Matching Games and Algorithms for General Video Game Playing

Philip Bontrager, Ahmed Khalifa, Andre Mendes, Julian Togelius
New York University
New York, New York 11021
philipjb@nyu.edu, ahmed.khalifa@nyu.edu, andre.mendes@nyu.edu, julian@togelius.com

Abstract

This paper examines the performance of a number of AI agents on the games included in the General Video Game Playing Competition. Through analyzing these results, the paper seeks to provide insight into the strengths and weaknesses of the current generation of video game playing algorithms. The paper also provides an analysis of the given games in terms of inherent features which define the different games. Finally, the game features are matched with AI agents, based on performance, in order to demonstrate a plausible case for algorithm portfolios as a general video game playing technique.

Introduction

The General Video Game AI Framework (GVGAI) is an attempt to create general framework for game-based testing of artificial intelligence methods (Levine et al. 2013; Perez et al. 2015b; Perez-Liebana et al. 2016). Unlike the general game playing competition (GGP), the GVGP is focused on video games, in particular two-dimensional games similar to classic arcade games. The general video game AI (GVGAI) competition aims to benchmark AI algorithms through testing them on a number of unseen games, i.e. games that the developer of the algorithm was not aware of at submission time. Figure 1 shows different games that can be represented in the GVGAI framework. A key underlying motivation is that for an agent to be generally intelligent, it needs to be capable of solving a large number of different problems (playing a large number of different games), not just one. General game players are also very useful for AI-assisted design tools and procedural content generation in general. Artificial game-playing agents need to be able to produce good feedback on game designs based on playing through the game.

It has previously been found that performance is non-transitive: different algorithms perform best on different games (Nielsen et al. 2015). Therefore, it should be possible to construct agents based on higher level game features that chooses the right algorithm for playing each game. However, selecting the right algorithm to play a particular game is a complicated task. One needs to know the best features that can help predict the performance of an algorithm on a game. In the face of limited data sets, one also needs to avoid overfitting by focusing on the most relevant regularities.

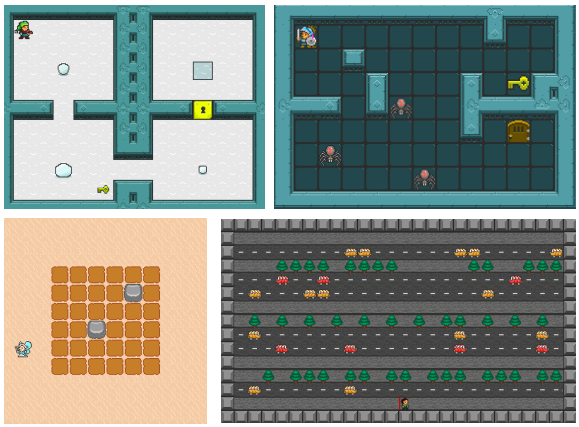

Figure 1: Different games in the GVGAI framework.

In this work, we approach the problem of matching algorithms with games (Mendes, Nealen, and Togelius 2016) through investigating the performance of a large number of algorithms on games from the GVGAI framework (Perez et al. 2015b). We look for regularities in the performances of algorithms in games, and cluster both games and algorithms based on playing performance. We also look for good features of games that can be easily extracted and are predictive of algorithm performance. Finally, we discuss the correlation between the groups of clusters from the different sources of information. We believe that such a study is important because it provides insight into how far we have come on GVGP. This data can reveal what games computers are good at and what problems we still need to solve. These groups can be used for algorithm portfolio techniques and provide direction on how to combine multiple algorithms to create a better-performing agents.

The paper is structured as follows. The next section describes general video game playing and the GVGAI framework as well as some principles of game genres and algorithm selection. The Methods section describes our methodology: which controllers and games we used, how we se-

lected features to use, how we collected data on controller performance, which methods we used to cluster the data and how we intend to use the information obtained to create an agent that uses algorithm selection. The Results section describes the results obtained using the clustering methods as well as how the clusters compare across the different source of information used. Finally, we present the results from different algorithm selection agents and compare their performance with the controllers in the their portfolios.

