EnHiC: An Enforced Hill Climbing Based System for General Game Playing

Amin Babadi[1], Student Member, IEEE, Behnaz Omoomi[2], Graham Kendall[3,4]
[1]Department of Electrical and Computer Engineering, [2]Department of Mathematical Sciences, [3,4]School of Computer Science
[1,2]Isfahan University of Technology, [3]University of Nottingham UK Campus, [4]University of Nottingham Malaysia Campus
[1,2]84156-83111, Isfahan, Iran, [3]United Kingdom, [4]Malaysia
[1]a.babadi@ec.iut.ac.ir, [2]bomoomi@cc.iut.ac.ir, [3]graham.kendall@nottingham.ac.uk, [4]graham.kendall@nottingham.edu.my

***Abstract*—Accurate decision making in games has always been a very complex and yet interesting problem in Artificial Intelligence (AI). General video game playing (GVGP) is a new branch of AI whose target is to design agents that are able to win in every unknown game environment by choosing wise decisions. This paper proposes a new search methodology based on enforced hill climbing for using in GVGP and we evaluate its performance on the benchmarks of the general video game AI competition (GVG-AI). Also a simple and efficient heuristic function for GVGP is proposed. The results show that EnHiC outperforms several well-known and successful methods in the GVG-AI competition.**

Keywords—General video game playing; general game playing; enforced hill climbing; heuristic functions

I. Introduction

Video games belong to the class of real-time software applications. They usually try to put the player in dynamic, fast-paced, and complex situations and the player has to act quickly in order to win (or gain points) while avoiding losing (or losing points). This complexity means that finding the right choice, even for a human player, can be a very difficult task. This is why designing intelligent agents for playing video games is a challenging and interesting problem in AI.

Most of efforts in video game AI are limited to only one game (e.g. chess). In contrast, the concept of *General Video Game Playing* (*GVGP*) or *General Game Playing* (*GGP*) has been introduced in the past few years in order to tackle the problem of designing agents that are able to perform well in each arbitrary game environment [1]. It is clear that in designing such an agent, one cannot use domain-dependent knowledge (e.g. techniques of chess) [2]. The *General Game Playing Competition* (*GGP*) and *General Video Game AI Competition* (*GVG-AI*) has been running since 2005 and 2014, respectively. These competitions focus on several popular two-dimensional (2D) arcade games and they are a suitable benchmark for comparing GGP methods [3].

The problem of GGP is very much similar to the automated planning whose target is to design methods that can help robot-like agents to solve problems in an unseen environment. This problem can be formulated as a state-space search problem in which the planner tries to use a sequence of actions in order to transform the world state from an initial state to (one of) the goal state(s) [4]. One of the most successful search methods in automated planning is called *Enforced Hill Climbing* (*EHC*) that was first introduced in the popular *Fast-Forward* planning system (*FF*), the winner of *Artificial Intelligence Planning and Scheduling Systems Competition 2000* (AIPS-2000) [5], [6]. EHC is a variation of popular hill climbing method and several state-of-the-art planning systems have been designed using this search method [7]–[9]. Because of similarities between automated planning and GGP, evaluating EHC method in GGP domains seems to be an interesting line of research.

In this paper an EHC-based search mechanism for GGP is proposed. EHC uses a heuristic function in order to evaluate states and quality of this heuristic function has a great impact on the performance of EHC. In this work a new heuristic function with a simple and yet efficient structure is proposed. This mechanism is completely consistent with GVG-AI competition framework; so all the implementations and evaluations are carried out using the competition framework and will be publicly available after the competition.

This paper is organized as follows: Section II reviews the literature in GGP. Sections III and IV describe the details of our proposed heuristic function and the EHC-based search method, respectively. Finally, Sections V and VI report the results; draw conclusions; and propose future works.

