Hyper-heuristic General Video Game Playing

Andre Mendes
Department of Computer Science
New York University
New York, New York 11021
andre.mendes@nyu.edu

Julian Togelius
Department of Computer Science
New York University
New York, New York 11021
julian@togelius.com

Andy Nealen
Department of Computer Science
New York University
New York, New York 11021
andy@nealen.net

Abstract—**In general video game playing, the challenge is to create agents that play unseen games proficiently. Stochastic tree search algorithms, like Monte Carlo Tree Search, perform relatively well on this task. However, performance is non-transitive: different agents perform best in different games, which means that there is not a single agent that is the best in all the games. Rather, some types of games are dominated by a few agents whereas other different agents dominate other types of games. Thus, it should be possible to construct a hyper-agent that selects from a portfolio, in which constituent sub-agents will play a new game best. Since there is no knowledge about the games, the agent needs to use available features to predict the most suitable algorithm. This work constructs such a hyper-agent using the General Video Game Playing Framework (GVGAI). The proposed method achieves promising results that show the applicability of hyper-heuristics in general video game playing and related tasks.**

I. Introduction

In order to be generally intelligent, an agent (artificial or natural) needs to be capable of behaving intelligently in a wide range of problems or environments. While one can use artificial intelligence methods to solve a particular problem, the resulting AI application is AI in only a narrow sense. To create general AI, one therefore needs to develop algorithms that are capable of solving a large number of problems. At least, this is the definition of intelligence and artificial intelligence implied by Legg and Hutter [1], variants of which are widely accepted in the Artificial General Intelligence community [2]. In other words, the AI application should not be dependent on the human designer re-tailoring the algorithm for each problem, otherwise we simply have a set of narrow AI solutions, with a human designer choosing which one to apply where.

This is also part of the reasoning behind the General Game Playing (GGP) [3] and General Video Game Playing (GVGAI) [4] competitions. In both, competitors submit game-playing agents, or controllers, that play any game adhering to a given interface. The developers of these controllers do not know at submission time which games their algorithms will be tested on and the winner is the controller which plays these *unseen* games best.

So far, the winners of the GGP and GVGAI competitions are mostly based on generic tree search algorithms. In particular, algorithms based on some version of Monte Carlo Tree Search (MCTS) [5] have been performing well. However, other very general algorithms, such as rolling horizon evolutionary planning [6] have also achieved good results.

It stands to reason that an algorithm that is capable of solving a large number of dissimilar problems will in some way be adaptive, such that it chooses which strategy to adopt depending on the problem. The analogy is that a human approaches different problems in various ways, depending on some familiarity with the type or initial experiences with solving these problems. In an artificial intelligence context, we could imagine an agent that includes several problem-solving algorithms or sub-agents, and chooses which one of them to use every time it encounters a new problem. The general concept of a method that selects sub-level methods to solve a problem seems to have been proposed independently in different lines of research and is couched in different terminologies. These ideas have been expressed in hyper-heuristics [7] [8], algorithm selection [9], meta-learning [10], [11] and ensembles [12].

In this paper, we 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, i.e. it uses set of trained instances to acquire information about controllers performance and create a selection model that generalizes well for new, unseen games. For clarity, we use the term hyper-agent instead of hyper-controller because it does not directly control the main character but it selects the best controller to do so.

Such an undertaking assumes that playing strength is non-transitive, i.e. different controllers excel at different games, and that there is some kind of regularity to which games are played best by which controllers. It also assumes that we can find game features that let us explore this regularity and use it to predict which controller would play an unknown game best. These assumptions are, as far as we can tell, hitherto untested for our target domain (general video game playing), making this paper the first attempt at using algorithm selection or hyper-heuristic techniques in general video game playing.

For the experiments in this paper, we use the GVGAI framework (the software used for the GVGAI competition), and a number of controllers developed for the competition. Based on a set of games, classifiers are trained to predict which controller excels at a given game. Then a hyper-agent uses the best classifier in games it has never been trained on. The final goal is to choose the best controllers to play each game in real time.

The paper is structured as follows. The next section describes general video game playing and the GVGAI framework as well as some principles of hyper-heuristics and algorithm selection. Section III describes our methodology: which controllers we used as constituent controllers for our hyper-heuristic agent, how we extracted features to use and collected data on controller performance, and train classifiers to predict the best controllers. Section IV describes the results of using these classifiers in a hyper-heuristic agent, as well as an attempt to use the cluster analysis to understand the game space induced by the selected game features.

II. Background

Hyper-heuristics are methods or learning mechanisms for selecting or generating heuristics to solve computational search problems [13]. To classify these methods two components are considered: the nature of the heuristics' search space, and the different sources of feedback information. According to the nature of the search space, the methods can be defined as heuristic selection, when used for selecting existing heuristics, and heuristic generation, when the goal is to use existing heuristics to generate new ones [7], [9]. By this definition, heuristic selection has a large overlap with the algorithm selection problem, where the goal is to select the best algorithm to solve different instances of the problem without modifying the algorithm [14]. In this work, we explore the literature from both fields and although we prefer to use the term hyper-agent, we acknowledge that it can also be considered as an algorithm selection approach.

