Improved algorithm selection in the hyper-agent approach to General Video Game Playing

Quintana Pelayo, Guillermo; Clappers, Lisa; Oberlies-Rodrigues, Jory; Rao, Chinmay; Romita, Alessio & Scholer, Tom
AI & DSDM Master Students
Data Science and Knowledge Engineering
Maastricht University
g.quintanapelayo, l.clappers, j.oberlies-rodrigues, c.rao, a.romita, t.tomfernandgeorges @student.maastrichtuniversity.nl

January 22nd 2020

Abstract

General Video Game Playing (GVGP) [1] is a subfield in Artificial Intelligence (AI) research that aims to create proactive programs capable of playing many different types of video games successfully. In this research project, rather than building a single general video-game playing agent, we attempt to further the research along the lines of the "hyper-agent" approach. This involves having an algorithm selecting an agent from a collection of agents, such that the most suitable to play a given specific video game does play that game. More specifically, we perform game categorization to classify games into different computational genres based on certain useful features. Our objective is to find out which game features are relevant with respect to agents' performance and to analyze the quality of game categorization obtained using these features.

This project is part of the Master Research Project from the Faculty of Data Science and Knowledge Engineering Master program of the Maastricht University, and with it, we aim to create a hyper-agent for GVGP for single player video games.

Keywords: *hyper-agent, GVGP, GVGAI, game classification*

1 Introduction

General Video Game Playing involves creating a proactive program called an *agent* that can successfully play many different video games, in particular games that are unknown to the agent. This is in contrast to designing an algorithm to work optimally on a single game known to the designer beforehand.

To facilitate development in GVGP, the General Video Game AI (GVGAI) framework [2] was introduced to serve as a general framework for game-based testing of artificial intelligence methods. The video games included in the GVGAI framework are classic 2D arcade-style games since these are relatively simple, interactive enough and require real-time response. In addition to the framework, the GVGAI competition [3] aims to benchmark the general game-playing AI algorithms on a set of games unknown to the contenders.

The objective of GVGP is the creation of a generally intelligent game-playing agent. Solving the GVGP problem would take the AI research community a step closer to the truly general solver, an algorithm that could perform a wide spectrum of real-world tasks. The GVGAI competition consists of different tracks with a variety of objectives (two players learning, planing, etc.). The relevance of our work lies on improving the existing methods in GVGP, specifically in the Single-player Planning track of GVGAI. This is due to the complexity of the games that involve several players and also due to time constraints for the project itself.

Several efforts have been made to create successful agents for this task. Common successful search strategies that have been applied to GVGP are Monte-Carlo Tree Search (MCTS) [4, 5, 6], or the N-Tuple Bandit Evolutionary Algorithm (NTBEA) [7].

However, the objective of our work moves away from these past attempts in the sense that we aim to create a hyper-agent capable of selecting the most suitable agent among all the already existing ones for any game that can be presented to it. This means that we did not develop any new agents like the ones stated in section 3. Instead, we collected already existing agents submitted in the past GVGAI competitions (on the single-player planning track) and worked with them to create our selector using the hyper-agent approach. Thus, we aim to know if the performance of a General Video Game Playing hyper-agent can be improved by improving the game classification used by the hyper-agent.

Furthermore, as will be further explained later, we explored the influence of adding new game features (on which the classification is based) on the performance. To accomplish this task, we used two specific clustering algorithms: K-means and Fuzzy-ART.

The paper is structured as follows. First, a description of general video game playing and the GVGAI framework as well as some principles of hyper-heuristics and algorithm selection are given. Then, we describe our methodology: which controllers (agents) were used for the hyper-agent, how we extracted features to use and collected data on controller performance, and we describe the classifiers used to predict the best controllers. Following this we explain the results of using these classifiers and finally we give the conclusions derived from the conducted experiments and describe further work and research that could be done.

2 Motivation

The defining feature of hyper-heuristics is that they operate on a search space of heuristics rather than directly on a search space of problem solutions. This property of hyper-heuristics provides the potential for increasing the level of generality of search methodologies [8].

