Efficient Implementation of Breadth First Search for General Video Game Playing

Suguru Ito*, Zikun Guo*, Chun Yin Chu†, Tomohiro Harada*, Ruck Thawonmas*
Intelligent Computer Entertainment Laboratory
*College of Information Science and Engineering, Ritsumeikan University, Shiga, Japan
†Graduate School of Information Science and Engineering, Ritsumeikan University, Shiga, Japan
ruck@is.ritsumei.ac.jp

***Abstract*—This paper proposes an efficient implementation of Breadth First Search (BFS) for General Video Game Playing (GVGP). This method is specialized for deterministic games, which often cannot be solved by a single action and require a more extensive search to solve. Most existing AI programs cannot search the game space efficiently, and thus perform poorly in deterministic games. To improve the efficiency of tree search, we propose limiting the branching of game tree in BFS. Hash code is assigned to each tree node and used to identify similar game states. A tree node with a game state that has been visited in previous search will not be expanded. Using a deterministic game set for evaluation, our experiment shows that the proposed method outperforms existing methods.**

I. Introduction

Most existing game AI programs are designed for a specific game. Such AI cannot be applied to other games, and do not possess the versatility of human mind. On the other hand, General Game Playing (GGP) aims at the creation of AI that can perform well across various games. In GGP, the AI program has no knowledge about the game prior to playing it, and has to learn the game through training or simulation.

General Video Game Playing (GVGP) brings the concept of GGP to video games. Similar to GGP, GVGP attempts to develop AI programs that can adapt to different types of arcade games. Since 2014, IEEE CIG has been hosting the General Video Game AI Competition [1].

Games in GVGP can be divided into deterministic games and non-deterministic games. In deterministic games, no random element exists, which often means that the avatar is the only moving sprite in the game map. Non-deterministic games are games that contain random elements, e.g., NPCs are moving randomly. Common techniques such as Monte Carlo Tree Search (MCTS) are strong against non-deterministic games but weak to deterministic games. Therefore, many successful AIs in the GVG-AI Competition use MCTS for non-deterministic games, but switch to Breadth First Search (BFS) when deterministic games are played [1].

Due to its excessive memory usage, BFS is not fit for deep search. This paper proposes the use of hash code, which improves the efficiency and performance of BFS in GVGP. Test result is presented to show the effectiveness of the proposed method.

II. General Video Game Playing Framework

The GVG-AI Framework[1] is developed by researchers at University of Essex. The framework allows games to be created easily using the Video Game Description Language (VGDL) [2], and provides a vast corpus of games that researchers can use to evaluate their AIs. In the framework, an AI is given only 40 milliseconds for decision-making at each game step. The framework informs the AI of the current game state (e.g., score, the coordinates of player and other sprites). However, information regarding the game rule, i.e., the role of each sprite and the termination condition of the game, is not given to the AI. Based on information provided by the framework, the AI can learn the game through simulation, and make a reasonable decision in the game space.

III. Breadth First Search in GVGP

BFS is a search algorithm that prioritizes nodes that are closer to the root [3]. In GVGP, BFS is mostly used for deterministic games. Although the GVG-AI Framework requires the AI to complete its search within 40ms, in most deterministic games, sprites on the map will not move as long as the player is idle. Thus, it is possible to keep returning ACTION_NIL (do nothing) until the search is ended, allowing deep search that takes more than 40ms without witnessing any change on the game map. However, BFS requires vast memory space to store its tree nodes before reaching the solution. As memory resource is limited in GVG-AI Framework, the depth reachable by BFS is limited in GVGP.

IV. Modification to BFS

In basic BFS, all discovered nodes are stored in the game tree, even if their corresponding game states have already been visited in previous nodes. In our proposed method, a node that represents a game state already visited in other nodes will not be expanded. To identify visited states efficiently, our proposed method assigns hash code to each visited node.

Algorithm 1 shows how hash code is generated for a given state. The initial value of $hashcode$ and $prime$ can be any integer and any prime number, respectively. Member values used to represent a game state are listed in Table I. All member values must be integer.