A. Hyper-heuristics

With regard to the feedback information, a hyper-heuristic is considered a learning algorithm if it uses feedback from the search process to improve its performance. This feedback can be online and offline. It is online when the learning happens while the algorithm is solving an instance of a problem. It is offline if it uses information in the form of rules or programs, from a set of training instances, that will hopefully generalize well for unseen instances [13][15]. There are also hybrid methodologies that combine online and offline feedback [16], and heuristic selection with heuristic generation [17] [18].

Three requirements can be considered when defining a hyper-heuristic approach. It has to manage a set of low-level heuristics, it searches for a good method rather than for a good solution, and most importantly, it uses only limited problem-specific information in order to generalize well for other applications [19]. The method proposed in this paper fulfills all three requirements and can be classified as a selection hyper-heuristic that uses offline feedback information.

A similar method has been proposed to explore patterns of regularities of two heuristics for constraint satisfaction. The collected performance of heuristics is used to generate a heuristic that selects others in the constructive process to produce a solution [20]. Another study uses feature extraction and hyper-heuristics to improve the problem state representation of irregular packing problems [21].

Most of the applications of hyper-heuristics approaches have been focused on problems in domains such as production scheduling [22] and education timetabling [23]. In the games domain, Li and Kendall use a heuristic selection mechanism integrates a number of existing heuristics for specialized strategic games into an automated game player [24]. Elyasaf et al. evolve heuristics to guide staged deepening search to develop top-notch solvers for the hard game of FreeCell [25]. Using a evolutionary algorithm to pick from a set a low-level heuristics, Salcedo-Sanz et al. proposed a hyper-heuristic method for the puzzle-game Jawbreaker [26] and Benbassat et al. presents two types of approaches, ”minimalist” (less human knowledge in the setup) and ”maximalist” (using human knowledge in the setup) to evolve game strategies for board games [27].

Although in the same domain, the methods proposed differ from our problem in significant ways such as: the type of games, the type of the heuristics' search space and feedback information, and the information available to the hyper-heuristic about the game and the low-level heuristics.

B. Algorithm Selection

One influential taxonomy of algorithm selection methods [9] 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 [28], [29] if the algorithms are defined a priori and never change or dynamic if the portfolio changes while solving the problem [30]. From the portfolio, methods can select the single best algorithm or allocate times slots to combine results from different algorithms. Concerning to when they select, the methods can pick before the solving of the actual problem starts or while the problem is being solved [31]. Another important step is how this selection is made. The decision involves, for example, analyzing accuracy, computational cost and time and even number of low-level heuristics to manage [18]. Finally, there is also an essential step that concerns about finding information to help the selection, such as feature selection and extraction [32],[28] 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 the problem is being solved. Our selection method uses machine learning models based on the previous performance of our algorithms (controllers) in each instance (games). To train these models we manually select features available at the start of each game.

Algorithm selection has been applied to domains such as linear algebra [33], linear systems [34] and specially to combinatorial search problems [31], [29]. The use of algorithm selection in games seems to be restricted to game theory [35] and we could not find any applications in video games in the literature.

C. General video game playing

In the past decade, video games have become increasingly popular as an AI benchmarks, as they require a rich repertoire of cognitive capabilities for humans to play well, but can be simulated simply and cheaply on a computer. A number of recent AI competitions have been based on different kinds of video games. The GVGAI competition was created mainly to counteract the problem that these competitions allow participants to tailor their submissions strongly to a particular game. Instead, game-playing controllers submitted to the GVGAI competition are pitted against a number of *unseen* games [4]. This separates it from e.g. the Arcade Learning Environment, where controllers are trained to play a number of well-known games [36]. 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 [37] 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 [38].

III. Methods

A. 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 [39]. Using level description and game description, VGDL can describe a wide variety of video games, including approximate versions of Space Invaders, Sokoban, Pong, Pac-Man, and Mario Bros.

In this work, we use 41 games available in the GVGAI framework as of December 2015. 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.

B. Game-playing controllers

Since the hyper-agent selects the best low-level controllers, its performance is directly related to the performance of the controllers available in its portfolio. In light of that, it seems reasonable to think that the portfolio should be composed of the best controllers. However, based on previous work [40] and on our analysis of the competition, we noticed that the overall champions are able to perform well in most of the games but fail badly in playing some specific ones. Meanwhile other overall low-performance controllers are able to win in these specific games. Therefore, we want a balanced portfolio composed of complimentary controllers with different strengths that, together, win a broader variety of games. Another important characteristic is to select a number of different controllers that guarantees variability but is not so large that makes hard for the model to choose from [28].

Within these considerations, we use a static portfolio composed of seven controllers. Three of them are high-performing algorithms that were among the best places in 2014 and 2015 competition. The other four are standard algorithms created by the developers of the framework with mediocre overall performance but with good results in specific games where the high-performing algorithms fail.