Background

Video Game Genres

The definition of game genres is the subject of many debates. Studies argue that genres are an agreement between producers and players with relation to established entertainment industry mediums such as movies and books (Wolf 2002). Others disagree with this point and claim that genres need to be rethought with a critical perspective rather than depending on similarities with other mediums (Juul 2005).

The general idea is that video game genres are a way to organize games according to certain aspects. One of the main aspects to be considered is game play (Apperley 2006). King and Kryzwinska categorize games based on 4 levels: Genre, Platform, Mode, and Milieu (King and Krzywinska 2002). Platform is the current system that the game is running on. Mode is the way the game is experienced by the players, such as single player or multi player. Milieu describes the aesthetics of the game in a similar way that movies do, using definitions like fantasy and science fiction.

In this work we categorize the games based on properties in the games and how various algorithms play them. This is more inline with the definition from Schatz, in which, game genres are defined based on the interactivity in games (platformer, shooter and puzzle) rather than using the story content (Schatz 1981). The reason for this is that the games in GVGAI are abstract and they focus more on interactivity rather than content. Also this definition provides information on which computer genres we have good algorithms for.

To the best of our knowledge, there is no previous work clustering video games by this criteria. Most of the work in the literature focuses on clustering games based on surveys (Heintz and Law 2015; Fang, Chan, and Nair 2009), clustering player behaviors (Bauckhage, Drachen, and Sifa 2015), or clustering based on similarities to theoretical games (e.g. Prisoner Dilemma) (Feldman, Lewin-Eytan, and Naor 2012).

General video game playing

In the past decade, video games have become increasingly popular as an AI benchmark. They require a rich repertoire of cognitive capabilities for humans to play well and they can be simulated quickly and cheaply on a computer. A number of recent AI competitions have been based on specific video games or types of video games. The GVGAI competition was created to counteract the problem that these competitions allow participants to tailor their submissions to a particular game. Instead, game-playing controllers submitted to the GVGAI competition are pitted against a number of *unseen* games (Perez et al. 2015a). This separates it from e.g. the Arcade Learning Environment, where controllers are trained to play a number of well-known games (Bellemare et al. 2012). Another difference is that GVGAI controllers get access to a structured representation of the game state as well as a simulator that allows the use of tree search algorithms to play the games.

So far, the results have shown that methods based on MCTS and MCTS-like algorithms have done very well on this benchmark, but based on the algorithm superiority (Brodley 1993) and in our observation, we see that performance is non-transitive: different controllers play different games best. This can be seen even when adding seemingly minor modifications to the core MCTS algorithm; modifications that improve performance of MCTS on one game may degrade performance on another (Frydenberg et al. 2015).

Algorithm Selection

One influential taxonomy of algorithm selection methods (Kotthoff 2012) suggests that an algorithm can be identified in steps. First it analyzes the type of *portfolios*, i.e the set of algorithms that will be selected. A portfolio can be static (Pulina and Tacchella 2009), (Xu et al. 2008) if the algorithms are defined a priori and never change, or dynamic if the portfolio changes while solving the problem (Fukunaga 2008).

From the portfolio, methods can select the single best algorithm or allocate times slots to combine results from different algorithms. Concerning when to select the algorithm, the methods can make a decision before the solving of the actual problem starts or while the problem is being solved (OMahony et al. 2008). Another important step is how this selection is made. The decision involves, for example, analyzing accuracy, computational cost and time. Finally, there is also an essential step that concerns finding information to help the selection, such as feature selection and extraction (Gerevini, Saetti, and Vallati 2009),(Pulina and Tacchella 2009) and the use of the performance of the selected algorithms in the past.

Based on the problem and data that we have and following the proposed organization, our method can be defined as using a static portfolio, that selects the best single algorithm before and while the problem is being solved. Our selection method uses a decision tree model that is trained based on the clusters achieved using our selected features.