II. Related works

The idea of general game playing goes back to 2005, when the first annual AAAI GGP competition was organized by Stanford University [10]. Since 2014, another annual competition has been started under the banner of the GVG-AI competition [3]. In these competitions, player has no prior information about games. In the GGP competitions, before starting each match, the player is given a formal abstract description of the game rules (i.e. the *game description*) along with the initial positions of the objects and the layout of the current level of the game (i.e. the *level description*). These descriptions are defined using the *Game Definition Language* (*GDL*), a language that is inspired by the *Planning Domain Definition Language* (*PDDL*) from automated planning [2], [11]. On the contrary, in GVG-AI competition, the player has no access to the formal description of the game; so the player has to discover the game rules on its own, only based on the given observations from the match. These observations include: 1) how much time is passed, 2) a list of available actions in the game, 3) location of each object along with some

This research is funded by Iran Computer and Video Games Foundation (http://www.irvgame.com/).

978-1-4799-8622-4/15/$31.00 ©2015 IEEE

typical properties of that object, 4) history of collisions that happened so far between player and any other object, and 5) whether the match has finished (player won or lost) or not (ongoing) [3]. The main concern of this paper is to compete with the methods proposed in the GVG-AI framework.

The first winner of GGP competitions in 2005 was *ClunePlayer* that tried to extract informative heuristic functions by simplifying each game such that the simplified versions capture the key aspects of the original game [12]. The winner of the 2006 competition was *FluxPlayer* which used reasoning techniques in order to construct heuristic functions [13]. The winner of the competitions in 2007, 2008, and 2012 was *CadiaPlayer*, that used Monte Carlo simulation search guided by a search-control learning mechanism [14]. The winner of the 2009 and 2010 competitions was a program called *Ary* that used a parallelized version of Monte Carlo tree search (MCTS) called the *Root Parallel Algorithm*. This program uses several instances of MCTS algorithm for analyzing each game state separately and finally, chooses the right action based on results of the executed analyses [15]. *TurboTurtle* was the winner of the competitions in 2011 and 2013. The winner of 2014 competition was a program called *Sancho* [16].

The winner of the first GVG-AI competition in 2014 was a program called *adrienctx* whose method was inspired by the *Hierarchical Open-Loop Optimistic Planning* [3].

In the GVG-AI framework, each program is allowed to choose its next action from a valid action set. The valid action set of each game is a subset of *{UP, DOWN, LEFT, RIGHT, USE, NIL}*. The first four actions are navigational actions that help the player change its position or orientation. Action USE can have different effect depending on the nature of each game and can be used for shooting, digging, attacking by sword, etc. There is also a null action (*NIL*) for determining that the player does nothing in the current game cycle. A game program (or so called controller, or agent) in the GVG-AI framework must implement two methods. The first method is responsible for controller initialization and should be executed in less than 1 second. The second method is called in every game cycle and lets the controller choose its next action in less than 40 milliseconds. If it takes the controller between 40 and 50 milliseconds to choose its next action, it returns action NIL automatically. If the controller violates the 1 second limit for initialization, or the 50 milliseconds limit for choosing its next action, the controller is disqualified from the current game [3].

III. The Heuristic Function

The quality of heuristic function plays a crucial role in determining the performance of hill climbing search methods. This section describes our proposed heuristic function that was used in the EnHiC system. This function (called h_{EnHiC}) is inspired by *SimpleStateHeuristic* function in the GVG-AI framework and uses a new concept, called *Preservation*.

Preservation value of a state is the amount of reward or penalty that the controller gets if it tries to preserve its position. To have a better understanding, consider a situation from the *Pacman* game shown in Fig. 1. In this situation, if Pacman stays in his current position, one of the ghosts will catch him after 4 moves, resulting in a very low preservation value.

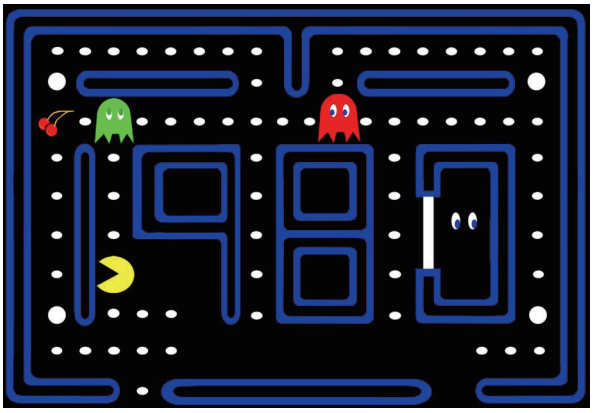

Fig. 1. Example of a situation in the Pacman game with a very low preservation value.