The controller *adrienctx* (ACT), created by Adrien Couëtoux and first place in 2014, uses the algorithm Open Loop Expectimax Tree Search (OLETS) [4], which is inspired by the Hierarchical Open-Loop Optimistic Planning algorithm (HOLOP) [41]. It uses a method called Open Loop Expectimax (OLE) with uses nodes containing a reward function that can store the empirical average reward obtained from simulations that exited in the node and the maximum of reward functions values from the nodes children.

The second place in 2014, created by Jerry Lee, is called *JinJerry* (JJ). JinJerry is based on a MCTS with a selection strategy similar to the one used in [42]. In every game cycle, It creates a one-level tree with the current game state as the root node and the one-action-ahead states of all actions as the leaf nodes. After that, it needs to evaluate the immediate and potential score of all the leaf nodes so that it can use a scoring heuristic to evaluate the states. The action selection strategy is to select the action with a safe state and a high score. If the random actions lead to a state where the game is over, the potential score is not considered. Finally, the potential score will replace the immediate score if it has a higher score.

The champion algorithm for 2015 is YOLOBOT (YB) created by Tobias Joppen, Nils Schroeder and Miriam Moneke. It combines methods like breadth first search and MCTS to analyze the current state and simulations of future states reachable through one-action-ahead. It also creates an observation list of all the sprites reachable in the state and defines the most interesting target based on a function that analyzes empirical values from the observations.

There are also four sample controllers. Sample One Step Look Ahead (SOS) uses an simple heuristic function to evaluate the states reached within one move from the current state and select the action with highest reward. The Genetic Algorithm controller (GA) implements an online (rolling horizon) genetic algorithm, where each individual represents a sequence of actions and its fitness is evaluated using an heuristic function. Small populations are evolved in each game step and the move returned is the first action of the best individual found.

The last two controllers, the sample MCTS (MCTS) and OL-MCTS (OLMCTS), use a similar vanilla Monte Carlo Tree Search implementation [5]. The sample MCTS is extended with common enhancements such as using an Upper Confidence Bound for trees selection and a random expansion. The algorithm evaluates the states by giving a high reward for a won game and a negative reward for a lost game. If the final state is not reached, it uses the score achieved. OL-

MCTS player is a sample MCTS that uses an Open Loop implementation, focusing more on information gained from the current state of the game [4].

C. Non-Transitive nature

To evaluate the non-transitive nature of the controllers, we played all the games using the procedures explained in Section III-H. Table I shows the sum of victories that each controller achieved for five levels in each game and Table II shows the normalized average scores. The intensity of the cells is proportional to the performance of the controller, where higher and lower values are respectively presented in lighter and darker cells. These results show that our portfolio is composed of complimentary controllers that can excel in a wide variety games.

D. Game Features

Features have a direct impact in the performance of our classifier. Although the game in the GVGAI platform is unknown to the controller, the state observation of the game provides information about various elements, such as number and type of NPCs, type of resources, dimensions of the map, action set, and types and number of sprites. We analyzed the classes available in the general state observation of all games and defined 14 features that are presented in each of them and we believe would represent different types of games. Table III shows the features, the group they belong to, and a description of each of them.

The first group is the resources available in the game. We believe that theses features provide important information about changes in the rules of the game, such as when pacman eats a pill and can eat enemies. The second group refers to Non-Player Characters (NPCs) and it provides certain information about the type of game, for example, if the agent has enemies or it is alone (like in a puzzle game). Dimensions is used to give an idea about the space the agent has in the game. Although elements such as block size seems like an unusual choice, Figure 2 shows that it provides useful information in the decision tree model.

The Sprites groups provide information mainly about movable things in the game and how it affects the agent. Portals, similarly to resources, is a good indication of changes in the game. For example, a game with portals can offer more changes because new enemies or resources can come out of them. Finally, the actions group gives very important information about what the user can do in the game, such as move up or down and attack enemies.

E. Data collection

For feature extraction, we analyzed different games in their start (game tick = 0) and after a period of time (game tick > 0). Except for number of NPCs, all of our features are static, i.e. they remain the same during gameplay. We then created a collector bot that saves the features at the initial state of each level in each game.

In our approach, the labels are the best controllers for each level of the games. In order to compare the these controllers, we ran the experiments described in Section III-H. Using the features extracted and the average performance of the controllers, we created a dataset composed of 205 instances (41 games *times* 5 levels) with 14 features and 205 labels (best controller for each instance).

F. Classification

Using the dataset from the data collection, we utilize multiclass supervised learning to create a model for algorithm selection using methods implemented in the Weka machine learning software [43]. Similar to previous work in the field [44], [45], we choose algorithms that could achieve good accuracy, such as Support Vector Machines (SVM, using libSVM implementation [46]) and Multi-Layer Perceptrons (MLP), and methods that could also report an understandable strategy for the classification, like Decision trees (J48 implementation in Weka) and logistic regression (LR), like in previous works.