Algorithm selection has been applied to domains such as linear algebra (Demmel et al. 2005), linear systems (Bhowmick et al. 2006) and specially to combinatorial search problems (OMahony et al. 2008), (Xu et al. 2008), and recently applied to General Video Game Playing (Mendes, Nealen, and Togelius 2016).

Methods

Games

The games in the GVGAI platform are created using a Video Game Description Language (VGDL) that is designed around objects that interact in two dimensional space (Ebner et al. 2013). Using level description and game description,

Feature Name	Feature Definition
Vertical movement	The avatar can move vertically.
Can use	The avatar can use space bar to do something.
Player can die	The avatar can die from colliding with harmful sprites.
Is survival game	The end condition of the game is timeout.
Win conditions	The % of termination conditions with win=True.
Lose conditions	The % of termination conditions with win=False.
Solid sprites	The % of solid sprites with respect to all sprites.
Avatar sprites	The % of avatar sprites with respect to all sprites.
Harmful sprites	The % of harmful sprites with respect to all sprites.
Collectible sprites	The % of collectible sprites with respect to all sprites.
Goal sprites	The % of goal sprites with respect to all sprites.
Other sprites	The % of other sprites with respect to all sprites.
Spawned sprites	The % of spawned sprites with respect to all sprites.
Max number of Interaction rules	The max number of interaction any of the game sprites appears in them.

Table 1: Extracted game features from VGDL files.

VGDL can describe a wide variety of video games, including approximate versions of Space Invaders, Sokoban, Boulder Dash, Pac-Man, and Zelda. In this work, we use 49 games available in the GVGAI framework. Each game has five different levels that differ from each other through variations on the locations of sprites, resources available and variations on non-player character (NPC) behavior.

Games Clustering

We use K-Means (Likas, Vlassis, and Verbeek 2003) and Agglomerate (Davidson and Ravi 2005) algorithms to cluster the games either over game features or game-playing controllers performance. We also tried the Gaussian Mixture Models (GMM) (Reynolds 2015) in clustering but the results were similar to K-Means.

Game Features Features are an important part of the games described in the GVGAI. In this platform, every game contains a description in a VGDL file that provides rich information about various elements: such as the ability to attack, move vertically, the type of resources, dimensions of the map, action set, and other characteristics of the game. We analyzed the classes available in the VGDL file of all games and defined 14 features shown in Table 1, that represent different types of games.

Game-playing controllers Another important way to analyze the games is to look at how various algorithms perform when playing them. Since the algorithms are unfamiliar with each game, it can be considered an unbiased indicator of which strategy is effective against a particular game. In this work we used 23 controllers chosen from the sample set as well as controllers submitted to the GVGAI competition in the summer of 2015.

The controllers from the competition were selected because their competition results were better than a random agent. All of the controllers use standard AI algorithms at their core. The creators of the controllers described the algorithm their controller is based on when they uploaded it to the competition. The reported algorithms can be seen next to the controller names in figure 4.

A couple of the controllers vary from their core algorithm in notable ways. The controller called "AIJim" uses a variant of MCTS called Knowledge-based Fast Evolutionary MCTS (Perez, Samothrakis, and Lucas 2014). The controller "adrienctx" uses Open Loop MCTS (Perez Liebana et al. 2015). There were also two controllers that tried to combine algorithms. The controller "mrtndwrd" uses A* to find out information for each different sprite type and then MCTS to determine which path to take. "Rooot" uses some learning techniques to analyze a multistep look-ahead. For simplicity, it was decided that all of the above algorithms, with the exception of "Rooot", would be grouped under MCTS.

Data Collection

To collect the data, we ran each controller in its own instance and had it play each of the 49 games 25 times. Each game consists of 5 levels, so the controller played each level 5 times. For each run of the controller, we recorded whether it was a win or loss, the score achieved by the controller, and how long the play lasted.

The resulting data was compiled into a winnings matrix and a scores matrix. Each matrix is a 49 games by 23 controllers. Each value in the win matrix is a number between 0 and 25, representing the number of times controller i beat game j. The score values were normalized between 0 and 1.