EnHiC always prefers states with higher preservation value. The pseudocode for computing preservation value is shown in Fig. 2. *Score* and *Adv* are two embedded functions in GVG-AI framework; *Score* simply returns the current score of the controller in the game and *Adv* gets a state *s* and an action *a* as input, and returns the state resulting from applying action *a* to state *s*.

As can be seen in Fig. 2, *ComputePreservation* function's core is a simple loop that applies NIL action for at most a limited number of steps (K). If the game is over in any of these states, this function stops the loop and returns preservation value regarding the final game result (lines 7-12). If game score is changed in any of these states, preservation value is updated regarding the new score (lines 13-16). If no important change in game is detected after K steps, 0 is returned as

```
Function ComputePreservation
 1:  Input: current state observation s
 2:  Output: preservation p
 3:  prev ← s
 4:  p ← 0
 5:  For i = 1 to K Do
 6:      next ← Adv(prev, ACTION_NIL)
 7:      If player is lost in next
 8:          Return p ← -αε^i
 9:      End If
10:      If player is winner in next
11:          Return p ← αε^i
12:      End If
13:      diff ← Score(next) - Score(prev)
14:      If diff ≠ 0
15:          p ← p + diff × βε^i
16:      End If
17:      prev ← next
18:  End For
19:  Return p
```

Fig. 2. Pseudocode for computing preservation value. Variable *K* is an integer for determining the horizon that the controller checks for computing preservation value. Variables α and β make sure that winning or losing in the game is more important than a simple change in the game score, and $0 < \varepsilon \leq 1$ is the discount factor. In this work $(K, \alpha, \beta, \varepsilon)=(5, 2\times10^7, 10^6, 0.9)$.

preservation value.

So far the concept of preservation has been described. We now explain the formula of EnHiC heuristic function h_{EnHiC}. To begin, h_{EnHiC} assigns a constant value to a state s if s is a terminal state (i.e. game is over in state s). If the player is winner in s, $h_{EnHiC}(s)=-\alpha$, and if player is lost in s, $h_{EnHiC}(s)=\alpha$ (this is the same α used in Fig. 2).

If s is a non-terminal state, one trivial observation is that the controller should always try to maximize its score, collect more resources, and go to states with higher preservation value. A simple and yet helpful feature of GVG-AI framework is that it can provide the controller with a list of objects from a specific type in each game state along with the distance they have to the controller's avatar. h_{EnHiC} uses this feature to compute the average distance from the position of controller's avatar to positions of *portals* and *resources* in each game state. Portals are objects that act like doors of the game. Resources are collectable objects in the environment that increase special kinds of player's properties, such as health, mana, or ammunition. h_{EnHiC} assumes that the controller's preferences are to always get closer to portals and resources while trying to decrease the number of *non-playable characters* (i.e. *NPCs*) in the game (because in GVG-AI games NPCs usually are enemies of the player). So the value of h_{EnHiC} for a state s is

$$\begin{aligned} h_{EnHiC}(s) = {} & AverageDistanceToPortals(s) \\ & + AverageDistanceToResources(s) \\ & + NumberOfNPCs(s) \times k_{NPC} \\ & - NumberOfControllerResources(s) \times k_{Resource} \\ & - Score(s) \times k_{Score} \\ & - ComputePreservation(s). \end{aligned} \quad (1)$$

where *NumberOfNPCs* and *NumberOfControllerResources* functions return the number of NPCs and controller's resources in state s, respectively. *AverageDistanceToXXX* functions compute the average distance between the controller's avatar and set of all objects from a specific type (e.g. portals) in the game. For example, if M is set of all portals in a state s, and position of an object O is denoted by (x_O,y_O), the average distance between the controller's avatar (denoted by P) and the set of all movable objects is