Our labels for the classification are the ones obtained during data collection described in section III-E. To prevent overfitting due to the small size of our dataset, we perform two types of cross validation. First we use a normal 10-fold cross-validation (CV-1) where in each round the data is shuffled and partitioned in training and test subsets. Then validation results are averaged over the rounds. In the second method (CV-2), to ensure that the model would be tested with truly *unseen* data, we manually divided the dataset into 10 different subsets, where each test subset is composed of four different games with five levels. We then perform the 10-fold cross validation and in each round the model was trained in 37 games (37*5=185 levels/instances) and tested in 4 (4*20 levels/instances) completely unseen games.

G. Online playing

The hyper-agent works following a sequence of three steps: feature collection, algorithm classification and algorithm selection (Figure 1). In feature collection, the hyper-agent uses the state observation of the game to collect the selected features. Then it normalizes them using the same filter as in the offline training. This creates a dataset similar to the one used for training the model. In algorithm classification, it uses the trained model to predict the best controller to play the current game. Finally in algorithm selection, it uses the output decision and loads the best controller. This controller will play the rest of the game, returning actions using its own defined strategy.

H. Experiments

To collect data and compare the proposed hyper-agent with those submitted to the GVGAI competition, we use the same software, scoring method and ranking criteria [4]. All the levels for each game are played five times by the seven controllers in our portfolio and by two variations of hyper-agents. This represents a total of 1,025 game plays by each controller/agent and an overall total of almost 10,000 game plays. The experiments were performed locally using the framework available on the competition website.

Three measures were applied to the experiments: victories, score and time. Since it is more important to win the game

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21
MCTS	25	1	23	0	5	11	4	6	12	2	1	0	0	22	1	3	0	12	23	1	1
OLMCTS	25	1	23	0	5	16	1	7	7	4	2	0	4	20	0	5	0	11	25	3	0
GA	25	1	23	0	6	13	3	6	12	6	0	0	7	19	0	3	0	11	21	0	2
SOS	25	1	23	0	1	14	4	6	13	1	1	0	1	21	0	3	0	14	20	0	2
YB	25	7	25	11	24	22	21	21	13	25	25	1	25	25	6	24	2	1	24	12	20
ACTX	25	15	25	14	6	10	25	4	10	25	5	0	5	25	0	23	14	18	25	1	3
JJ	12	0	25	8	20	25	0	3	10	13	8	0	10	23	2	17	0	9	6	1	1

	G22	G23	G24	G25	G26	G27	G28	G29	G30	G31	G32	G33	G34	G35	G36	G37	G38	G39	G40	G41	Total
MCTS	0	2	3	1	3	0	19	2	0	9	14	1	17	0	0	0	6	25	1	2	258
OLMCTS	0	6	3	0	4	0	14	0	0	6	19	0	10	0	0	0	5	25	0	1	252
GA	0	2	1	0	4	0	19	2	0	5	15	2	16	0	0	0	5	25	5	3	262
SOS	0	3	3	0	4	0	18	4	0	5	14	2	13	0	0	0	5	25	2	3	251
YB	0	9	10	25	20	25	24	25	25	24	10	1	23	0	14	10	10	25	10	5	654
ACTX	0	22	0	2	5	13	4	25	0	6	25	0	22	0	0	18	0	25	2	6	453
JJ	0	0	3	1	6	5	1	17	0	4	5	7	25	0	0	0	25	25	5	5	327

TABLE I: Non-Transitivity for victories - Five levels for each game are played five times and we sum the win rate for each agent. The intensity of the cells is proportional to the results and they indicate that different agents excel at different games, showing the non-transitive behavior of our set of agents with regarding to victories.

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15	G16	G17	G18	G19	G20	G21
MCTS	0.23	0.17	0.61	0.41	0.18	0.27	0.16	0.2	0.3	0.14	0.04	0.77	0.06	0.08	0.25	0.22	0.49	0.11	0.14	0.1	0.08
OLMCTS	0.32	0.19	0.48	0.46	0.25	0.35	0.04	0.37	0.02	0.12	0.08	0.66	0.39	0.09	0.23	0.14	0.62	0.1	0.31	0.31	0
GA	0.24	0.18	0.55	0.45	0.2	0.27	0.12	0.25	0.28	0.34	0	0.47	0.34	0.09	0.16	0.15	0.52	0.14	0.14	0.05	0.14
SOS	0.13	0.17	0.43	0.02	0	0.3	0.16	0.28	0.32	0.07	0.04	0.33	0.04	0.11	0.18	0.21	0.39	0.21	0.14	0.05	0.14
YB	1	0.34	0.1	0.86	1	0.77	0.84	1	0.45	0.21	1	1	1	0.57	0.97	0.68	0.08	0.2	0.86	0.81	1
ACTX	0.14	0.92	0.11	0.94	0.45	0.25	1	0.2	0.99	1	0.2	0.53	0.51	1	0.5	0.76	1	0.78	0.67	0.06	0.24
JJ	0.58	0.17	0.14	0.92	0.82	0.87	0	0.2	0.73	0.57	0.32	0.58	0.23	0.49	0.81	0.53	0.7	0.64	0.11	0.2	0.16

