieght
        IF selected move has a vote weight higher than the current best move
            Update the best move
            Update the highest vote weight
        ENDIF
    ENDFOR
    RETURN best move
```

Agent Evaluation

Experiment Parameters

An experiment was developed to evaluate the performance of the hyper-agents as general game playing agents. The experiment required each agent to play a representative set of 30 games, playing each game 50 times. The agent being evaluated always occupied the first player position, while any additional players required for a game were populated with standard UCT agents. Each agent had 1 second of thinking time per move, and each game had a move limit of 1000 per agent. The result of each game was recorded, assigning the evaluated agent a score between 0 and 1. 0 indicated a loss, 1 indicated a win, and a draw was indicated by 1/(number of players). The evaluated agents included the 5 sub-agents, the portfolio agent, the standard ensemble agent, the weighted ensemble agent and a random agent. Additionally, parallel implementations of the ensemble agents (with each sub-agent provided the full thinking time) were also evaluated.

Game Selection

The representative set of games for the experiment was determined using an algorithm that iteratively selected the game with the highest minimum distance from each of the previously selected games. Of the 1109 games available in the Ludii database, 460 were present in the agent model's training dataset. The experiment sample was selected from the remaining 649. Listing 6 outlines the game selection process.

Listing 6 - Representative game selection algorithm using maximum minimum distance

```
FUCNTION selectGames
    Initialise game list
    Initialise selected game matrix
    Retrieve distance matrix
    WHILE the length of game list is less than 30
        Initialise min distance to 0
        Initialise selected index
        FOR each column in the selected game matrix
            Get the lowest value in the column
            IF the value is larger than current min distance
                Update min distance
                Update selected index
            ENDIF
        ENDFOR
        Append the row selected index from the distance matrix to the selected game matrix
        Append the game name to the game list
    ENDWHILE
    RETURN game list
```

Distance between two games was calculated using the angle between the two vectors defined by the set of ludemes and concepts that describe each game (Cosine distance)[28]. Cosine distance was chosen over Euclidian distance due to the high dimensionality of the vectors. Prior to the pairwise distance calculations, a bi-symmetric log transform was performed on the game concepts data as described here[28]. This operation was executed with the intention minimising the impact of concepts with large continuous ranges while preserving the relative significance of concepts with binary values. Finally, the concepts data was scaled. The final set of selected games is listed here:

Starchess, DisPath, Ludus Anglicorum, Trax, Quixo, Banqi, Terhuchu, Katro, Mini Wars, Surakarta, Mylna, Lines of Action, Ploy, Triad, Amazons, La Liebre Perseguida, Pylos, Chomp, Chameleon, Tawlbwrdd, Vela, Aralzaa, Chuka, Hops-a-Daisy Generalized, Mehen, Janggi, Patok, Toki, Omega and Gauntlet.

T-distance stochastic neighbour embedding of the games is represented in figure 1, illustrating the coverage of the selected games relative to the full set of games in the Ludii database. Solid blue points represent games available for selection, faded blue points indicate games in the training set (excluded from selection), and selected games are represented with pink crosses.