$$\frac{\sum_{O \in M} \sqrt{(x_P - x_O)^2 + (y_P - y_O)^2}}{|M|}. \quad (2)$$

Since value of *AverageDistanceToXXX* and *ComputePreservation* functions are of order 10^5, and the value of *NumberOfNPCs*, *NumberOfControllerResources*, and *Score* functions are of order 10^2, in order to smooth the weight of different terms in (1), we used three positive integer multipliers k_{NPC}, $k_{Resource}$, and k_{score}. In this work, $(k_{NPC}, k_{Resource}, k_{score})=(5\times10^5, 10^3, 10^6)$.

IV. Enforced Hill Climbing Search

Enforced hill climbing is a local search method that was first introduced by the FF planning system, the winner of Artificial Intelligence Planning and Scheduling Systems Competition 2000 (AIPS-2000). Pseudocode of this search method is shown in Fig. 3. The main difference between EHC and original hill climbing method is that EHC performs a breadth-first search until a successor of the current state with a better heuristic value is found (line 6). Once such a state is found, the path between the current and the new state is added to the end of the found path and the world state is transformed into the new state (lines 10-11). If no better successor is found, the search stops and returns failure (lines 7-9). This routine goes on until a goal state is found where search is completed and the final path is returned as an answer (line 13). This simple strategy turns out to be an efficient solution to the problem of plateau in the hill climbing search algorithm [5].

Pseudocode for EnHiC search method is shown in Fig. 4. In order to use the enforced hill climbing method in GVG-AI framework efficiently, several slight alterations were made to this search method:

- In GVG-AI framework, the controller has to choose its next action using a function *act* that is called once per frame and gives the controller at most 40 milliseconds to choose its next action [3]. One simple observation is that the controller cannot take as long as it wants to use the EHC (or any other) method for finding the best action that leads to a goal state. So we decided to run a new instance of EHC method once per frame just to find the best action in the current state. This seems to be a reasonable choice since there can be at most 2000 consecutive frames at each game run in GVG-AI framework and it means that each game can end in at most a couple of minutes. So being able to choose and execute actions with a relatively high speed seems to be a crucial need for any GVG-AI controller.

Function EnforcedHillClimbing

```
 1:  Input: current state observation so0
 2:  Output: an action sequence for
     transforming so0 into a goal state
 3:  sequence ← <> (empty action sequence)
 4:  s ← so0
 5:  While s is not a goal state
 6:      Perform breadth-first search for a
         state s' such that h(s')<h(s)
 7:      If no better state is found
 8:          Return "Failure"
 9:      End If
10:      Add action sequence from s to s' to
         the end of sequence
11:      s ← s'
12:  End While
13:  Return sequence
```

Fig. 3. Pseudocode for original enforced hill climbing method as used in the FF planning system [5].

```
Function EnHiC_act
 1:  Input: current state observation so_0
 2:  Output: next action to be performed
 3:  bestAct ← a randomly chosen action
 4:  If player loses in Adv(so_0, bestAct)
 5:      bestAct ← ACTION_NIL
 6:  End If
 7:  bestScore ← h_EnHiC(so_0)
 8:  Initialize openSet to be a queue with
     one element <so_0, ACTION_NIL>
 9:  While openSet is not empty AND time is
     not over
10:     <state, action> ← openSet.Dequeue()
11:     Foreach available action nextAct
12:         nextState ← Adv(state, nextAct)
13:         If nextState is visited
14:             Continue
15:         End If
16:         If action != ACTION_NIL
17:             nextAct ← action
18:         End If
19:         nextScore ← h_EnHiC(nextState)
20:         If nextScore < bestScore
21:             bestScore ← nextScore
22:             bestAct ← nextAct
23:         End If
24:         Mark nextState as visited
25:         openSet.Enqueue(<nextState,
            nextAct>)
26:     End Foreach
27:  End While
28:  Return bestAct
```

Fig. 4. Pseudocode for EnHiC's core EHC-based search method.