We looked at the results for both scores and the wins over all the games. Results obtained from the score data and the win data were not significantly different. We decided to only use the win data since it is a better indicator of which algorithm is better. This is due to discrepancies in how different games are scored. Some games only give a score for a win or loss and others have many ways to get points. The score data between games are therefore not easily comparable, even after normalization.

Algorithm Selection

We use the feature clustering we obtained to build machine learning models to select the best controller for each game. Our assumption is that these clusters provide helpful information about the style and characteristics of the games and we can map these information to controllers performance through a decision tree classifier. We then use this decision tree classifier as a decision model to select sub-controllers to play games in each cluster. Since the agent that uses this model combines the strengths of different controllers, we expect that its performance is better than its constituents.

Results

Game Features

Each game is represented by 14 features. To cluster the high dimensional data, we applied Principal Component Analysis (PCA) (Jolliffe 2002) to reduce the data to 2 dimensions. Figure 2 shows 4 different clusters that are widely dispersed and are fairly compact. We also applied K-Means without PCA as well as an Agglomerate hierarchical clustering and

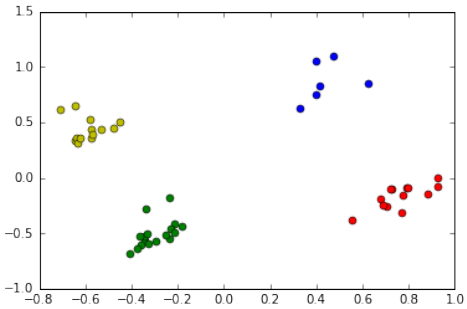

Figure 2: Game clusters using K-Means on PCA game features.

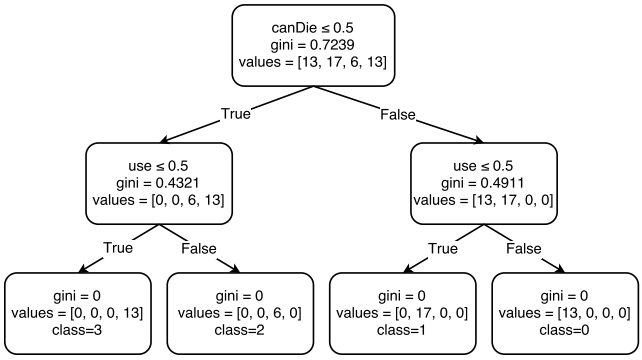

Figure 3: Trained decision tree using the K-Means clusters on game features.

the results for all three techniques were very similar (assuming we have 4 clusters). In order to understand these clusters, we generated decision trees to analyze which features affected the groupings. Figure 3 shows a tree that only uses two game features ("can use" and "can die") to cluster the games. These two features make sense as you can differentiate between action games and other games if the player have a "use" button. You can differentiate between puzzle games and other games if the player can die.

Algorithm Performance

To analyze the games based on the algorithms, we clustered the games based on the performance of the 23 controllers. We looked at clustering the data using both k-means and agglomerate search.

We tried k-means with regular Euclidean distance. Each game is represented as point in 23 dimensional space, where each dimension is the number of wins for each algorithm. To understand the results we used PCA to reduce the data to two dimensions. Viewing the data in two dimensions, there was no clear distinctions of where clusters should be. Though, when we ran k-means for four clusters on the 23 dimensional data and then viewed it in two dimensions, the clusters remained mostly separated. This clustering did not provide much insight.

To get a better understanding of how many groups to cluster the games into, we used an hierarchical clustering. For this we used the agglomerate algorithm, using Euclidean Distance to be consistent with k-means, and Ward as the linkage criterion. The outcome can be seen in figure 5. This cluster matches the k-means cluster for 73.5% of the games.

In the dendrogram, highlighted in green, there are four groups that were played very different. A clustering of four groups is also convenient for comparing to the feature clustering. To interpret the meaning of this clustering, the data is organized in a heat map. Figure 4 represents the matrix of wins each controller had against each game. The rows and columns are sorted such the most similar games and controllers are next to each other, according to the agglomerate clustering. To further visualize the information, red bars have been added to show the boundaries between the different clusters.