	G22	G23	G24	G25	G26	G27	G28	G29	G30	G31	G32	G33	G34	G35	G36	G37	G38	G39	G40	G41	Total
MCTS	0.28	0.23	0.45	0.17	0.03	0.03	0.19	0.11	0.06	0.36	0.45	0.47	0.45	0.07	0.08	0.55	0.87	0	0.1	0.63	0.1
OLMCTS	0.22	0.19	0.31	0.09	0.14	0.06	0.21	0.03	0.12	0.24	0.28	0.55	0.16	0.05	0.04	0.52	0.69	0	0.14	0.28	0.07
GA	0.35	0.24	0.1	0.04	0.14	0.06	0.22	0.11	0.09	0.2	0.45	0.58	0.69	0.01	0.01	0.52	0.86	0	0.31	0.46	0.1
SOS	0.21	0.22	0.3	0.1	0.13	0.03	0.29	0.16	0.13	0.2	0.46	0.45	0.59	0.04	0.03	0	0.86	0	0.23	0.2	0
YB	0	0.34	0.7	1	1	1	1	1	1	1	0.47	0	0.32	0.15	0.8	0.77	0.92	0.79	1	0.98	1
ACTX	0.41	1	0.2	0.22	0.49	0.52	0.16	1	0.95	0.24	0.05	0.07	0.49	0.8	0.33	1	0.12	0.91	0.23	0.7	0.66
JJ	0.5	0.1	0.48	0.21	0.42	0.19	0.03	0.68	0.95	0.08	1	0.98	0.87	0	0.29	0.57	1	0	0.42	0.71	0.53

TABLE II: Non-Transitivity for scores - In the GVGAI framework, different games have different scores. In order to compare them, results are normalized so that the algorithm with the highest score receives 1, the lowest receives 0 and the others are ranked between those values. Similar to the victories, algorithms also present non-transitive behavior for scores.

Group	Feature	Description
Resources	gameHasResources	If there are resources in the game
	avatarHasResources	If the avatar has resources
	nTypeResourcesGame	Number of types of resources
	nTypeResourcesAvatar	Resources that the controller has
NPC	nNPC	Number of NPCs in the game
	nTypeNPC	Number of types of NPCs
Dimensions	Area	The area of the game
	blockSize	The number of pixels of a block
Sprites	nTypeImmovableSprites	Number of types of immovable Sprites
	nTypeMovableSprites	Number of types of movable Sprites
Portals	hasPortals	If the game has portals
	nTypePortals	Number of types of portals
Actions	MoveVertically	If the controller can move vertically
	CanShoot	If the controller can attack enemies

TABLE III: Selected features used to represent unknown games.

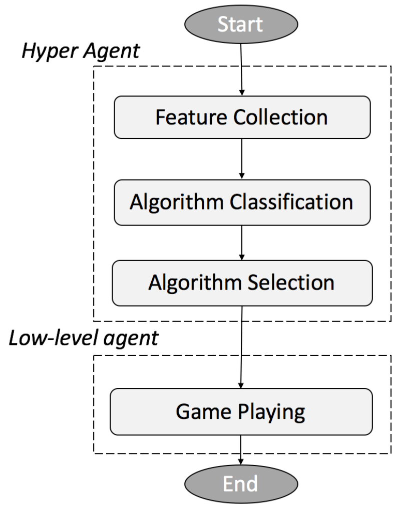

Fig. 1: The framework with steps for online playing.

than fail with a high score, the GVGAI competition weighs the number of victories higher than the achieved score. The time measure is used only as a second tie-breaker because high score victories are weighted higher than fast wins.

IV. Results

From the results presented in Table IV, we selected SVM and J48 to be selection models for the hyper-agent. SVM achieved the best accuracy in both validations and J48 was the second best in the second validation, which is the most important. Besides, it provides an understandable decision tree (Figure 2) that justifies its selection strategy. Although MLP had good results in the first validation, the second validation shows that model overfits the dataset an its performance is inferior to ZeroR algorithm that just selects the majority class in the dataset.

A. Controller selection at start

With trained models from SVM and J48, we created two variations of the hyper-agents, HA-SVM and HA-J48. They both use the framework for online playing, but differ in the model selected to perform algorithm classification.

To show that the hyper-agents significantly differ from the other algorithms, we compare their distributions through a

	Accuracy	
Algorithms	CV-1	CV-2
SVM	0.7125	**0.6373**
MLP	0.6976	**0.4801**
J48	0.6878	**0.5936**
LR	0.6023	**0.5201**
ZeroR	0.5042	**0.5042**

TABLE IV: Classification accuracy for the machine learning algorithms using normal cross-validation (CV-1) and domain-aware cross-validation (CV-2).