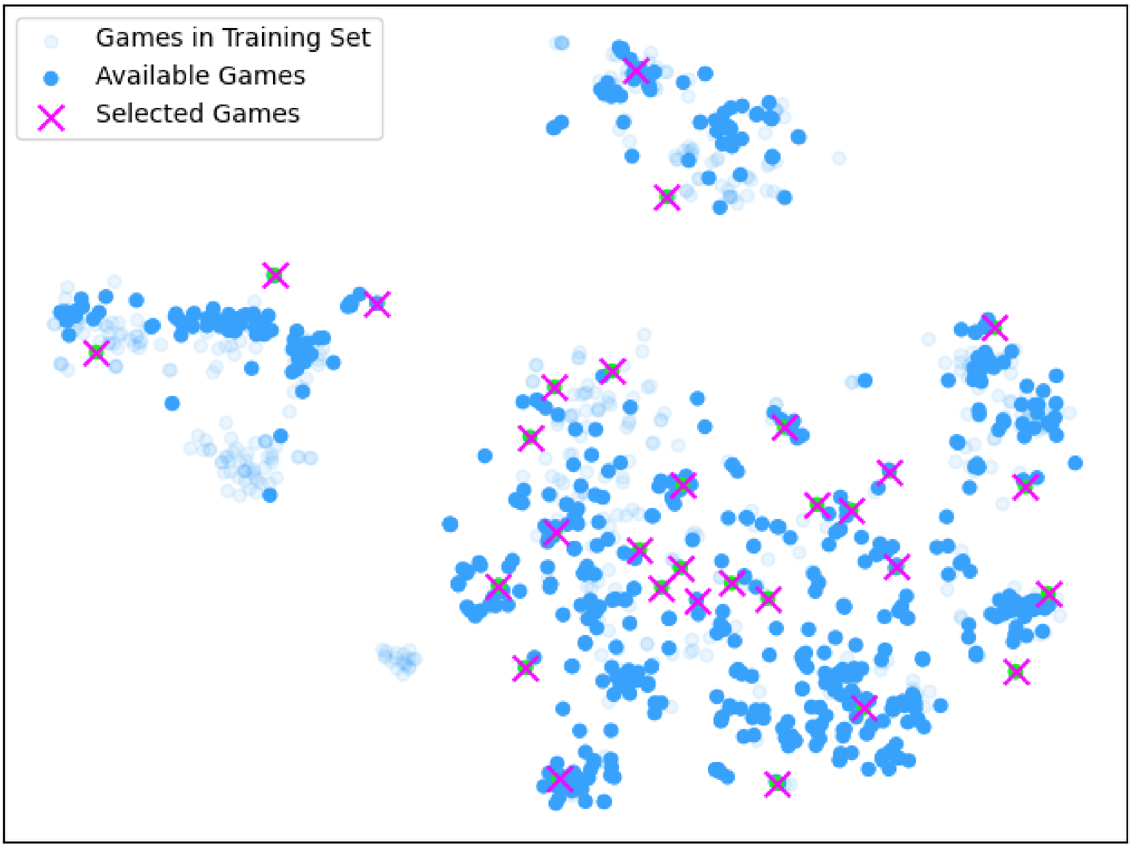

Figure 1 - 2-dimensional t-SNE, selected game coverage relative to the full game set

Experiment Hardware

The experiment was run using Docker containers, on 3 memory optimized Digital Ocean cloud servers with the following specifications:

CPU: 2x Intel(R) Xeon(R) Gold 5318Y CPU @ 2.10GHz

Memory: 16GB

Each server ran 2 docker containers for a total of 6, results were recorded in a MySQL database.

Experiment Analysis

Average win rates for each agent across the set of 1500 game playouts were used as a measure of general game playing performance. A two-tailed student’s t-test was used to evaluate the statistical significance of the experiment results.

RESULTS AND DISCUSSION

Portfolio Models

The model that produced the lowest regret for both the agents and heuristics data was the Random Forest Regressor, achieving a significant improvement over the naive approaches (Dummy Classifier, Dummy Regressor).