Looking at the game clusters in figure 4 from left to right; the first group are games easily beaten by most algorithms, the second group can be mostly beaten by the bottom group of controllers, the third group is particularly susceptible to MCTS controllers, and the last group, the largest, are difficult for all the controllers. In the fourth group, there are four games that none of the controllers could ever beat: Digdug, RealSokoban, Superman, and RealPortals. The size of the fourth group indicates that there are still many games that the current generation of general game playing algorithms cannot play.

Clustering Similarities

The intent of looking at multiple clusterings was to find their similarities. Similarities between the performance and feature clusterings would give insights into how the algorithms perform when certain features are present in a game. Our data revealed only a 40.8% similarity between the two clusterings. One interpretation of this is that the features we have are not strong indicators of performance.

We tried to create a decision tree using our features to classify games by their performance clustering. The resulting tree was very large with many seemingly arbitrary leaves. It is very likely that the tree was just memorizing the data, we therefore abandoned this effort. This further supports the claim that our features are not good indicators of performance.

Algorithm Selection

Best controllers for each game In algorithm selection, there are many ways to define the strategy to select the best algorithm to the problem at hand. In this work, we try to create a selection model that uses the information from our clusterings. Our goal is to investigate if the same decision tree obtained from the previous sections could also be used to pick the best controllers for the games in each cluster.

In order to do that, we first used the data obtained in the Data Collection subsection to identify the best controller for each cluster. We say that a controller dominates a cluster of games if the controller wins more games in a particular cluster than any other controller. To better understand this section, we divide our experiments in two cases.

The first case considers all 23 controllers that we describe in the Game Playing Controllers section. In Table 2, we can