Hyper-agent methods are expected to have better performance than single agents. Nevertheless, it is possible that we do not obtain a better performance than the best existing agent. Improving the classification requires using higher quality features and the algorithms that use these features. As it will be mentioned in section 3, a similar approach to the one that we are proposing here has already been tested and, even though the results were not good as expected, they significantly improved the overall scores. That significant improved performance is one of the main reasons why we still bid for this approach, because we see a lot of potential in applying this methodology to General Video Game Playing.

3 Previous work

In order to better understand the background surrounding our research it is important to look at some of the approaches that have already been tested, both as a single agent and as a higher level view such as the one that this projects discusses.

General Video Game Playing has been deeply researched for several years now and it has too many tracks to be discussed here. Since this history lies out of the scope of our project, the previous work discussed here focuses on the approaches considered interesting and on which we founded the base of our research.

The GVGAI framework contains several agents aimed at demonstrating how a controller can be created for the single player track of the competition [9]. It includes the most basic agents, like *doNothing* and *onesteplookahead* that rolls the model forward for each one of the available actions in order to select the one with the highest action value. Without a doubt the methods that have been more explored are Tree Search Methods, such as Monte-Carlo Tree Search (MCTS), Open Loop Expectimax Tree Search (OLETS) and multiple enhancements of MCTS [4, 10, 11]. But some of these methods have been proved to not being particularly strong when it comes to playing very different games, one common outcome is that these agents perform on average better on a certain type of game but not on a huge amount of them.

In Mendes *et al.* 2016 [12] the authors present a similar hyper-approach to the one presented here. They describe the creation of a “hyper-agent” for general video game playing that utilizes the strengths of multiple individual controllers to play unseen games better than any of them individually. This hyper-agent uses an offline learning approach similar to the one we followed. The final goal of this research was to choose the best controllers to play each game *in real time*. However, their work has some limitations; for example it sometimes fails to select the best agent for a game and other times it just selects the less risky agent on average. Moreover, they conclude that the developed hyper-agent *significantly* outperformed the winners of the 2014 and 2015 competitions but also that there is a lot of room for improvement. These results suggest that the use of hyper-heuristics and algorithm selection may have an important role in general video game playing and that it is a feasible choice to perform further research.

4 Methodology

This section contains the process followed to collect the game data necessary for our hyper-agent as well as the game categorization that has been made with this data. It also describes how each category got assigned its best performing agent.

4.1 Data collection

Two types of data were collected and used to make the hyper-agent; game performance data, how well each selected agent does on each of the games, and game features data, which describe each game by giving its features. How both of these kinds of data were collected will be discussed here in turn.

Before performance data could be collected, the agents that would be in the portfolio of the hyper-agent were selected. In this work, the portfolio is the collection of agents from which the hyper-agent gets to choose. Five agents were selected from the sample agents provided within the framework, and five agents were selected that had competed in the GVGAI contest and had performed well. Of the five sample agents, there was one agent that served as a non-intelligent comparison to the intelligent agents (the DoNothing agent). The four other sample agents were the agents classified in the framework as advanced. Only these advanced sample agents were chosen because we believed advanced agents had a higher chance of good performance. The other five agents that make up the portfolio were the top five competitors in the 2018 GVGAI tournament. These agents were chosen because they had been shown to be able to perform well. No agents from other years were selected because of incompatibility with the framework. For specific descriptions of all agents used, see Table 1. Performance data was collected for these ten agents.

Agent	**Description**
DoNothing	Does not take any action
SampleMCTS	Monte Carlo tree search (MCTS)
SampleRS	Random search
SampleRHEA	Rolling horizon evolutionary algorithm
olets	Open Loop Expectimax Tree Search
asd592	Combination of a genetic algorithm (GA), map guidance to a Main Objective and breadth first search (BFS) for deterministic games
crazypet	A*, greedy path generation through a GA and Monte Carlo search
fanatax	Reinforcement learning using an existing knowledge base
fraBOT	BFS for deterministic games, MCTS for non-deterministic games
NovelTS	Optimized iterated width algorithm

Table 1: Agents in hyper-agent portfolio [13]

The performance data was collected by letting each agent play each level of each game available in the framework five times. Since each game has five levels, that meant each game was played 25 times by every controller. The amount of games won was summed up, resulting in each game receiving a score of 0-25 with 0 indicating that the agent lost all of the simulated games, and 25 indicating that it won all games. This data was collected on various personal computers. We are aware that this makes the data collected less robust, but splitting the computation was necessary because of time constraints. It was also attempted to collect all data through a server, which would allow for faster and more robust data collecting. However, because of circumstances outside of our control, the server was made inaccessible and the data was lost.

