Evolving Artificial General Intelligence for Video Controllers

Itay Azaria, Achiya Elyasaf, and Moshe Sipper

Abstract The General Video Game Playing Competition (GVGAI) defines a challenge of creating controllers for general video game playing, a testbed—as it were—for examining the issue of *artificial general intelligence*. We develop herein a game controller that mimics human-learning behavior, focusing on the ability to generalize from experience and diminish learning time as *new* games present themselves. We use genetic programming to evolve hyper heuristic-based general players, our results showing the effectiveness of evolution in meeting the generality challenge.

Key words: Genetic Algorithms, Genetic Programing, Hyper-Heuristic

1 Introduction

Imagine playing a video game for the first time. You probably spend some time developing an understanding of the effects of your actions, identifying the various hostile or friendly non-player characters (NPCs), figuring out which items to avoid and which to collect, and so forth. In other words, you familiarize yourself with the goals of the game. A game can have many goals, such as collecting artifacts, killing NPCs, reaching a certain place in the game world, etc. There may be more complicated goals that are a combination of others, for example, collecting a key and then reaching the exit portal. Some of these goals lead to victory, some to award points, and some to both. Ultimately, we want to complete as many of these goals as possible before attempting the "end" goal that leads to victory and finishing the game, thus maximizing our score.

As you advance in the game you no longer have to focus on understanding it. Instead, you can focus on developing more-advanced strategies or predicting the immediate future of the game and estimating whether or not the strategies you have in mind are feasible.

Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel.

When you move to other levels, games, or both, your previous experience helps to improve the learning curve. For example, you might not yet recognize the good and bad resources but you already understand the concept of resources and non-player characters and the importance of keeping a certain distance from them. You will have to spend some time learning the specific new elements, or you might have to completely re-evaluate some insights—perhaps a wall is no longer an object you can't get through?—but you can rely on your previous experience to shorten the learning process.

Our goal in this work is to develop an artificial intelligence *game controller* (or *player*) that mimics this behavior, specifically focusing on the ability to generalize from experience and diminish learning time as new games present themselves.

The General Video Game Playing Competition (Perez et al (2015)) proposes the challenge of creating a controller for general video game playing, as a testbed for examining the issue of artificial general intelligence.

The GVGAI competition provides a framework that comprises a set of games, which differ in various aspects, including: winning conditions, scoring mechanism, sprite types, and available actions. The world the agent plays in is fully observable and a forward model is provided. However, the games are stochastic and no information is provided regarding winning conditions or interactions between different elements in the world. It is up to the agent to either infer such information or otherwise search in the state space.

Our goal is *not* to win the GVGAI competition (even though our results turned out to be excellent), but to use the offered framework as a convenient, extant benchmark for our work. This decision stems partly from a technical reason (involving unsupported multi-threading, as elaborated in Section 3.4) that prevents us from competing online directly, and also from a desire to emphasize the *general* part of AGI, whereupon we wished to avoid specificity as much as possible.

The chapter is organized as follows: In the next section we examine previous and related work. In Section 3 we describe our method. Finally, we end with concluding remarks and future work in Section 5.

2 Previous Work

2.1 Automated planning and MDP

Automated planning is a field of research in which generalized problem solvers (known as *planning systems* or *planners*) are constructed and tested across various benchmark puzzle domains.

An MDP (Markov Decision Process) is a model of an agent interacting synchronously with some given "world". The agent takes as input the state of the world and generates actions as output, which themselves affect the state of the world. While there is uncertainty in the MDP framework regarding the outcome of the agent's actions, the agent's current state is known. The description of MDP problems usually includes: 1) a transition function, assigning probabilities of reaching each state after performing a specific action in a given state; and 2) a reward function, describing the immediate reward for performing an action at a certain state Kaelbling et al (1998).

The problem we face is similar in that the world can be represented as a set of states, and the agent must choose from a set of actions according to the current state. However, we have neither the transition nor the reward function. Moreover, since we are not alloted time for pre-computation, it is not possible to derive estimations of these functions.