- Another simple observation is that even if the controller does manage to find a goal state using an action sequence of length more than one, there is no guarantee that this action sequence is completely applicable. This is because most of the games in GVG-AI framework run in fast-paced dynamic environments where any changes may happen after each single frame. Thus any action in the found action sequence (except the first one, obviously) may become an invalid or wrong action after its previous actions are executed. So once EHC method finds a better state than the current one, instead of storing the complete path to the found state, it just stores the first action in this path as the corresponding action to that state (line 25).
- An important issue in every hill climbing search method is how to handle plateaus. Since there can be a lot of situations with delayed rewards (such as the *Camel Race* domain), EHC may fail to find a better successor in the 40 milliseconds time limit. For escaping plateaus, EnHiC chooses a random action out of available action set before it starts the EHC search method (line 3). If EHC fails to find a better successor in the specified time limit, the chosen random action is returned as the current action of the controller. The only consideration here is that in some games (such as *Frogs* domain) choosing a random action may cause the controller to completely lose the game. For avoiding such situations, EnHiC system checks the validity of chosen random action and replaces that with a null action if it is proved that the chosen action causes the controller to lose (lines 4-6).
- Since GVG-AI framework gives the controller 40 milliseconds for choosing its next action, it seems to be a wise strategy to make the most of this short time limit by continuing the search even after finding the first successor that has a better heuristic value. This means that after beginning the EHC search method, EnHiC always stores the best state it could find along with heuristic value of that state, until the controller's time is approximately over. After that, EnHiC returns the action corresponding to the best state it could find. If no such state has been found, the chosen random action is returned.

V. Experimental Results

We evaluated the EnHiC system using all three game sets available in the GVG-AI competition framework. Each game has five different levels that can be downloaded from the competition website[1]. We ran each level five times; so every game has been played 25 times. For comparing EnHiC search method with other GGP approaches, five different GGP controllers from GVG-AI competition and previous works have been evaluated:

- **Random**: This controller simply returns a random action in each game cycle.
- **One-Step-Look-Ahead**: This controller computes heuristic value for all immediate successors of the current state (states that are reachable from the current state using only one action) and picks the action that leads to the state with the best heuristic value. Heuristic function used in this controller, called *SimpleStateHeuristic* is available in GVG-AI framework.
- **GA**: This controller picks its next action in each game cycle using a genetic algorithm.
- **MCTS**: This controller uses Monte Carlo tree search algorithm for choosing its next action in each game cycle.
- **KB-FE-MCTS**: This controller uses a variation of MCTS algorithm, called *Knowledge-based fast evolutionary MCTS* as described in [2].

All results, except for the (best) results of KB-FE-MCTS that are reported directly from [2], have been recorded using a computer with Microsoft Windows 7 OS, 6 GB RAM, and 2.30 GHz Core i7 CPU. Results obtained from CIG 2014 training set games are shown in Table I. The most important measure in the GVG-AI competitions is the percentage of

[1] Available at http://www.gvgai.net/.

TABLE I. COMPARISON OF PERCENTAGE OF VICTORIES AND AVERAGE SCORE BETWEEN ENHIC AND OTHER CONTROLLERS ON CIG 2014 TRAINING SET GMES. BEST RESULTS ARE IN BOLD. EACH NUMBER IS THE AVERAGE RESULT OBTAINED BY PLAYING 5 DIFFERENT LEVELS OF EACH GAME FOR 5 TIMES.