The second type of data that was collected, was feature data. This was collected per game. The features collected can be seen in Table 2. The features that were collected were a combination of features used by [14] and new features. Not all features of [14] were used; only the features that seemed most informative were included. Of the features shown in Table 2, Can Use, Can Die, % of Win Conditions, % of Lose Conditions, % Solid Sprites, % Harmful Sprites, % Goal Sprites and Max Interaction Rules were taken from this earlier research. Since the features in [14] did not seem to predict performance very well, new features were included and some features were changed in meaning in order to make them more informative. The new features extracted were 'Is Timeout Game', 'Death by Environment', 'Death by Enemy' and 'Is Puzzle'. These were not present in the previous work. Furthermore, the feature 'Is Survival Game' was given a new meaning. The first three of these features seemed to be meaningful ways of splitting earlier features, while 'Is Puzzle' would possibly be a very informative feature. All of the features were collected by hand, by going through the VGDL files for all but two features. In the VGDL files, the rules that govern that particular game and which sprites are used are specified. Based on this information many of the features could be determined. There were two features that could only be collected by playing the games; 'Can Use' and 'Is Puzzle'.

Feature	**Description**
Can Use	The agent can press the space bar to interact with the environment
Can Die	The agent can die by colliding with another sprite
Is Survival Game	The agent wins if a timer runs out
Is Timeout Game	The agent loses if it does not reach its goal before a timer runs out
% of Win Conditions	% of terminal conditions that are win conditions
% of Lose Conditions	% of terminal conditions that are lose conditions
% Solid Sprites	% of sprite types that do not interact with the player or the environment
% Harmful Sprites	% of sprite types that can harm the player if it comes into contact with them
% Goal Sprites	% of sprite types that are goal sprites
Max Interaction Rules	Maximum amount of interaction rules for one type of sprite
Death by Environment	The agent can die by walking into stationary sprites
Death by Enemy	The agent can die by colliding with moving objects
Is Puzzle	The game is a puzzle

Table 2: Selected features used to represent games.

4.2 Game Categorization

Using the game feature dataset, we used multiclass supervised learning to create a model for algorithm selection using methods implemented in the Weka machine learning software [15]. Similar to previous work [16, 17] clustering was performed first to split the games into different categories and then these clusters were used as labels (along with the

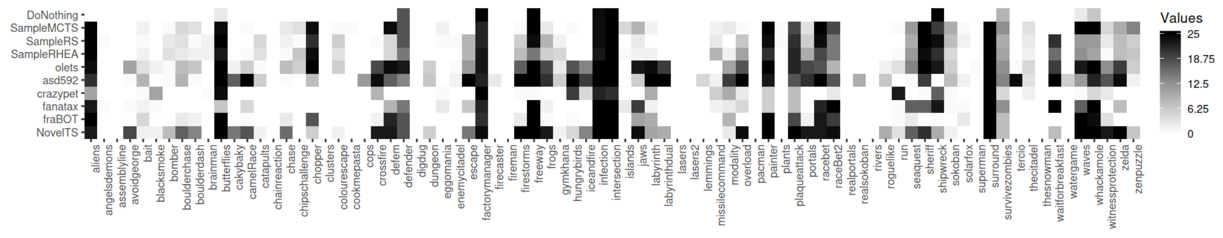


Figure 1: Heatmap representation of the performance of selected 10 agents (y-axis) in 81 discrete-physics games of the single player planning track of GVGAI framework (x-axis). Darker cells correspond to better performance.