2.2 Heuristic search

Search algorithms for video controllers (as well as for other types of problems) are strongly based on the notion of approximating the distance of a given configuration (or *state*) to the *goal* (e.g., maximum score, catching the flag, etc.). Such approximations are found by means of a computationally efficient function, known as a *heuristic function*. By applying such a function to states reachable from the current one considered, it becomes possible to select more-promising alternatives earlier in the search process, possibly achieving better results (e.g., a higher score, or reaching the goal faster). The putative result is strongly tied to the quality of the heuristic function used: employing a perfect function means simply "strolling" onto the solution (i.e., no search de facto) and maximizing the solution score, while using a bad function could render the search less efficient than totally uninformed search, such as breadth-first search (BFS) or depth-first search (DFS).

2.3 Hyper-heuristics

In the area of combinatorial optimization the term hyper-heuristics was first used by Cowling et al (2000) to describe heuristics to choose heuristics. This definition of hyper-heuristics was expanded later (Burke et al, 2010a) to refer to an automated methodology for selecting or generating heuristics to solve hard computational search problems. In the process of hyper-heuristics learning, heuristics are used as building blocks. These heuristics can be of high level, usually complex and memory-consuming (e.g., landmarks and pattern databases), or low-level heuristics that are usually intuitive and straightforward to implement and compute.

Hyper-heuristics have been applied in many research fields, among them:

- Classical planning (Yoon et al, 2008; Levine et al, 2009; Fawcett et al, 2011).
- Classical NP-Complete domains, e.g., 2D and 3D bin-packing (Burke et al, 2010b, 2012), personnel scheduling (Burke et al, 2003; Cowling et al, 2000).

- Classical AI domains and puzzles, e.g., the Rush Hour puzzle (Hauptman et al, 2009), the game of FreeCell (Elyasaf et al, 2012), and the Tile Puzzle (Elyasaf et al, 2011; Arfaee et al, 2010).
- Mining RNA sequence-structure motifs (Elyasaf et al, 2015).

The growing research interest in techniques for automating the design of heuristic search methods motivates the search for automatic systems for generating hyper-heuristics.

2.4 Real-rime learning of hyper-heuristics

While research on learning hyper heuristics is numerous, there is little work on real-time learning of hyper heuristics. In many cases, converting offline algorithms to real-time ones is not trivial because real-time algorithms must handle the rapid change of the domain and the problems, while maintaining previous knowledge.

2.5 Solvers from GVGAI (Monte Carlo)

Many Monte Carlo Tree Search (MCTS) Browne et al (2012) submissions often outperformed other alternatives, with even the given, sample MCTS algorithm ranking 3rd on the 2014 GVGAI competition.

Top performers included fast evolutionary MCTS and knowledge-based fast evolutionary MCTS Perez et al (2014), which embedded the algorithm roll-outs within evolution. The individuals of the evolutionary algorithm were weight vectors, used to bias the roll-outs of MCTS. Every roll-out performed during the search evaluated a single individual of the evolutionary algorithm, providing as fitness the reward calculated at the end of the roll-out.

Also of note is MCTS with influence map Park and Kim (2015), the latter element being a numerical representation of influence on the game map, helping find a road to rewards over the horizon. The influence map essentially assigns a value to each object in the game world, representing if it is ’good’ or ’bad’. The value is then updated upon interaction with the object, eventually causing the player to focus on rewarding interactions.

3 Method

Recall our discussion on playing a video game for the first time. Our goal herein is to apply AI real-time learning techniques to develop controllers for video games that mimic the ability to generalize from previous experience, and reduce learning time as new levels present themselves.

Our approach comprises two principle components working in parallel as the game-playing agent encounters new levels:

1. Learning the heuristic function for evaluating the states' potential for achieving the different goals. The output of this component is a hyper-heuristic that receives a game state and returns a linear combination of different game goals, representing the potential for achieving these goals.
2. A game controller that uses generated hyper-heuristics in order to pick the most promising course of action. During its run the game controller passes encountered game states to the learning component. The latter learns the given states and incorporates the extracted knowledge to the evolved hyper heuristic.