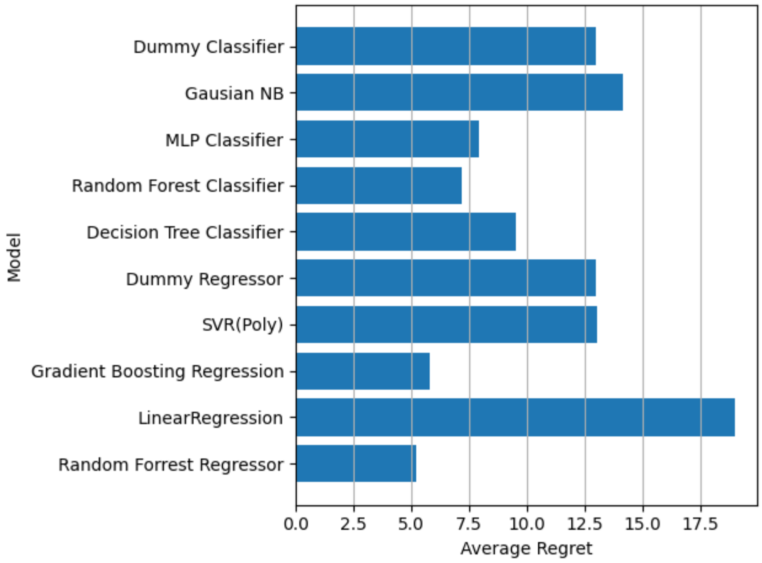

Figure 2 – Average regret for models predicting best agents based on game ludemes and concepts

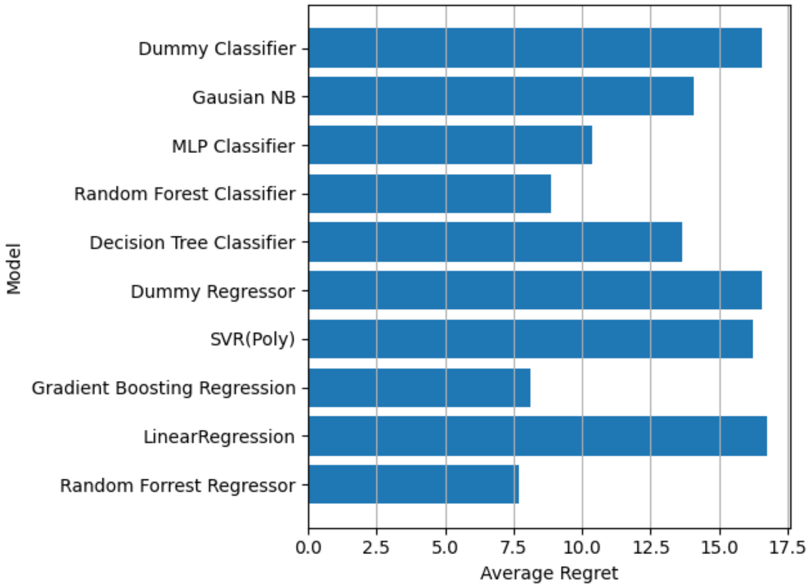

Figure 3 - Average regret for models predicting best heursitics based on game ludemes and concepts

Experiment Results

Figure 4 shows the average win rate of each agent across the 1500 game playouts. Parallel ensemble results are denoted with a “(P)”.

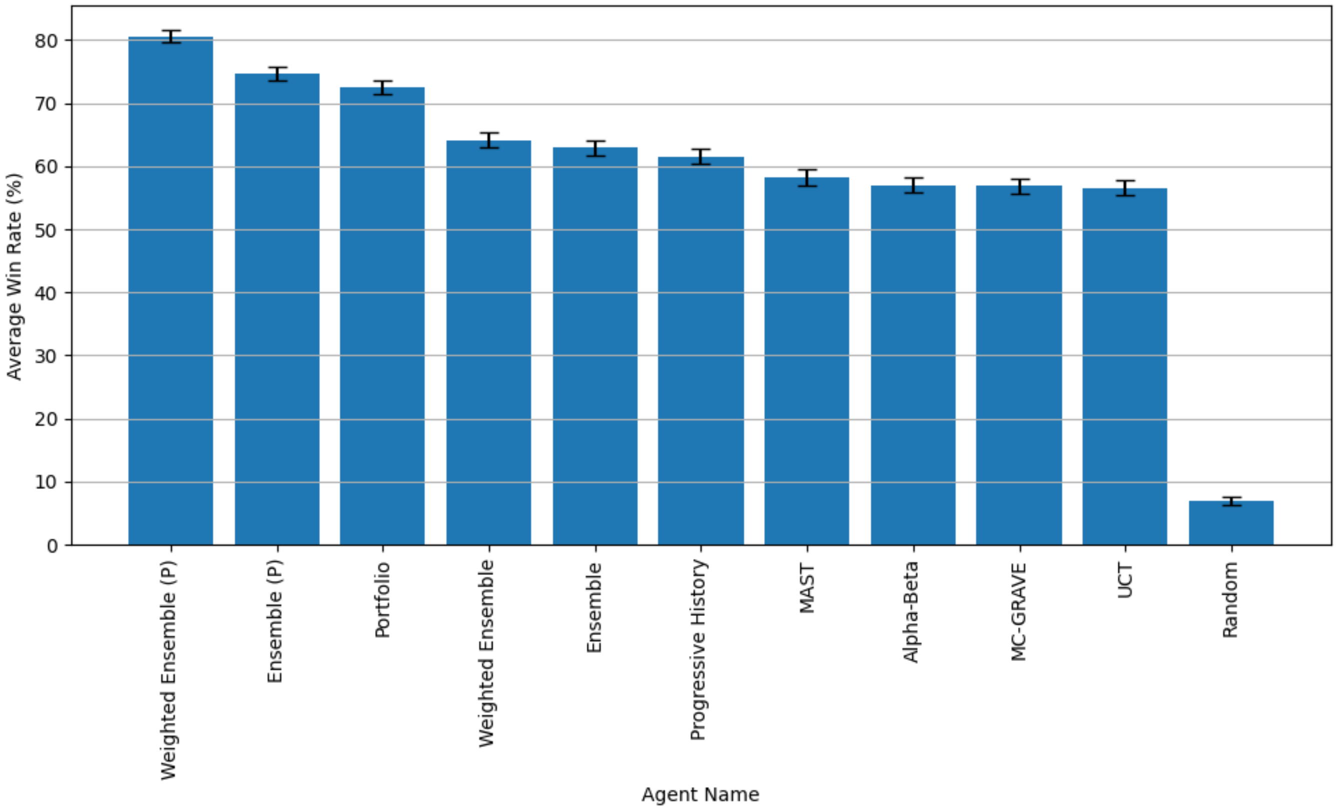

Figure 4 – Average agent win rates for 1500 game playouts (50 iterations of 30 games)

As expected, the hyper-agents outperformed each of the sub-agents on average across the set of games. The best performing agent in a non-parallelized environment was the Portfolio agent, while the introduction of parallelization showed the weighted ensemble achieved the best win rate. To determine whether these results are of statistical significance, a two-tailed student’s t-test was performed. Table 5 contains a T-value for each agent pair’s game results, indicating the magnitude and direction of the difference between the average win rates, while accounting for the variance. The accompanying P-value indicates the probability of observing the T-value by chance. For a statistically significant result, the P-value should be less than 0.05.