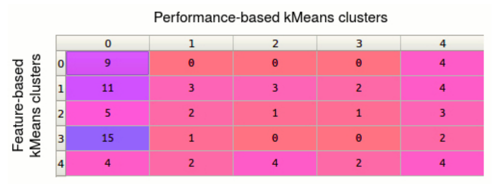
feature data) to train a J48 Decision Tree. Based on our analysis that will be discussed in the later paragraphs, we identified the following six boolean features that are most informative: "Can Use", "Is Survival Game", "Is Timeout Game", "Death by Environment", "Death by Enemy" and "Is Puzzle". The specifics of our approach are mentioned in the following paragraphs.

4.2.1 Clustering stage

In [14], the authors used k-means for clustering games. Here, we use two different clustering methods - k-means and Fuzzy ART - which resulted in two sets of clusters. A total of 81 games, represented by certain features, were used as input.

K-means requires the number of clusters to be specified before running. To determine the optimal k, we used the elbow method using the Within-Cluster-Sum-of-Squares (WCSS) measure. The 81 games were thus categorized into 5 clusters based on their features. In order to check to what extent were these features a good indication of controller performance, we clustered the games represented by controller performance into 5 clusters and created an overlap matrix as shown in 2.

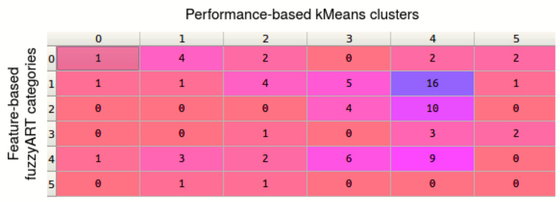
To check whether using a different clustering method can improve the clustering quality, we used an unsupervised learning algorithm called Fuzzy ART[18]. Fuzzy ART has an architecture similar to a two-layer neural network and uses fuzzy logic operations to recognize patterns in the data and categorize similar data-points together, resembling the clustering process. The number of categories are initialized to zero and grow as the training progresses. A hyper-parameter called *Vigilance*, denoted as ρ, is used to control the "strictness" with which the algorithm checks the inclusion of a data-point to each category. If a given data-point doesn't belong to any of the existing categories, a new category is created and the instance is put to it. With possible values of vigilance parameter being in the range [0,1], higher vigilance means more "strict" checking and thus the possibility of more categories being created. Using a vigilance value of 0.5, we obtained 6 categories of games based on their features. Again, to compare the quality of these categories, we clustered the games, represented by controller performance, into 6 clusters using k-means and recorded the overlap matrix shown in 3.


Performance-based kMeans clusters

Feature-based kMeans clusters

	0	1	2	3	4
0	9	0	0	0	4
1	11	3	3	2	4
2	5	2	1	1	3
3	15	1	0	0	2
4	4	2	4	2	4

Figure 2: Overlap matrix for feature-based k-means clusters v/s performance-based k-means clusters. The colors represent the overlap, same as the numbers in the cells. Darker shade corresponds to better overlap. These colors are only meant for reader's convenience.



Performance-based kMeans clusters

Feature-based fuzzyART categories	0	1	2	3	4	5
0	1	4	2	0	2	2
1	1	1	4	5	16	1
2	0	0	0	4	10	0
3	0	0	1	0	3	2
4	1	3	2	6	9	0
5	0	1	1	0	0	0

Figure 3: Overlap matrix for feature-based Fuzzy ART categories v/s performance-based k-means clusters

Overlap matrix in an ideal case would "associate" each feature-based cluster with a performance-based cluster. The word "associate" here means that for all feature-based clusters (rows), there will be a unique performance-based cluster (column) such that both of their constituent games completely overlap (and vice-versa), and the rest of the row-column combinations have zero overlap. The significance of such an ideal case lies in the fact that the features of a game are good indicators of controller performance and using this, one can explain why a particular controller performs well on a particular game. On the other hand, the overlap matrix of the worst possible case will have all the games of any performance-based cluster (column) being distributed uniformly over all the feature-based clusters (rows).