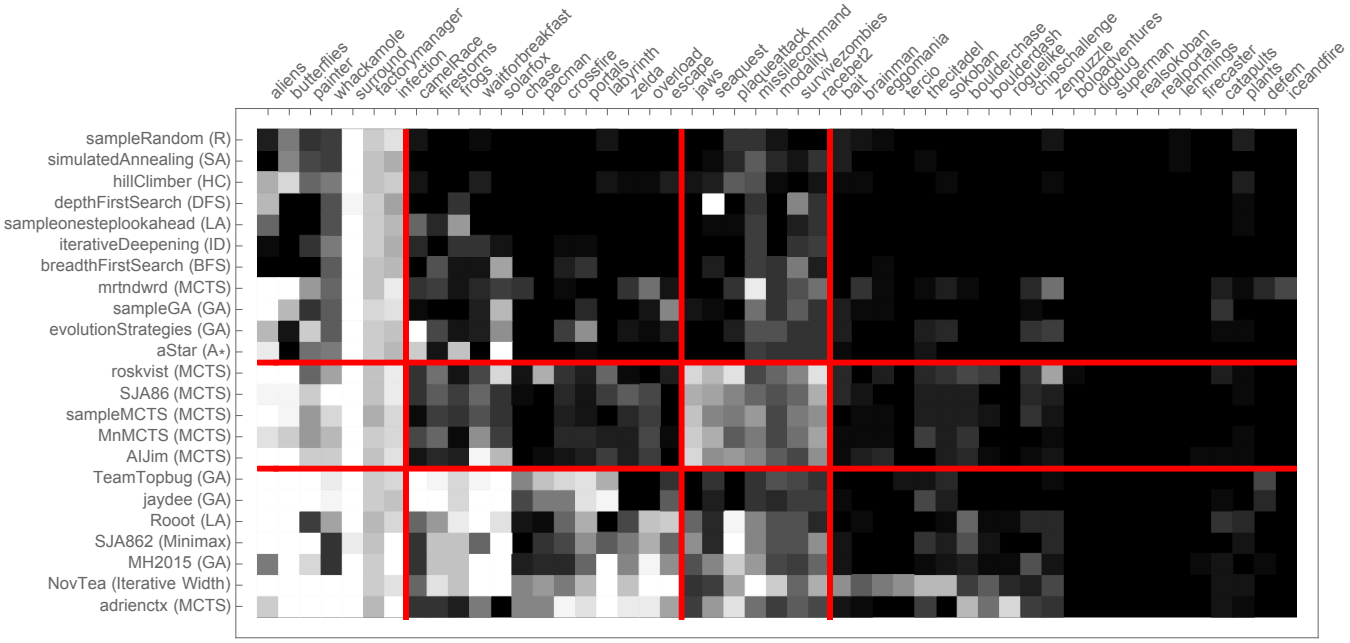

Figure 4: Heat map of game plays. Each square represents the number of times a particular controller beat a particular game (25 wins is white, 0 wins is black). The red bars represent the boundaries between each cluster. Vertical bars cluster the games, and horizontal bars cluster the controllers.

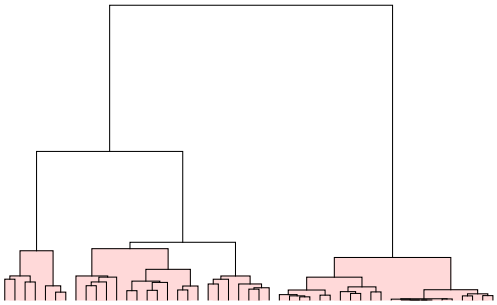

Figure 5: Dendrogram of games from figure 4.The depth of the connecting bracket represents the similarity between two different games. The colored regions represent different clusters.

see that the controller NovTea dominates three clusters in group 1 (game feature clustering) and two in group 2 (algorithm performance clustering). Being the dominant controller does not mean it dominates the majority of the game in a cluster. On average NovTea only dominates 34% of each cluster.

In the second case, we consider only 11 controllers that are simple implementations of traditional algorithms such as depth first search, A*, and genetic algorithms. In this case, we can see that the best controllers are sampleMCTS and evolutionStrategies. Even though their dominance is still not strong, it is stronger than in case 1 since the average dominance is 46% for sampleMCTS (SMCTS) and 43.4% for evolutionStrategies (ES). In both cases, this suggests that

Case 1 (All Controllers)				
	Group 1		**Group 2**	
Cluster	**Controller**	**Dominance**	**Controller**	**Dominance**
1	NovTea	2/5 - 40%	adrienctx	3/7 42%
2	adrientcx	5/14 - 35%	NovTea	3/13 23%
3	NovTea	4/8 - 50%	roskvist	3/7 42%
4	NovTea	6/22 - 27%	NovTea	8/22 36%
Case 2 (Simple Controllers)				
	Group 1		**Group 2**	
Cluster	**Controller**	**Dominance**	**Controller**	**Dominance**
1	SMCTS	3/5 60%	SMCTS	3/7 42%
2	SMCTS	5/8 - 62%	SMCTS	4/13 30%
3	SMCTS	8/14 - 57%	SMCTS	4/7 57%
4	ES	8/22 - 36%	ES	9/22 40%

Table 2: Top controllers for each cluster. Group 1 represents clusters based on features. Group 2 represents clusters from controller performance.

Controller	win	score	time
ASA1	695	1.00	0.00
NovTea	637	0.99	0.08
adrienctx	496	0.72	0.19
MH2015	426	0.67	0.33
TeamTopbug	426	0.55	0.34
SJA862	411	0.61	0.26
jaydee	410	0.53	0.30
Rooot	409	0.65	0.42
roskvist	398	0.54	0.55
AIJim	343	0.46	0.69
SJA86	338	0.39	0.74
MnMCTS	312	0.43	0.74
mrtndwrd	282	0.35	0.27

Table 3: Results for the simple controllers and the algorithm selection agent for case 1 (ASA1).

Controller	win	score	time
ASA2	326	0.41	0.57
SMCTS	300	0.39	0.76
ES	242	0.35	0.39
SGA	210	0.22	0.72
A*	200	0.17	0.70
HC	168	0.26	0.25
DFS	140	0.00	1.00
BFS	137	0.07	0.81
SOSLA	127	0.08	0.77
SR	122	0.14	0.09
SA	117	0.12	0.07
ID	115	0.06	0.69

Table 4: Results for the best controllers and the algorithm selection agent for case 2 (ASA2).

there is not a single controller that performs really well in the great majority of the games in any cluster.