	Percentage of Victories						**Average Score**					
Game	***Random***	***One-Step-Look-Ahead***	***GA***	***MCTS***	***KB-FE-MCTS***	***EnHiC***	***Random***	***One-Step-Look-Ahead***	***GA***	***MCTS***	***KB-FE-MCTS***	***EnHiC***
Aliens	4%	36%	96%	8%	**100%**	**100%**	41.92	41.2	64.44	36.72	56.52	**67.0**
Boulderdash	0%	0%	0%	0%	**23.3%**	0%	2.2	0.4	6.28	9.96	**18.24**	3.8
Butterflies	56%	0%	92%	88%	**100%**	**100%**	31.2	18	30.4	27.84	**31.76**	26.96
Chase	12%	0%	40%	12%	**97.4%**	88%	3.68	0.08	5.44	4.04	**9.78**	8.56
Frogs	0%	60%	40%	24%	28%	**100%**	-1.6	0.2	0	-0.88	-0.48	**1**
Missile Command	24%	24%	48%	20%	65.9%	**72%**	-0.56	-1	3.04	-1.44	**4.54**	4.44
Portals	4%	0%	36%	12%	**37%**	20%	0.04	0	0.36	0.12	**0.37**	0.2
Sokoban	0%	0%	12%	0%	**13.4%**	**20%**	0.48	0	1	0.16	0.7	**1.2**
Survive Zombies	0%	12%	16%	44%	**53.9%**	36%	9.32	5.48	37.76	13.28	24.66	**50.68**
Zelda	0%	0%	4%	8%	**37%**	28%	0.36	0	2.8	0.08	0.9	**4.84**
Overall	10%	13%	38%	22%	55.6%	**56%**	8.7	6.4	15.1	9	14.7	**16.87**

victories in each game, and average score is then used for breaking the ties. Overall obtained score in all games does not seem to be an interesting measurement since each game uses a unique scoring system. On the contrary, the overall percentage of victories can be helpful for determining the success of a controller.

According to Table I, EnHiC system is the leading controller in both overall percentage of victories (56%), and overall average score (16.87). EnHiC even managed to slightly outperform KB-FE-MCTS, which is one of the most successful variations of the well-known MCTS algorithm. This shows the high potential of enforced hill climbing search method with respect to Monte Carlo tree search methods.

An important question is, when does EnHiC system perform well? Since the enforced hill climbing method does not keep track of the history of its actions and always uses the breadth-first search to simply find a state with better heuristic value, it converges to a goal state faster when there are only a few sub-goals and those sub-goals are independent from each other. Thus EnHiC system has a great performance when it comes to fast-paced games where the player is most concerned with staying alive and achieving a simple goal (such as collecting resources or reaching a specific point). This is the main reason that EnHiC achieves a great performance in *Aliens*, *Butterflies*, *Chase*, Frogs, and *Missile Command* games. In the Aliens domain, the player should collect points by destroying the blocks and enemies (i.e. aliens) in the game while avoid being shot by the enemies, and the game is won when all the aliens are eliminated. In the Butterflies domain, there are some butterflies moving randomly in the environment, the player gets 2 points by capturing each butterfly, and the target is to capture all the butterflies in the game. Descriptions of all games in the GVG-AI competition is available at [3].

The most difficult games for EnHiC, are *Boulderdash*, *Portals*, and *Sokoban*. It seems that one of the most important factors that keeps EnHiC from obtaining good results in these two games is that EnHiC uses Euclidean distance while in these games Euclidean distance does not count as a good measure of distance between two points. So using the concept of shortest path could increase the performance of EnHiC in these games. The problem is that computing shortest path in GGP domains is not always straightforward. For example, in the Portals domain, there can be several portals in each level and the controller can use those portals to teleport itself from on location to another. Thus, points that are far (or even unreachable) from each other in the Euclidean space, can become very close to each other (or even identical) in the Portals domain.

For the sake of better evaluation, 4 different versions of EnHiC system have been tested:

- **EnHiC**: This version uses the complete search system as described in Fig. 4. In each one of the next 3 versions of this system, one of the alterations that were described in section IV are undone (name of each version determines the alteration that is undone).
- **Fast EnHiC**: In this version, EHC search is stopped once the first better state is found.
- **Random-Free EnHiC**: In this version, when EHC search has failed to find a better state, this version returns a NIL action instead of a random one.
- **Preservation-Free EnHiC**: In this version computation of preservation value is removed from heuristic function so we can evaluate how the concept of preservation affects the quality of our heuristic function.