In our case, the 5x5 overlap-matrix for k-means clustering in 2 contains most of the games, grouped by controller performance, in the performance-based cluster 0. However, these games are distributed across all the feature-based clusters with the maximum being concentrated in feature-based cluster 3. As for the remaining part of the matrix, it is hard to see any unique overlap between feature-based and performance-based clusters. Moreover, the games in column 4 are distributed almost uniformly over all the rows which is not a good sign.

In the 6x6 overlap matrix for Fuzzy-ART categories in 3, the performance-based cluster 4 contains majority of the games which are distributed over different feature-based clusters as in the k-means case. However, compared to k-means case, the distribution of games in any performance-based cluster is distributed less uniformly over the feature-based clusters even though there are more clusters.

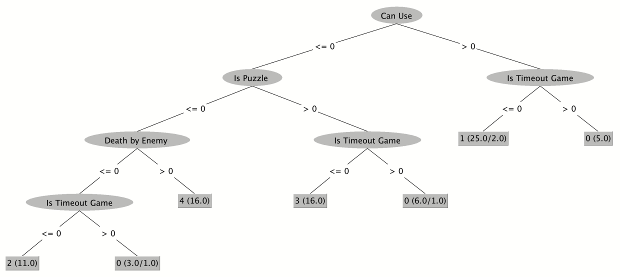


Figure 4: Decision tree built using K-means clusters

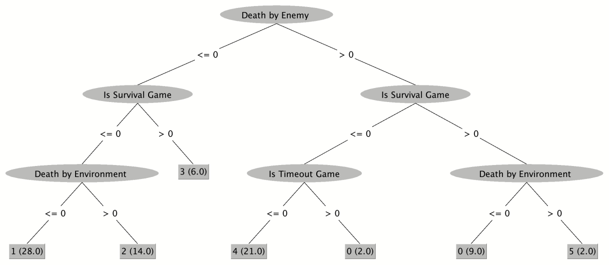


Figure 5: Decision tree built using Fuzzy-ART categories

In both the matrices, the performance-based cluster containing the majority of the games represents a class of "hard" games where all controllers perform poorly. The fact that this class contains so many games in the first place indicates that there is a need for controllers that can perform well in such games.

Figure 6 shows the same content as the earlier heatmap except that the games are sorted and grouped by their categories generated by k-means and Fuzzy ART based on game features. In both the cases in this figure, it can be observed that the games (columns) with all white cells are distributed across all the categories. These games are the so-called "hard" ones. Also, many other games with identical controller performance are not grouped together, for example in the k-means case, the games "Infection" and "Intersection" are put into clusters 1 and 4 respectively despite their similar controller performance values. This suggests that even though our features are more informative than the ones used in [14], they are still not a good indication of controller performance.

4.2.2 Building the decision model

The two sets of feature-based clusters were then used to build two J48 decision trees - one for k-means and Fuzzy ART each. The decision trees were created using the Weka machine learning software.

First, in order to check the quality of the game features, five of the features from the previous work[14] were used to cluster the games followed by creation of the decision trees. These features were :"Can Die","Can Use","Is Survival Game","% Win condition" and "% Lose Condition". The first two trees were build with nine clusters retrieved from the k-means and 10 clusters from the Fuzzy Art. These trees had a label accuracy of 100% and 92.6% respectively. Although we got good accuracy values for the trees, the overlap matrices of the clustering were not ideal which means that these old features were not the best indicators of performance.

Therefore, we then introduced the new features discussed earlier and along with just one of the old features "Can Use", we clustered the games again using k-means and Fuzzy ART. The overlap matrices discussed earlier and shown in 2 and 3 correspond to this case. Again, this cluster data was used as labels along with the feature data to build two J48 decision trees. This time, the label accuracy of the trees were: 91.46 % for K-means clusters and 95.122% for the Fuzzy ART categories. The validation technique used to build all the decisions tree was 10-fold cross-validation. Figures 4 and 5 visualize the two aforementioned decision trees trained on new features.

Class labels obtained from	Accuracy
K-means	91.4634%
Fuzzy-ART	95.122%

Table 3: J48 run results for the full dataset using 10-fold cross-validation as test mode.

These decision trees serve as decision models for the two hyper-agents. The hyper-agents created using k-means and Fuzzy-ART will be henceforth referred to as *HA-KM* and *HA-ART* respectively.