Table 5 - Student T-Test

	P > 0.05						P < 0.05					
	Agent	Alpha-Beta	MAST	MC-GRAVE	Progressive History	Random	UCT	Portfolio	Ensemble	Weighted Ensemble	Parallel Ensemble	Parallel Weighted Ensemble
Ludii Agents	Alpha-Beta	N/A	T: -0.727 P: 0.467	T: 0.077 P: 0.938	T: -2.657 P: 0.008	T: 36.428 P: 0.000	T: 0.271 P: 0.786	T: -9.450 P: 0.000	T: -3.455 P: 0.001	T: -4.199 P: 0.000	T: -10.950 P: 0.000	T: -15.165 P: 0.000
	MAST	T: 0.727 P: 0.467	N/A	T: 0.806 P: 0.420	T: -1.932 P: 0.053	T: 37.647 P: 0.000	T: 1.002 P: 0.316	T: -8.738 P: 0.000	T: -2.737 P: 0.006	T: -3.481 P: 0.001	T: -10.238 P: 0.000	T: -14.453 P: 0.000
	MC-GRAVE	T: -0.077 P: 0.938	T: -0.806 P: 0.420	N/A	T: -2.741 P: 0.006	T: 36.442 P: 0.000	T: 0.194 P: 0.846	T: -9.551 P: 0.000	T: -3.540 P: 0.000	T: -4.286 P: 0.000	T: -11.056 P: 0.000	T: -15.288 P: 0.000
	Progressive History	T: 2.657 P: 0.008	T: 1.932 P: 0.053	T: 2.741 P: 0.006	N/A	T: 40.776 P: 0.000	T: 2.943 P: 0.003	T: -6.820 P: 0.000	T: -0.824 P: 0.410	T: -1.567 P: 0.117	T: -8.315 P: 0.000	T: -12.510 P: 0.000
	Random	T: -36.428 P: 0.000	T: -37.647 P: 0.000	T: -36.442 P: 0.000	T: -40.776 P: 0.000	N/A	T: -36.289 P: 0.000	T: -51.955 P: 0.000	T: -41.490 P: 0.000	T: -42.760 P: 0.000	T: -54.905 P: 0.000	T: -64.247 P: 0.000
	UCT	T: -0.271 P: 0.786	T: -1.002 P: 0.316	T: -0.194 P: 0.846	T: -2.943 P: 0.003	T: 36.289 P: 0.000	N/A	T: -9.772 P: 0.000	T: -3.741 P: 0.000	T: -4.489 P: 0.000	T: -11.283 P: 0.000	T: -15.533 P: 0.000
Hyper-Agents	Portfolio	T: 9.450 P: 0.000	T: 8.738 P: 0.000	T: 9.551 P: 0.000	T: 6.820 P: 0.000	T: 51.955 P: 0.000	T: 9.772 P: 0.000	N/A	T: 5.929 P: 0.000	T: 5.197 P: 0.000	T: -1.464 P: 0.143	T: -5.533 P: 0.000
	Ensemble	T: 3.455 P: 0.001	T: 2.737 P: 0.006	T: 3.540 P: 0.000	T: 0.824 P: 0.410	T: 41.490 P: 0.000	T: 3.741 P: 0.000	T: -5.929 P: 0.000	N/A	T: -0.735 P: 0.463	T: -7.404 P: 0.000	T: -11.532 P: 0.000
	Weighted Ensemble	T: 4.199 P: 0.000	T: 3.481 P: 0.001	T: 4.286 P: 0.000	T: 1.567 P: 0.117	T: 42.760 P: 0.000	T: 4.489 P: 0.000	T: -5.197 P: 0.000	T: 0.735 P: 0.463	N/A	T: -6.672 P: 0.000	T: -10.795 P: 0.000
	Parallel Ensemble	T: 10.950 P: 0.000	T: 10.238 P: 0.000	T: 11.056 P: 0.000	T: 8.315 P: 0.000	T: 54.905 P: 0.000	T: 11.283 P: 0.000	T: 1.464 P: 0.143	T: 7.404 P: 0.000	T: 6.672 P: 0.000	N/A	T: -4.064 P: 0.000
	Parallel Weighted Ensemble	T: 15.165 P: 0.000	T: 14.453 P: 0.000	T: 15.288 P: 0.000	T: 12.510 P: 0.000	T: 64.247 P: 0.000	T: 15.533 P: 0.000	T: 5.533 P: 0.000	T: 11.532 P: 0.000	T: 10.795 P: 0.000	T: 4.064 P: 0.000	N/A

Discussion

The results indicate that hyper-agent approaches do yield statistically significant improvements as general game playing agents over the existing agents implemented in the Ludii game system. The portfolio agent achieved the best performance of the non-parallelized agents, with a statistically significant improvement over each of the sub-agents and a minimum average win-rate difference of +6%. While both parallel Ensemble implementations performed better than the non-parallelized agents, the difference between the standard parallel ensemble agent and the portfolio agent was not statistically significant. The parallel weighted ensemble agent performed best overall, with a statistically significant improvement over every other agent in the experiment.