3.1 Heuristic templates

Many games' goals can be generalized into the same principal goal, with minor variations. The most straightforward example is goals that relate to distances in the game, as in Pacman and Zelda, two seemingly different games Perez et al (2015). In Pacman, the main goal is to clear the board, i.e., "eat" all the pills and power pills. Bonus points are given for "eating" ghosts while under the effect of a power pill. In Zelda, the goal is to collect the key, then exit the level. Here, killing enemies with the sword rewards the player with bonus points.

The goals of collecting pills/keys in Pacman/Zelda, respectively, are computationally identical—we need to minimize the distance between the player and the objects, represented as number of steps. Similarly, we have the goal of maintaining a certain distance from either ghosts/enemies in Pacman/Zelda, respectively. The mechanism of calculating the distance is identical, however, the conditions regarding when to minimize or maximize it are different: in Pacman we wish to minimize the distance if we are under the effect of a power pill, and maximize otherwise; in Zelda we want to minimize distance if we can use our sword in the appropriate direction, and maximize it otherwise, if we are short on time and need to exit the level.

Heuristic templates are our method of encoding knowledge that is relevant to most game domains, in a way that can be customized as the game runs. They can be viewed as parameterized heuristics, where the parameters are the current state and any number of additional integers representing: a) a specific object in the game world; b) a type of an object in the game world; or c) a list (enum).

A complete list of our heuristic templates is given in Table 1.

Additional non-parameterized heuristics are given in Table 2.

Table 1 Heuristic templates.

Id	Name	Parameters	Description
1	Heuristic distance between object types	Integer type 1, Integer type 2	Parametrized by two types of objects. Values are taken from a set of types the controller encountered during the game. Calculates the geometric mean for each sprite type, and returns the Manhattan distance between the two means.
2	Count nearby NPCs	Integer Manhattan distance	Counts the number of NPCs that are less than Manhattan distance away from the player.
3	Score in K moves	Integer *k*	Game score if the controller does nothing for *k* moves. Useful for predicting how "safe" a position is.
4	Distance from corner	enum *corner*	If there is a path between *corner* and the player, returns its length. Otherwise, returns the Manhattan distance between the player and the corner.
5	Movable distance from immovable	Integer *immovable*	Calculates the A* distances between the selected immovable and all movable objects.
6	Heuristic distances	Object reference, list objects	Calculates the Manhattan distances between the given reference object and the list of objects. The reference object/Objects list can be any of the ones exposed by the state.

3.2 Hyper-heuristics

Combining several heuristics to get a more accurate one is considered one of the most difficult problems in contemporary heuristics research (Samadi et al, 2008; Burke et al, 2010a).

This task typically involves solving two major sub-problems:

1. How to combine heuristics by *arithmetic* means, e.g., by summing their values or taking the maximal value.
2. Finding exact conditions (i.e., *logic* functions) regarding *when* to apply each heuristic, or combinations thereof—some heuristics may be more suitable than others when dealing with specific state.

In order to accomplish the first task, we first need to generate heuristics from our heuristic templates. To do that we differentiate between the different return values of the templates. If the template returns a real number, the generated heuristic is the returned value multiplied by a fixed weight, randomized during the heuristic

Table 2 Non-parameterized heuristics.

Id	Name	Description
7	Facing NPC	Equals 1 if an NPC is directly in front of the player, -1 otherwise.
8	Avatar Resources Count	The total number of resources the player possesses.
9	Immovable Count	The total number of immovable objects that are not walls, in the game world.
10	NPC Count	The total number of NPCs in the game world.
11	Last Action is 'use'	Equals 1 if the last action successfully performed by the player was 'use'. Can be used to reward applying that action in games where that is beneficial.
12	Touching Walls Count	Counts the number of walls blocking the player (0-3).
13-18	A* Distance	A list of the distances—measured in number of states—between the player and one of: NPCs, immovable objects, portals, resources, movable objects. The distance is calculated as follows: at the beginning of each level, a graph is generated from the world, with a node for each position the player can stand on. Nodes are connected if they are adjacent. Upon a request for a distance between two nodes, a path is calculated using weighted A* Pohl (1970) and then cached for the duration of the level.

generation. If the returned value is a list of real numbers, we first have to perform one of the following aggregating arithmetic: min, max, sum, multiplication, or division. Similarly, we do the same for basic heuristics.