	Group 1 (k-means clusters)	
Cluster	Controller	Dominance
0	asd952	6/13 (46%)
1	NovelTS	10/23 (43%)
2	asd952	6/12 (50%)
3	asd952	7/18 (39%)
4	asd952/NovelTS	8/16 (50%)

Table 4: Dominance analysis for k-means clusters.

4.3 Finding the best controller for each category

Through analysis of the performance of each controller for each cluster, we can determine which controller has the best performance (i.e. dominates) and include this controller in our hyper-agent. We consider a controller to be dominant if, for the games in a given cluster, it is able to win each game more often than the other controllers in the cluster.

Table 4 contains the results for the clusters returned from the k-means clustering method. We can see that the controller asd952 performs very well on most of these clusters, losing to the controller NovelTS in only one cluster and tying with NovelTS in another.

Table 5 deals with the controllers' performance on the clusters from the Fuzzy ART method. In this case, we can see that asd952 and NovelTS still have a significant presence, as before, but now multiple other controllers appear as well. This is most notable in the last cluster for this group, where we have a four-way tie for dominance. However, with only two games in this cluster, a result like this is not entirely surprising.

5 Experiments and Results

This section describes the experiment that we conducted and the results that we received from this experiment.

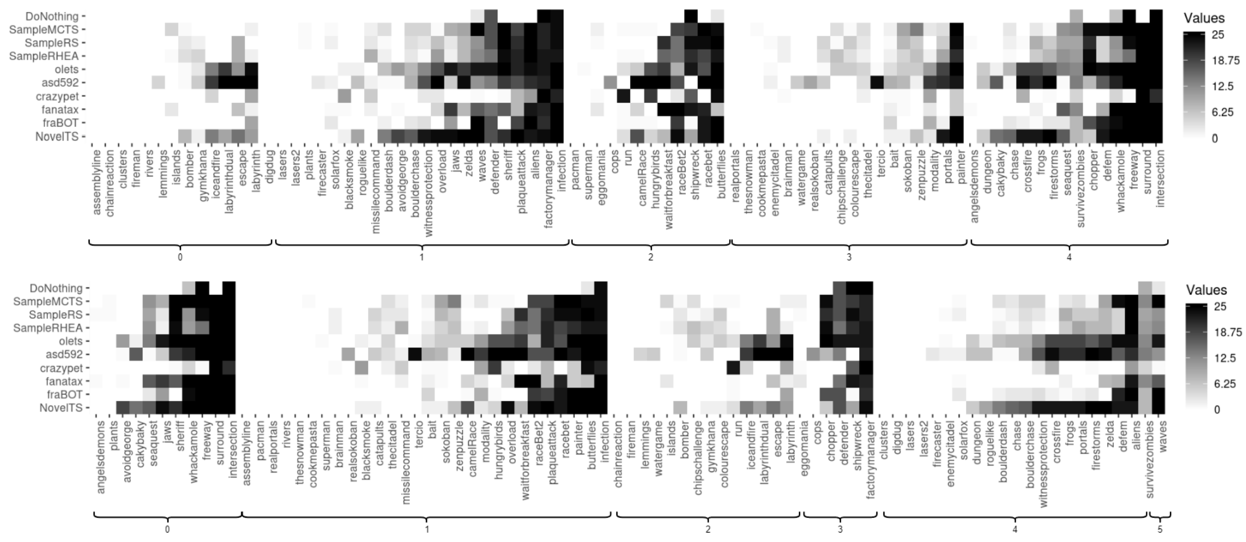


Figure 6: Heatmap representations of the agents (y-axis) performance vs single-player planning track discrete physics games (x-axis). The games are sorted by their k-means clusters (top) or Fuzzy-ART categories (bottom) created based on their features and also by the average performance. The numbers at the bottom of each heatmap indicate the clusters. Darker cells correspond to better performance.