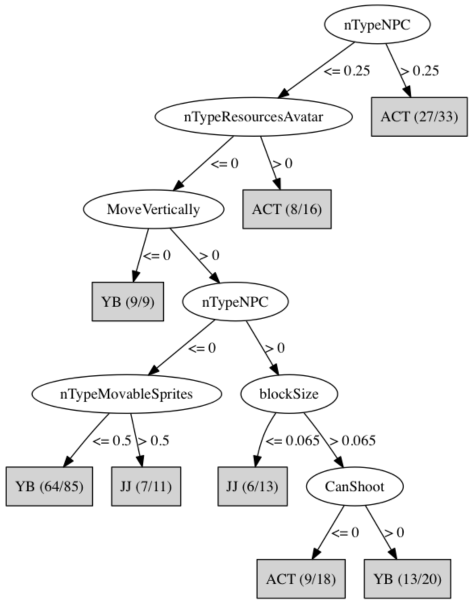

Fig. 2: J48 decision tree using features with high information gain. For each leaf, it shows the controller selected and the number of correctly classified instances out of the total instances reaching the leaf.

statistical test using the a Mann-Whitney-Wilcoxon (MWW) approach [47]. Table V shows the p-values for victories and scores for the hyper-agents and the competition controllers. If the p-value is small (p<0.05), it is possible to reject the null hypothesis, which in this case says that difference is due to random sampling, and conclude instead that the populations are indeed distinct. The victories and scores for HA-J48, and victories for HA-SVM, are proven to be significantly different from the others. However, although HA-SVM achieved higher scores than YB, the null hypothesis cannot be entirely discarded because the p-value between the two distributions is 0.08.

In Table VI we show general results for the controllers and an illustration that visually shows the superior performance of the hyper-agents in victories, scores and time, over standard and competition algorithms. To compare the controllers in each game, we sum the wins for each level as in Table I and average the values for score and time, as in Table II. We consider that a controller dominates a game if it wins more than 3 levels of that game. In Table VII, the total results show that HA-SVM dominates 28 games and HA-J48 25, achieving first and second place respectively. When not considering the hyper-agents, YOLOBOT is the best in 21 games, adrienctx in 14 and JinJerry in 6. Standard algorithms performed well in some levels, but none of them dominated any game.

Victories	YB	ACT	JJ	Scores	YB	ACT	JJ
HA-SVM	0.014	0.000	0.000	HA-SVM	0.084	0.000	0.000
HA-J48	0.035	0.000	0.000	HA-J48	0.023	0.000	0.000

TABLE V: p-values from comparing the victories and scores distributions for hyper and low agents using MWW test. If p-value < 0.05, the null hypothesis can be discarded.

	wins	scores	time
HA-SVM	721	0.95	1.00
HA-J48	709	1.00	0.89
YB	654	0.78	0.87
ACTX	453	0.51	0.72
JJ	327	0.43	0.62
GA	262	0.09	0.00
MCTS	258	0.09	0.03
OLMCTS	252	0.06	0.04
SOS	251	0.00	0.05

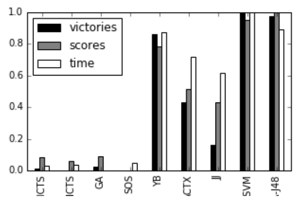

TABLE VI: Left - Overall results with total number of victories and normalized values for score and time. Time is inversely normalize so that values closer to 1 means faster controllers. Right - Visual representation with all metrics normalized.

V. Discussion

The proposed algorithm selection approach uses features to predict the best algorithm for an unseen game. We selected features that were available in the framework and exposed to the controller through the API.

As in many machine learning problems, having good features with high information gain plays an important role in the performance of the classifiers. To better understand our agent, we used a J48 algorithm to see the decisions that the agent makes based on the features available. As expected, features like CanShoot and MoveVertically are present in the selection but with less importance than we anticipated. Since they have good information to differentiate types of games (for humans at least), we thought they would appear in the first or second level of the tree.

With regard to the controllers, the classifier focus on the three "safe" high performance controllers, which suggests high exploitation. The downside is that other controllers in our portfolio are not being used. To balance the trade-off and improve explorations, we increase the tree depth. Although the new tree shows more different controllers being selected, neither the accuracy of the model or the performance of the agent is significantly improved, whereas the complexity of the tree is.

We believe that is happens because if the model selects one of the three high performance controllers, it has a good chance to perform well even if it does not pick the best. On the other hand, if it wrongly selects a "risk" low performance controller, the likelihood of failing in the game is much higher.