Our learning algorithm solves the problem of combining heuristics by quick evaluation, thereby ruling out unpromising hyper-heuristics. Finding the exact condition under which to apply each hyper-heuristic is solved by design—new hyper-heuristics are evaluated specifically on game states the controller encountered recently, and often states the controller will encounter shortly.

3.3 Learning hyper-heuristics through evolution

3.3.1 Individuals

Our individuals are hyper-heuristics. An individual is composed of a list of heuristics derived from heuristic templates, basic heuristics, and basic observations. A hyper-heuristic is a linear combination of these heuristics.

An individual is represented as an *abstract syntax tree* (AST) Jones (2003) of a Java class, which is compiled during evaluation.

3.3.2 Fitness function

As the real-time algorithm imposes fast hyper-heuristic evaluations, we expand the given initial state to the depth of X and set the individual fitness to be: $h(final\ state) - h(initial\ state)$ where h is the individual. In order to return values in a timely manner X was chosen to be 70.

At each depth we choose the next action in the following manner: We preform a 3-step-look-ahead by applying all possible actions to the depth of three. We choose the action that—on average—led to the highest heuristic value.

We used standard, Koza-style GP with the usual suspects: tournament selection with group size $k = 3$; subtree crossover; and constant and subtree-grow mutation operators.

3.4 GVGAI

As discussed in Section 1, the GVGAI competition proposes the challenge of creating controllers for a large range of stochastic real-time games, allowing the participants to train on a set of games, while testing them on a different, undisclosed set of games.

The competition imposes a single-thread limit, preventing us from participating because we need multi-threading for our online learning. Instead, we use the framework provided for the competition as a benchmark for our learning algorithm. We note that while the element of the undisclosed games is removed, we do not perform any sort of training on a subset of the games, or perform adjustments for specific games; instead we focus on real-time learning of the scenarios presented to the controller.

The GVGAI framework works by presenting the controller with a state that represents the fully observable world. The controller then has 40 milliseconds to return a selected action. Along with the state a forward model is provided in the form of a search tree, where each node represents a state, and each edge an action. Being of stochastic nature, performing action A from state S may yield different results whenever we attempt it.

3.5 Game controller

As discussed above, under the GVGAI framework our controller has 40ms to return an action for a given state. We first handle communication with the learning component: We save the current state for future GP generations, and take the best hyper-heuristic learned so far for our current evaluation.

We then use a 3-step-lookahead search algorithm that is almost identical to the one used during fitness calculation: expand the search tree to depth 3 and calculate the highest heuristic score reachable for each possible move. However, unlike during fitness calculation, we repeat this step, taking the average heuristic score, thus avoiding moves that are not likely to constantly lead to desirable moves.

4 And the Winner is . . .

Though a comparison with the 2014 entries would be somewhat of an "apples and oranges" case, given our intentional deviation from the strict GVGAI framework, we still note—with some pride—that we would have ranked number 3 of 19.

For testing, we used the 3 game sets used in the GVGAI competition in CIG 2014—which are now fully available as training sets. GVGAI used a formula-one like scoring system, awarding scores for the best performing contenders for each game. However, since we're mostly interested in the generality of our algorithm, and not in individual game performance, we compare it using percent of games won.

In the CIG 2014 GVGAI competition our algorithm would have ranked 3rd out of 19, with 36.3% of games won. If we increase the time the controller has before it must return an action from 40 to 80 milliseconds this percentage increases to 40.09% games won, leading us to believe our method is scalable, and would perform better with more resources.

Though the road ahead still has many paths to follow, we believe we have begun meeting our challenge: using genetic programming to evolve hyper heuristic-based *general* players through real-time, online learning.