	Group 2 (Fuzzy ART clusters)	
Cluster	Controller	Dominance
0	NovelTS	7/11 (64%)
1	asd952	12/28 (43%)
2	asd952	7/14 (50%)
3	DoNothing/olets	2/6 (33%)
4	NovelTS	9/21 (43%)
5	asd592/sampleMCTS/ fraBOT/NovelTS	1/2 (50%)

Table 5: Dominance analysis for Fuzzy ART clusters.

We added a new java class to the GVGAI framework called HyperDataCollector in which the hyper-agent is implemented. The hyper-agent can be run in three different ways by using three different methods to decide which controller to assign to which game. The first method is the random hyper-agent referred to as HA-RA. HA-RA randomly chooses a controller for each game from the controller portfolio and loads it to play the game. The second method is the k-means-based hyper-agent referred to as HA-KM. HA-KM chooses a controller to play a game based on the decision tree built using k-means clusters. The third method is the Fuzzy-ART-based hyper-agent referred to as HA-ART. HA-ART chooses a controller to play a game based on the decision tree built using Fuzzy-ART categories. The hyper-agent takes as input two files and outputs a new file showing the performance of every game. The two input files are feature data, where the features of every game are listed, and game data, where the performance of every controller for every game is listed. Then based on the game features of a game, the hyper-agent goes through the decision tree and allocates the dominant controller of the resulting category to play the game. This process is illustrated in Figure 7.

The results show that as expected, HA-RA did not perform as well as the other two hyper-agents. HA-KM and HA-ART both performed significantly better than HA-RA with an 76% and 80% increase in performance respectively. HA-ART in turn performed 2% better than HA-KM which does not seem significant.

In order to compare our agent to previous work, the percentage improvement over the best singular agent was also calculated. Our best agent improved 17.73% over the best singular agent. In previous work that used a similar method [14] the best agent only improved 9.10% over the previous best agent. This can indicate that our decision method is better, since it improves more over its best singular agent. However, other explanations are also possible (such as a better average performance of the agents in the portfolio). In Table 6 we listed the average performance of the three hyper-agents across all 81 games. In Table 7 we listed the increase in performance when we compare the hyper-agents to each other.

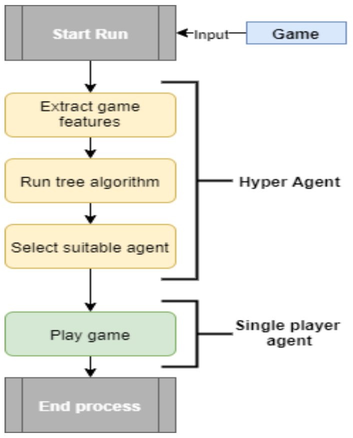


Figure 7: Final program process workflow diagram.

Hyper-agent	Average Performance
HA-RA	0.2737
HA-KM	0.4815
HA-ART	0.4933

Table 6: Average performance across 81 games of the three hyper-agents

6 Conclusion

The purpose of this research was to find ways to improve the hyper-agent approach to GVGP AI. We did this by improving the categorization with two different approaches: using a new algorithm, and introducing different features to cluster on. In this section we will evaluate the results of these approaches, as well as the hyper-agent performance overall.

Comparison	Performance increase
HA-RA vs. HA-KM	76%
HA-KM vs. HA-ART	2%
HA-RA vs. HA-ART	80%

Table 7: Performance comparison of the three hyper-agents among each other

As shown in section 5, the hyper-agents based on categorization perform much better than a random hyper-agent. HA-ART and HA-KM both are able to win almost half of the games they play; an impressive feat that not all humans can accomplish.

Whether the new clustering algorithm Fuzzy ART was indeed better than clustering with k-means is difficult to tell. The average performance between the two agents has only minimal difference, which would indicate that the two ways of clustering give similar results. However, the clusters achieved with Fuzzy ART seemed more distinct from one another, and had more clear dominating agents, which would indicate that Fuzzy ART is better suited to build clusters for this kind of application. These results are therefore inconclusive, but do not rule out Fuzzy ART as the better suited clustering algorithm.