Table II shows the results obtained from running four different variations of EnHiC system on CIG 2014 training set games. As before, each game is played by each version for 25 times. As it can be seen in Table II, the weakest variation of EnHiC system that scored 30% of victories is Fast EnHiC that returns the first better state it can find instead of looking for better states within the 40 milliseconds time limit. The next controller is Random-Free EnHiC that has no strategies for escaping plateaus. This controller won only 43% of its matches. Preservation-Free EnHiC with 49% of victories is the next controller. Comparison of performance of this controller with performance of full EnHiC system shows that Preservation-Free EnHiC won fewer matches in domains in which the environment is more dynamic (such as Aliens, Chase, Frogs, Missile Command, and *Zelda*). Finally, full

TABLE II. COMPARISON OF PERCENTAGE OF VICTORIES AND AVERAGE SCORE BETWEEN FOUR DIFFERENT VARIATIONS OF ENHIC SYSTEM ON CIG 2014 TRAINING SET GAMES. BEST RESULTS ARE IN BOLD. EACH NUMBER IS THE AVERAGE RESULT OBTAINED BY PLAYING 5 DIFFERENT LEVELS OF EACH GAME FOR 5 TIMES.

	Percentage of Victories				**Average Score**			
Game	***Fast EnHiC***	***Random-Free EnHiC***	***Preservation-Free EnHiC***	***EnHiC***	***Fast EnHiC***	***Random-Free EnHiC***	***Preservation-Free EnHiC***	***EnHiC***
Aliens	**100%**	**100%**	64%	**100%**	58.16	62.24	61.52	**67.0**
Boulderdash	0%	0%	0%	0%	0.56	2.4	**4.44**	3.8
Butterflies	64%	**100%**	**100%**	**100%**	24.72	**29.76**	26	26.96
Chase	52%	12%	84%	**88%**	6.12	4.2	**8.84**	8.56
Frogs	8%	**100%**	84%	**100%**	-0.24	**1**	0.52	**1**
Missile Command	52%	40%	68%	**72%**	1.68	2.2	**5.04**	4.44
Portals	0%	**20%**	**20%**	**20%**	0	**0.2**	**0.2**	**0.2**
Sokoban	8%	0%	**20%**	**20%**	1.08	0.4	**1.32**	1.2
Survive Zombies	16%	**44%**	32%	36%	6.16	49.48	**55.32**	50.68
Zelda	0%	16%	20%	**28%**	1.88	4	4	**4.84**
Overall	30%	43%	49%	**56%**	10.01	15.59	16.72	**16.87**

EnHiC system again outperformed other three versions in both overall percentage of victories and overall average score.

Tables III and IV show the results obtained from running EnHiC system on CIG 2014 validation set and CIG 2015 training set games, respectively. EnHiC system scored an overall percentage of victories of 33% and 15% in CIG 2014 validation set games and CIG 2015 training set games, respectively. The main reason of this weakness is due to fast decision making of EnHiC system and the fact that this system is not able to formulate plans when there are several correlated sub-goals in the game (such as *Bait* domain).

VI. CONCLUSIONS

General game playing (GGP) can be a great and interesting benchmark for testing artificial intelligence approaches. This paper has analyzed the power of enforced hill climbing (EHC) search method in the field of general game playing. A GGP controller based on EHC method was proposed and its performance was evaluated with respect to several well-known GGP controllers using available games from CIG 2014 and 2015. Also a simple and yet efficient heuristic function was proposed and several alterations were made to the EHC-based search method to increase its performance. The results are very promising and show that this system outperforms several powerful controllers such as MCTS-based controllers.

Since enforced hill climbing is a powerful search method in domains in which the agent has to make quick decisions and show quick reflexes, EnHiC system has a great performance in fast-paced games in GVG-AI competition benchmark (such as Aliens and Camel Race). However, this search method does not work out so well when it comes to domains in which the agent needs to make long-term decisions in order to reach a specific goal (such as Boulderdash and Sokoban). This is why EnHiC has a weak performance in CIG 2015 training set games. So an interesting subject for future works can be discovering how to use AI or machine learning approaches to strengthen EHC method for using in GGP.