Decision tree as selection model With the information about the best controllers for each cluster and the decision tree obtained in previous sections, we create an algorithm selection agent. As a decision model, we use the decision tree from Figure 3. This decision tree is simple and easy to understand and it is the only one that uses only features that are available to the agent in the game.

For the controllers in case 1, our selection is: If the agent has the action "use" and it can die in the game, we want to pick adrienctx to play that game. In all the other cases we pick NovTea. In case 2, we pick evolutionStrategies if the agent does not have the action use and it can die. Otherwise, we pick sampleMCTS to play.

To compare the performance of all controllers we use the criteria from the GVGAI competition (Perez et al. 2015b). There are three measures for each controller, win, score and time. The number of wins is the most important metric, followed by the score and as a second tie breaker, the time. In GVGAI, the scale of the results in each game significantly differ when considering score and time. For this reason we normalized the values in range from 0 to 1. Therefore, if a controller achieves the highest score it receives 1. For the time, the controller receives 1 if it is the fastest.

Table 3 shows the performance for the controllers used in case 1 and the algorithm selection agent created for it. For case 2, the results can be seen in Table 4. In both cases, the algorithm selection agent was able to outperform its constituents and all the other controllers in the group. These results show that the decision tree generated from the selected features is helpful to create a decision model to select controllers to play the games. Besides, the results also help to confirm that algorithm selection can be used in GVGAI to create high-performance agents.

Discussion & Conclusions

This paper explores two different clustering techniques to try and define GVGAI in terms of four different computational genres. Being able to divide video games into genres based on how they are played allows AI researchers to better understand the capabilities of each algorithm.

In general the four groupings we found seemed to each represent a different class of difficulty. Looking at figure 4, the leftmost group represents games that are easily beaten by most of the algorithms. This group is comprised of games that require only a few steps of look ahead in order to survive and they tend to offer positive rewards for each correct action.

On the right side of the figure are the hard games. This group is comprised of several different types of games. Games like Plants, which requires processing a very large map. Others, like Realportals, are challenging puzzles that require the player to look very far into the future for any positive feedback. These games share the common theme of having too large of search space for the controller to analyze. The one exception would be Defem. It features high stochasticity which causes the forward model to be unreliable.

The two middle groups feature games that require intelligent look ahead, but are still feasible for algorithms like MCTS and Iterative Width to search. From figure 5 it is clear that these two groups are similar, the main difference is how MCTS performs on them. It would be interesting to investigate further why basic MCTS does better at the third group while some of the more tailored algorithms do worse.

What this analysis shows is that our current algorithms' successes are very dependent on the size of the game. Faster processors will improve results, but at the same time, games can always be made larger. A game like CamelRace, where you simply have to hold right for a long period of time to win, is easy for a human to solve but difficult for a tree search algorithm to solve. There is still a need for new algorithms.

We were able to show that studying the features of games allow us to pick the best algorithm for different types of games. We found two features that allowed us to divide the games into four groups. Segregating the games into these clusters and using the optimal algorithm for each cluster did provide a performance improvement. Researchers can also design algorithms that take advantage of these features, such as taking more time to explore in games where you can't die.

This work suggests that algorithm portfolios can be a promising technique in GVGAI. There is still a need for better algorithms/creative hybrids that can play a lot of these games and we need to improve the features we are clustering on, but these initial clusterings already provide improved results performance over the state of the art. The data laid out in this paper provides a map for building a very robust algorithm portfolio for general video games.

References

Apperley, T. H. 2006. Genre and game studies: Toward a critical approach to video game genres. *Simulation & Gaming* 37(1):6–23.

Bauckhage, C.; Drachen, A.; and Sifa, R. 2015. Clustering game behavior data. *Computational Intelligence and AI in Games, IEEE Transactions on* 7(3):266–278.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2012. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*.