It is difficult to compare the performance of our current agent to previous work, since the most similar work does not give clear indications of performance. Furthermore, we changed multiple aspects of the hyper-agent at the same time, which makes it difficult to isolate individual factors that influence the performance. This was due to the fact that earlier research was based on a different version of the framework, and therefore it was not easily possible to isolate variables, since the controllers used with that framework are not compatible with the current one. However, the performance of our agent seems to be better that compared to earlier work. In previous work [14], the best hyper-agent improved on the best agent by 9.10%, while our best-performing hyper-agent improves 17.73% over the best performing singular agent in the portfolio. This suggests that the decision making of our agent is better, since it improves more over the best agent in its portfolio, though other explanations for this difference in improvement in performance are possible.

7 Future work

There are multiple possibilities to expand or improve on this research. A first avenue of further research would be to collect the game features during run-time, instead of collecting a database by hand. Aside from the workload that collecting features by hand causes for the researchers, using a database of features for existing games means that the hyper-agent we built is not able to play unknown games. It would also not be able to compete in the GVGAI tournament, since it is using preexisting knowledge of the games. A next step in making this agent a true GVGP AI therefore would be to collect these features automatically at run-time. This could be quite difficult; certain features, such as “Can Die”, can only truly be found out when the game ends, and other features, such as 'Is Puzzle' have very fuzzy definitions. However, we do believe that being able to determine these features during game-play will serve to improve the state of GVGP.

Another, related avenue of further research would be to collect other features that are not observable in the files, but that would be observable in the game, such as size and the number of sprites. This has already been done before [12], but combining these two different kinds of features could serve to get a very informative set of features.

The third possible approach would be to experiment with using different agents at different levels of the same game, since these levels could vary on certain properties which would make the separate levels better suited for different agents. Currently, we are not able to distinguish between levels of the same game based on the features. However, with a new method of feature collecting (such as collecting features at run-time) the various levels could be discovered to have variation for these features. In that case, it would be relevant to investigate whether using different agents for these different levels would increase performance.

References

[1] John Levine, Clare Congdon, Marc Ebner, Graham Kendall, Simon Lucas, Risto Miikkulainen, Tom Schaul, and Tommy Thompson. General video game playing. *Dagstuhl-Followups*, 6:77, 11 2013.

[2] GVGAI frameowrk. `http://www.gvgai.net/software.php`, 2016.

[3] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul and Simon Lucas. General Video Game AI: Competition, Challenges and Opportunities. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[4] Diego Perez, Spyridon Samothrakis, and Simon Lucas. Knowledge-based fast evolutionary MCTS for general video game playing. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[5] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[6] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[7] Simon M Lucas, Jialin Liu, and Diego Perez-Liebana. The n-tuple bandit evolutionary algorithm for game agent optimisation. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9. IEEE, 2018.

[8] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender

Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[9] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2015.

[10] Frederik Frydenberg, Kasper R Andersen, Sebastian Risi, and Julian Togelius. Investigating MCTS modifications in general video game playing. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 107–113. IEEE, 2015.

[11] Chun Yin Chu, Tomohiro Harada, and Ruck Thawonmas. Biasing Monte-Carlo rollouts with potential field in general video game playing. In *IPSJ Kansai-Branch Convention*, pages 1–6, 2015.

[12] Andre Mendes, Julian Togelius, and Andy Nealen. Hyper-heuristic general video game playing. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[13] The GVG-AI competition. `http://www.gvgai.net/index.php`. Accessed: 2020-01-15.

[14] Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 122–128, 2016.

[15] Ian H Witten and Eibe Frank. Data mining: Practical machine learning tools and techniques 2nd edition. *Morgan Kaufmann, San Francisco*, 2005.

[16] Haipeng Guo and William H Hsu. A learning-based algorithm selection meta-reasoner for the real-time mpe problem. In *Australasian Joint Conference on Artificial Intelligence*, pages 307–318. Springer, 2004.

[17] Luca Pulina and Armando Tacchella. A multi-engine solver for quantified boolean formulas. In *International Conference on Principles and Practice of Constraint Programming*, pages 574–589. Springer, 2007.

[18] Gail A Carpenter, Stephen Grossberg, and David B Rosen. Fuzzy ART: An adaptive resonance algorithm for rapid, stable classification of analog patterns. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 411–416. IEEE, 1991.