The Portfolio Agent's performance serves as further validation of the agent and heuristic models that were trained on the Ludii data and used to predict the best agent and heuristic for each game. The agent with the highest average win rate within the Ludii data set was Alpha-Beta, while the results from this experiment saw Progressive History perform better, with a statistically significant difference. Importantly, the game set for this experiment contained two games that Alpha-Beta doesn't support, resulting in a win rate of 0 for the 50 playouts of each of these. Furthermore, many games in the Ludii system contain optimal heuristics in their metadata, allowing Alpha-Beta to take advantage, and likely skewing its win rate. To ensure the results in this experiment were applicable to general game playing, the agents were not permitted to use this knowledge, instead relying on the heuristics predicted by the portfolio models. This may also suggest that Alpha-Beta's win rate was overrepresented in the training data, however the agent model was still reliable enough to produce statistically significant improvement for the Portfolio agent.

The non-parallel ensemble agents underperformed expectations, with no statistically significant performance differentiation between their results, nor the best performing Ludii agent (Progressive History). This is likely, in part, due to the limited thinking time given to each agent within the experiment. With a thinking time of 1 second per turn being split among the 5 agents within the ensemble, each agent would receive an average of 0.2 seconds to select a move. The same experiment run with increased thinking time per agent would likely yield improved performance for these ensemble implementations, though it is important to note that with each iteration of a move search, the search space grows exponentially (depending on the complexity of the game), since each move under consideration will lead to a new set of possible moves to consider. Thus, increases in thinking time will likely lead to diminishing performance improvements, and sufficiently large thinking time increases will see the non-parallelized ensemble agents approach the performance their parallelized counterparts.

CONCLUSION AND FUTURE WORKS

Key Findings

The development of a portfolio model to predict optimal game playing agents using game concepts and ludemes was shown to produce an improvement in average win rate across the set of games in the Ludii agent results data. This model was confirmed empirically with the implementation and evaluation of a portfolio agent within the Ludii system across a representative set of 30 games, which produced a statistically significant improvement in performance over each of its sub-agents. Ensemble agents implemented in the Ludii system also showed an improvement in average win rate over each of their sub-agents, with the parallel weighted ensemble performing best overall.

Thus, the research showed that hyper-agents are not only a viable approach to general game playing but produced statistically significant improvements over existing agent implementations within the Ludii system.

Future Works

A key limitation of this research lies in the limited set of games that were selected for hyper-agent evaluations. Although the results showed statistically significant differences in agent win rates, future research might include a more robust set of evaluation games.

Non-parallel implementations of the ensemble and weighted ensemble underperformed expectations, with no statistically significant difference between the two, nor the closest sub-agent. Future research might explore how the win rates of these hyper-agents scale as the allowed thinking time per move changes.

REFERENCES

1. Anderson, D., *General video game playing using ensemble decision systems*. 2020.
2. Mathias, F. and S. Andreas, *Games and AI: Paths, Challenges, Critique.* Eludamos, 2020. **10**(1).
3. Campbell, M., A.J. Hoane, and F.-h. Hsu, *Deep Blue.* Artificial Intelligence, 2002. **134**(1): p. 57-83.
4. Schrittwieser, J., et al., *Mastering Atari, Go, chess and shogi by planning with a learned model.* Nature, 2020. **588**(7839): p. 604-609.
5. Hunter, C. and B.E. Bowen, *We'll never have a model of an AI major-general: Artificial Intelligence, command decisions, and kitsch visions of war.* Journal of Strategic Studies, 2024. **47**(1): p. 116-146.
6. Li, J. and G. Kendall, *A hyper-heuristic methodology to generate adaptive strategies for games.* IEEE transactions on computational intelligence and AI in games., 2015. **PP**(99).