Games	YB	ACT	JJ	HA-SVM	HA-J48
aliens (G1)	✓			✓	✓
boulderdash (G2)		✓		✓	✓
butterflies (G3)				✓	✓
chase (G4)		X			
frogs (G5)	✓			✓	✓
missilecommand (G6)			X		
portals (G7)		✓		✓	✓
sokoban (G8)	X				
survivezombies (G9)	X				
zelda (G10)		X			
camelrace (G11)	✓			✓	✓
digdug (G12)	✓			✓	✓
firestorms (G13)	✓			✓	✓
infection (G14)		✓		✓	✓
firecaster (G15)	X				
overload (G16)	✓			✓	✓
pacman (G17)		✓		✓	✓
seaquest (G18)		✓		✓	
whackamole (G19)		X			
eggomania (G20)	✓			✓	
bait (G21)	✓			✓	✓
boloadventures (G22)			X		
boulderchase (G23)		X			
brainman (G24)	✓			✓	
catapults (G25)	X				
chipschallenge (G26)	✓			✓	
escape (G27)	✓			✓	✓
jaws (G28)	✓			✓	✓
labyrinth (G29)	✓			✓	✓
lemmings (G30)	✓			✓	✓
modality (G31)	✓			✓	✓
painter (G32)		✓		✓	✓
plants (G33)			✓	✓	✓
plaqueattack (G34)			✓	✓	✓
realportals (G35)		X			
realsokoban (G36)	✓			✓	✓
roguelike (G37)		✓		✓	✓
solarfox (G38)			✓	✓	✓
surround (G39)		X			
thecitadel (G40)	✓			✓	✓
zenpuzzle (G41)		✓			✓
Total	**21**	**14**	**6**	**28**	**25**

TABLE VII: Each row is a game. In the left three columns, the best agent is assigned with a check mark if it has been selected by one of the hyper-agents, or with a X otherwise. In the right two columns, the check mark indicates that hyper-agent selected the best low-agent for the game.

This trade-off is important to improve the performance our agent because the main advantage of algorithm selection is to use of the strengths of different algorithms in the portfolio. To achieve this, we need to improve the performance of the classifier to select the low performance controllers with higher accuracy.

By extracting and selecting better features, it is likely that the performance of the classifiers could be increased by improving the classification accuracy of the model. In particular, it might be possible to define features based on the first few time frames of game play, including observations on the behavior of moving sprites, spawning sprites etc. It could also be possible to define features that rely on longer observations of the game, and which change during gameplay, making it possible to dynamically switch between controllers. The construction of features that are not dependent on the API of the GVGAI software will also increase the generality of the method. Another important is step is to incorporate a feature selector system to automatically extract different features and train the model more dynamically.

Another important step for this project is to increase the number of games used. While we use cross-validation and statistical significance testing to validate the superior performance of the hyper-agents over its portfolio, having more games would help us to improve the strategy and understand the generality of our approach. That would also make it easier to compare our method with other standard-of-art methods in algorithm selection that uses larger datasets. One possible solution for this, is to *generate* games to train and test on. Initial attempts to generate games in the GVGAI framework are promising, and it is at least possible to automatically generate variations of existing ones [48].

VI. Conclusions

This paper has presented an algorithm selection approach to general video game playing in the GVGAI framework. Based on the observation that performance in this domain is non-transitive, a hyper-agent based on several well-playing controllers was constructed. A number of features available to the controllers were extracted and classification models were trained to predict the best controller on each game. These models were used in the hyper-agent to select the best controller to play unknown games. Testing the hyper-agent on unseen games showed that it significantly outperformed the winners of the 2014 and 2015 competitions. These encouraging results suggest that the use of hyper-heuristics and algorithm selection have an important role in general video game playing and related tasks and there are many possibilities for improvement.

For future work, we intent to develop the feature selection step, improve the size of the dataset potentially generating more games automatically and we want to continue exploring the application of hyper-heuristics in general video game playing. We believe that methods with online learning can perform well in this type of problem due to the many changes that happen during game play. We also want to investigate the use of heuristic generation in this the domain to not only select, but also create high performance algorithms.

Acknowledgment

Andre Mendes acknowledges financial support from CNPq Scholarship and Science Without Borders program.

References

[1] S. Legg and M. Hutter, "Universal intelligence: A definition of machine intelligence," *Minds and Machines*, vol. 17, no. 4, pp. 391–444, 2007.

[2] B. Goertzel and C. Pennachin, *Artificial general intelligence*. Springer, 2007, vol. 2.

[3] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the aaai competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.

[4] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C. Lim, and T. Thompson, "The 2014 general video game playing competition," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[5] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton *et al.*, "A survey of monte carlo tree search methods," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

[6] D. Perez, S. Samothrakis, S. Lucas, and P. Rohlfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013, pp. 351–358.

[7] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[8] P. Cowling, G. Kendall, and E. Soubeiga, *A Hyperheuristic Approach to Scheduling a Sales Summit*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 176–190.

[9] L. Kotthoff, "Algorithm selection for combinatorial search problems: A survey," *AI Magazine*, vol. 35, no. 3, pp. 48–60, 2014.

[10] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial Intelligence Review*, vol. 18, no. 2, pp. 77–95, 2002.

[11] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 6, 2009.