ACKNOWLEDGMENTS

This research is funded by *Iran Computer and Video Games Foundation*. Iran Computer and Video Games Foundation is a non-profit, non-governmental organization, which was established in 2007 with the aim of offering support to game studios in developing their ideas and products. We should thank Hasan Karimi (the president of this foundation), for his useful feedbacks on this paper.

TABLE III. PERFORMANCE OF ENHIC SYSTEM ON CIG 2015 TRAINING SET GAMES. EACH NUMBER IS THE AVERAGE RESULT OBTAINED BY PLAYING 5 DIFFERENT LEVELS OF EACH GAME FOR 5 TIMES.

Game	**Percentage of Victories**	**Average Score**
Bait	8%	4.64
Bolo Adventures	0%	0.52
Brainman	16%	25.48
Chips Challenge	0%	1.24
Modality	32%	0.32
Painter	32%	727.04
Real Portals	0%	0
Real Sokoban	0%	0.08
The Citadel	20%	1.88
Zen Puzzle	40%	25.8
Overall	15%	78.7

TABLE IV. PERFORMANCE OF ENHIC SYSTEM ON CIG 2014 VALIDATION SET GAMES. EACH NUMBER IS THE AVERAGE RESULT OBTAINED BY PLAYING 5 DIFFERENT LEVELS OF EACH GAME FOR 5 TIMES.

Game	**Percentage of Victories**	**Average Score**
Camel Race	100%	1
Dig Dug	0%	21.68
Fire Stroms	0%	-0.2
Infection	92%	13.2
Fire Caster	0%	6.28
Overload	8%	19.96
Pacman	0%	105.08
Sea Quest	8%	229
Whacamole	36%	17.76
Eqqomania	84%	63.8
Overall	33%	47.76

REFERENCES

[1] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, S. M. Lucas, R. Miikkulainen, T. Schaul, and T. Thompson, "General video game playing," in *press*.

[2] D. Perez, S. Samothrakis, and S. Lucas, "Knowledge-based fast evolutionary MCTS for general video game playing," in *IEEE Conference on Computational Intelligence and Games (CIG'14)*, 2014, pp. 1–8.

[3] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

[4] G. Malik, D. Nau, and P. Traverso, *Automated Planning Theory and Practice*. Elsevier, 2004.

[5] J. Hoffmann and B. Nebel, "The FF planning system: fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[6] J. Hoffman, "FF: The Fast-Forward Planning System," *AI Magazine*, no. 1, pp. 57–62, 2001.

[7] S. Yoon, A. Fern, and R. Givan, "Learning Heuristic Functions from Relaxed Plans.," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'06)*, 2006.

[8] J.-H. Wu, R. Kalyanam, and R. Givan, "Stochastic enforced hill-climbing," *Journal of Artificial Intelligence Research*, pp. 815–850, 2011.

[9] J. Hoffmann, "The Metric-FF planning system: translating 'ignoring delete lists' to numeric state variables," *Journal of Artificial Intelligence Research*, vol. 20, pp. 291–341, 2003.

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

[11] A. Babadi, M. Feyzbakhsh-Rankooh, and G. Ghassem-Sani, "LePoP: a learning based heuristic partial order planner," *unpublished*.

[12] J. Clune, "Heuristic evaluation functions for general game playing," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'07)*, 2007, pp. 1134–1139.

[13] S. Schiffel and T. Michael, "Fluxplayer: A Successful General Game Player," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'07)*, 2007, pp. 1191–1196.

[14] Y. Bjornsson and H. Finnsson, "CadiaPlayer: a simulation-based general game player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, 2009.

[15] J. Méhat and T. Cazenave, "A parallel general game player," *KI-Künstliche Intelligenz*, vol. 25, no. 1, pp. 43–47, 2011.

[16] M. Swiechowski, J. Mandziuk, and Y. S. Ong, "Specialization of a UCT-based General Game Playing Program to Single-Player Games," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.