7. Piette, E., et al., *Ludii--The Ludemic General Game System.* arXiv preprint arXiv:1905.05013, 2019.
8. Piette, É., et al., *General Board Game Concepts.* arXiv (Cornell University), 2021.
9. Jebari, K. and J. Lundborg, *Artificial superintelligence and its limits: why AlphaZero cannot become a general agent.* AI & society, 2021. **36**(3): p. 807-815.
10. Stephenson, M., et al. *General Game Heuristic Prediction Based on Ludeme Descriptions*. 2021. Piscataway: IEEE.
11. Thielscher, M. *A general game description language for incomplete information games*. in *Proceedings of the AAAI conference on artificial intelligence*. 2010.
12. É, P., et al. *An Empirical Evaluation of Two General Game Systems: Ludii and RBG*. in *2019 IEEE Conference on Games (CoG)*. 2019.
13. Świechowski, M., et al., *Monte Carlo Tree Search: a review of recent modifications and applications.* Artificial Intelligence Review, 2023. **56**(3): p. 2497-2562.
14. Soemers, D.J.N.J., et al. *Optimised Playout Implementations for the Ludii General Game System*. in *Advances in Computer Games*. 2022. Cham: Springer International Publishing.
15. James, S., G. Konidaris, and B. Rosman, *An Analysis of Monte Carlo Tree Search.* Proceedings of the AAAI Conference on Artificial Intelligence, 2017. **31**(1).
16. Knuth, D.E. and R.W. Moore, *An analysis of alpha-beta pruning.* Artificial Intelligence, 1975. **6**(4): p. 293-326.
17. Korf, R.E., *Multi-player alpha-beta pruning.* Artificial Intelligence, 1991. **48**(1): p. 99-111.
18. Stephenson, M. and J. Renz, *Creating a hyper-agent for solving angry birds levels*, in *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. 2017, AAAI Press: Little Cottonwood Canyon, Utah, USA. p. Article 34.
19. Dockhorn, A., et al. *Portfolio Search and Optimization for General Strategy Game-Playing*. in *2021 IEEE Congress on Evolutionary Computation (CEC)*. 2021.
20. Churchill, D. and M. Buro. *Portfolio greedy search and simulation for large-scale combat in starcraft*. in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. 2013.
21. Moraes, R., J. Mariño, and L. Lelis, *Nested-Greedy Search for Adversarial Real-Time Games.* Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2018. **14**(1): p. 67-73.
22. Sironi, C.F., J. Liu, and M.H.M. Winands, *Self-Adaptive Monte Carlo Tree Search in General Game Playing.* IEEE Transactions on Games, 2020. **12**(2): p. 132-144.
23. Anderson, D., et al. *Ensemble Decision Systems for General Video Game Playing*. in *2019 IEEE Conference on Games (CoG)*. 2019.
24. Khan, M. and C. Aranha, *A Novel Weighted Ensemble Learning Based Agent for the Werewolf Game.* arXiv (Cornell University), 2022.
25. Sironi, C.F., T. Cazenave, and M.H. Winands. *Enhancing playout policy adaptation for general game playing*. in *Monte Carlo Search International Workshop*. 2020. Springer.

26. Cazenave, T. *Generalized rapid action value estimation*. in *24th International conference on artificial intelligence*. 2015.
27. Nijssen, J.A. and M.H. Winands, *Playout search for Monte-Carlo tree search in multi-player games*, in *Advances in Computer Games*. 2011, Springer. p. 72-83.
28. Stephenson, M., et al. *Measuring Board Game Distance*. in *Computers and Games*. 2023. Cham: Springer Nature Switzerland.

APPENDICES

Appendix A. Source Code Repository

The source code used for this research is available at**:** https://github.com/thom11345/HyperAgents.

Code used for model development, results analysis and visualisation are in the Models directory. Input data is in Models/data.

The source code for the hyper-agent implementations is in the AI directory, with the compiled jar located in AI/build.

Experiment source code, database schema, and results data are in the Experiments folder.