Bhowmick, S.; Eijkhout, V.; Freund, Y.; Fuentes, E.; and Keyes, D. 2006. Application of machine learning to the selection of sparse linear solvers. *Int. J. High Perf. Comput. Appl*.

Brodley, C. E. 1993. Addressing the selective superiority problem: Automatic algorithm/model class selection. In *Proceedings of the Tenth International Conference on Machine Learning*, 17–24.

Davidson, I., and Ravi, S. 2005. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In *Knowledge Discovery in Databases: PKDD 2005*. Springer. 59–70.

Demmel, J.; Dongarra, J.; Eijkhout, V.; Fuentes, E.; Petitet, A.; Vuduc, R.; Whaley, R. C.; and Yelick, K. 2005. Self-adapting linear algebra algorithms and software. *Proceedings of the IEEE* 93(2):293–312.

Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a video game description language.

Fang, X.; Chan, S. S.; and Nair, C. 2009. A lexical approach to classifying computer games.

Feldman, M.; Lewin-Eytan, L.; and Naor, J. S. 2012. Hedonic clustering games. In *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, 267–276. ACM.

Frydenberg, F.; Andersen, K. R.; Risi, S.; and Togelius, J. 2015. Investigating mcts modifications in general video game playing. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, 107–113. IEEE.

Fukunaga, A. S. 2008. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation* 16(1):31–61.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *ICAPS*. Citeseer.

Heintz, S., and Law, E. L.-C. 2015. The game genre map: A revised game classification. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, 175–184. ACM.

Jolliffe, I. 2002. *Principal component analysis*. Wiley Online Library.

Juul, J. 2005. Games telling stories? a brief note on games and narratives. 2001.

King, G., and Krzywinska, T. 2002. *Screenplay: cinema/videogames/interfaces*. Wallflower Press.

Kotthoff, L. 2012. Algorithm selection for combinatorial search problems: A survey. *arXiv preprint arXiv:1210.7959*.

Levine, J.; Congdon, C. B.; Ebner, M.; Kendall, G.; Lucas, S. M.; Miikkulainen, R.; Schaul, T.; and Thompson, T. 2013. General video game playing. *Dagstuhl Follow-Ups* 6.

Likas, A.; Vlassis, N.; and Verbeek, J. J. 2003. The global k-means clustering algorithm. *Pattern recognition* 36(2):451–461.

Mendes, A.; Nealen, A.; and Togelius, J. 2016. Hyper-heuristic general video game playing. In *Proceedings of Computational Intelligence and Games (CIG)*. IEEE.

Nielsen, T. S.; Barros, G. A.; Togelius, J.; and Nelson, M. J. 2015. Towards generating arcade game rules with vgdl. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, 185–192. IEEE.

OMahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and OSullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish Conference on Artificial Intelligence and Cognitive Science*, 210–216.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couetoux, A.; Lee, J.; Lim, C.; and Thompson, T. 2015a. The 2014 general video game playing competition. *Computational Intelligence and AI in Games, IEEE Transactions on* PP(99):1–1.

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015b. The 2014 general video game playing competition.

Perez Liebana, D.; Dieskau, J.; Hunermund, M.; Mostaghim, S.; and Lucas, S. 2015. Open loop search for general video game playing. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, 337–344. ACM.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; and Lucas, S. M. 2016. General video game ai: Competition, challenges and opportunities. In *Proceedings of AAAI*.

Perez, D.; Samothrakis, S.; and Lucas, S. 2014. Knowledge-based fast evolutionary mcts for general video game playing. In *2014 IEEE Conference on Computational Intelligence and Games*, 1–8.

Pulina, L., and Tacchella, A. 2009. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints* 14(1):80–116.

Reynolds, D. 2015. Gaussian mixture models. *Encyclopedia of Biometrics* 827–832.

Schatz, T. 1981. *Hollywood genres: Formulas, filmmaking, and the studio system*. McGraw-Hill Humanities/Social Sciences/Languages.

Wolf, M. J. 2002. Genre and the video game. *The medium of the video game* 113–134.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. Satzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research* 565–606.