[12] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple classifier systems*. Springer, 2000, pp. 1–15.

[13] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[14] J. R. Rice, "The algorithm selection problem," 1975.

[15] P. Ross, S. Schulenburg, J. G. Marín-Blázquez, and E. Hart, "Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 942–948. [Online]. Available: http://dl.acm.org/citation.cfm?id=646205.682617

[16] P. Garrido and M. C. Riff, "Dvrp: A hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *Journal of Heuristics*, vol. 16, no. 6, pp. 795–834, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1007/s10732-010-9126-2

[17] J. Maturana, F. Lardeux, and F. Saubion, "Autonomous operator management for evolutionary algorithms," *Journal of Heuristics*, vol. 16, no. 6, pp. 881–909, 2010. [Online]. Available: http://dx.doi.org/10.1007/s10732-010-9125-3

[18] S. Remde, P. Cowling, K. Dahal, N. Colledge, and E. Selensky, "An empirical study of hyperheuristics for managing very large sets of low level heuristics," *Journal of the Operational Research Society*, vol. 63, no. 3, pp. 392–405, 2012. [Online]. Available: http://dx.doi.org/10.1057/jors.2011.48

[19] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," in *Adaptive and multilevel metaheuristics*. Springer, 2008, pp. 3–29.

[20] J. C. Ortiz-Bayliss, E. Özcan, A. J. Parkes, and H. Terashima-Marín, "Mapping the performance of heuristics for constraint satisfaction," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[21] E. López-Camacho, H. Terashima-Marín, P. Ross, and M. Valenzuela-Rendón, "Problem-state representations in a hyper-heuristic approach for the 2d irregular bpp," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 297–298.

[22] G. Ochoa, J. A. Vázquez-Rodríguez, S. Petrovic, and E. Burke, "Dispatching rules for production scheduling: a hyper-heuristic landscape analysis," in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*. IEEE, 2009, pp. 1873–1880.

[23] N. R. Sabar, M. Ayob, R. Qu, and G. Kendall, "A graph coloring constructive hyper-heuristic for examination timetabling problems," *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, 2012.

[24] J. LI and G. Kendall, "A hyper-heuristic methodology to generate adaptivestrategies for games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2015.

[25] A. Elyasaf, A. Hauptman, and M. Sipper, "Evolutionary design of freecell solvers," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 4, pp. 270–281, Dec 2012.

[26] S. Salcedo-Sanz, J. Matías-Román, S. Jiménez-Fernández, A. Portilla-Figueras, and L. Cuadra, "An evolutionary-based hyper-heuristic approach for the jawbreaker puzzle," *Applied intelligence*, vol. 40, no. 3, pp. 404–414, 2014.

[27] A. Benbassat, A. Elyasaf, and M. Sipper, "More or less? two approaches to evolving game-playing strategies," in *Genetic Programming Theory and Practice X*. Springer, 2013, pp. 171–185.

[28] L. Pulina and A. Tacchella, "A self-adaptive multi-engine solver for quantified boolean formulas," *Constraints*, vol. 14, no. 1, pp. 80–116, 2009.

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

[30] A. S. Fukunaga, "Automated discovery of local search heuristics for satisfiability testing," *Evolutionary Computation*, vol. 16, no. 1, pp. 31–61, 2008.

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

[32] A. Gerevini, A. Saetti, and M. Vallati, "An automatically configurable portfolio-based planner with macro-actions: Pbp." in *ICAPS*. Citeseer, 2009.

[33] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. C. Whaley, and K. Yelick, "Self-adapting linear algebra algorithms and software," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 293–312, 2005.

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

[35] A. S. Fukunaga, "Genetic algorithm portfolios," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 2. IEEE, 2000, pp. 1304–1311.

[36] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Int. Res.*, vol. 47, no. 1, pp. 253–279, May 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2566972.2566979

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

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

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

[40] L. Hong and S. E. Page, "Groups of diverse problem solvers can outperform groups of high-ability problem solvers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 46, pp. 16 385–16 389, 2004.

[41] A. Weinstein and M. L. Littman, "Bandit-based planning and learning in continuous-action markov decision processes." in *ICAPS*, 2012.

[42] D. Robles and S. Lucas, "A simple tree search method for playing ms. pac-man," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, Sept 2009, pp. 249–255.

[43] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[44] H. Guo and W. H. Hsu, "A learning-based algorithm selection meta-reasoner for the real-time mpe problem," in *AI 2004: Advances in Artificial Intelligence*. Springer, 2004, pp. 307–318.

[45] L. Pulina and A. Tacchella, "A multi-engine solver for quantified boolean formulas," in *Principles and Practice of Constraint Programming–CP 2007*. Springer, 2007, pp. 574–589.

[46] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[47] M. P. Fay and M. A. Proschan, "Wilcoxon-mann-whitney or t-test? on assumptions for hypothesis tests and multiple interpretations of decision rules," *Statistics surveys*, vol. 4, p. 1, 2010.

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