References

Arfaee SJ, Zilles S, Holte RC (2010) Bootstrap learning of heuristic functions. In: Proceedings of the 3rd International Symposium on Combinatorial Search (SoCS2010), pp 52–59

Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on 4(1):1–43

Burke EK, Kendall G, Soubeiga E (2003) A tabu-search hyperheuristic for timetabling and rostering. J Heuristics 9(6):451–470, URL `http://dx.doi.org/10.1023/B:HEUR.0000012446.94732.b6`

Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward JR (2010a) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin J (eds) Handbook of Meta-Heuristics 2nd Edition, Springer, pp 449–468

Burke EK, Hyde MR, Kendall G, Woodward J (2010b) A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. IEEE Trans Evolutionary Computation 14(6):942–958, URL `http://dx.doi.org/10.1109/TEVC.2010.2041061`

Burke EK, Hyde MR, Kendall G, Woodward J (2012) Automating the packing heuristic design process with genetic programming. Evolutionary Computation 20(1):63–89, URL `http://dx.doi.org/10.1162/EVCO_a_00044`

Cowling PI, Kendall G, Soubeiga E (2000) A hyperheuristic approach to scheduling a sales summit. In: Burke EK, Erben W (eds) PATAT, Springer, Lecture Notes in Computer Science, vol 2079, pp 176–190, DOI 10.1007/3-540-44629-X_11

Elyasaf A, Zaritsky Y, Hauptman A, Sipper M (2011) Evolving solvers for FreeCell and the sliding-tile puzzle. In: Borrajo D, Likhachev M, López CL (eds) Proceedings of the Fourth Annual Symposium on Combinatorial Search, SoCS 2011, Castell de Cardona, Barcelona, Spain, July 15-16, 2011, AAAI Press, URL `http://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/view/4018`

Elyasaf A, Hauptman A, Sipper M (2012) Evolutionary design of FreeCell solvers. Computational Intelligence and AI in Games, IEEE Transactions on 4(4):270–281, DOI 10.1109/TCIAIG.2012.2210423, URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6249736`

Elyasaf A, Vaks P, Milo N, Sipper M, Ziv-Ukelson M (2015) Learning heuristics for mining RNA Sequence-Structure motifs. In: Genetic Programming Theory and Practice XIII (GPTP 2015), Springer

Fawcett C, Karpas E, Helmert M, Roger G, Hoos H (2011) Fd-autotune: Domain-specific configuration using fast-downward. In: Proceedings of ICAPS-PAL 2011

Hauptman A, Elyasaf A, Sipper M, Karmon A (2009) GP-Rush: using genetic programming to evolve solvers for the Rush Hour puzzle. In: GECCO'09: Proceedings of 11th Annual Conference on Genetic and Evolutionary Computation Conference, ACM, New York, NY, USA, pp 955–962, DOI 10.1145/1569901.1570032, URL `http://dl.acm.org/citation.cfm?id=1570032`

Jones J (2003) Abstract syntax tree implementation idioms. In: Proceedings of the 10th conference on pattern languages of programs (plop2003), pp 1–10

Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. Artificial intelligence 101(1):99–134

Levine J, Westerberg H, Galea M, Humphreys D (2009) Evolutionary-based learning of generalised policies for AI planning domains. In: Rothlauf F (ed) Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO 2009), ACM, New York, NY, USA, pp 1195–1202

Park H, Kim KJ (2015) Mcts with influence map for general video game playing. In: Computational Intelligence and Games (CIG), 2015 IEEE Conference on, IEEE, pp 534–535

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

Perez D, Samothrakis S, Togelius J, Schaul T, Lucas S, Couëtoux A, Lee J, Lim CU, Thompson T (2015) The 2014 general video game playing competition

Pohl I (1970) Heuristic search viewed as path finding in a graph. Artificial Intelligence 1(3):193–204

Samadi M, Felner A, Schaeffer J (2008) Learning from multiple heuristics. In: Fox D, Gomes CP (eds) Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008), AAAI Press, pp 357–362

Yoon SW, Fern A, Givan R (2008) Learning control knowledge for forward search planning. Journal of Machine Learning Research 9:683–718, URL `http://doi.acm.org/10.1145/1390681.1390705`