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

TABLE I
MEMBER VALUES USED TO GENERATE HASH CODE

Member	Description
AvatarType	AvatarType
AvatarPosition	Position.x × map length + Position.y
AvatarOrientation	AvatarOrientation transform into 1∼4
ObjectID	specific value of each object
ObjectType	kind of object
ObjectPosition	Position.x × map length + Position.y

Algorithm 1 GenerateHashCode

```
1: $hashcode \leftarrow any\ integer$
2: $prime \leftarrow any\ prime\ number$
3: $list \leftarrow Information\ of\ each\ object\ in\ the\ map$
4: $hashcode \leftarrow hashcode \times prime + AvatarType$
5: $hashcode \leftarrow hashcode \times prime + AvatarPosition$
6: $hashcode \leftarrow hashcode \times prime + AvatarOrientation$
7: for Observation obj : list do
8:     $hashcode = hashcode \times prime + obj.Position$
9:     $hashcode = hashcode \times prime + obj.ObjectID$
10:    $hashcode = hashcode \times prime + obj.ObjectType$
11: return hashcode
```

V. Experiment

In order to evaluate the performance of the our method, the Training Set 3 of the GVG-AI Framework was used in the experiment. Training Set 3 consists of ten puzzle games, which are all deterministic. Each game contains 5 levels. The size of Hash codes in use is 64 bits.

A. Methodology

In this experiment, each AI will play each game level 5 times, i.e., a total of 250 game levels, the average number of states of which is 3.09E+80, were played by each AI. For comparison, SimpleBFS and SampleMCTS, described below, were tested along with the proposed method. For each game, the win rate of each AI was recorded, and the AI with the highest win rate was considered the winner of that game.

- SimpleBFS: Based on the basic BFS without optimization using hash code.
- SampleMCTS: One of the sample AIs in the GVG-AI Framework that performed the best among all sample controllers in Training Set 3.

When testing the proposed method and SimpleBFS, the maximum number of tree nodes was set to 20000, so as to avoid crash due to insufficient memory. When the number of tree nodes has reached the limit, each AI selects a solution by using the reward function. In this experiment, the reward function is based only on the score of the game. Moreover, the parameters of the proposed method were empirically initialized as $hashcode = 17, prime = 31$.

B. Experiment Result

Table II shows the win rate of each controller in every game of Training Set 3. The average depths reached by the proposed method and SimpleBFS in each game are shown in Table III. As shown in Table II, the proposed method outperforms the other AIs in 7 games. Table III shows that the proposed method was able to search deeper than SimpleBFS in all games. Despite being able to search deeper, the proposed method could not perform better than SimpleBFS in games where reaching the solution required even deeper search.

TABLE II
WIN RATE OF EACH METHOD IN EVERY GAME

	SimpleBFS	SampleMCTS	ProposedBFS
Bait	0.2	0.08	0.68
BoloAdventures	0	0	0
Brainman	0.04	0.08	0.4
ChipsChallenge	0	0.2	0.8
Modality	0.2	0.28	1
Painter	0.56	0.8	0.6
RealPortals	0	0	0
RealSokoban	0	0	0.52
TheCitadel	0.24	0.12	0.76
ZenPuzzle	0.28	0.12	0.28
Average	0.152	0.168	0.504

TABLE III
AVERAGE SEARCH DEPTH OF EACH METHOD

	Depth SimpleBFS	Depth ProposedBFS
Bait	7.4	24.2
BoloAdventures	7.2	13.4
Brainman	7	19.8
ChipsChallenge	7.8	20.2
Modality	7	23.25
Painter	6.8	7.4
RealPortals	8	13.8
RealSokoban	8	22.75
TheCitadel	8	16.4
ZenPuzzle	8	10.8

Interestingly, SampleMCTS outperformed all other AIs in Painter. Painter dose not require deep search to solve, and can be cleared by taking actions actively. The two BFS-based AIs spent most of thier time searching, but did not take enough actions, leading to their inferior performance in the game.

VI. Conclusion

This paper proposed the use of hash code to enhance the efficiency of BFS in GVGP, and analyzed how the proposed method impacts the performance of BFS. The result of our experiment proves that the proposed method is effective in many deterministic games in GVGP. We look forward to further improving BFS by optimizing the reward function.

References

[1] D. Perez, et al. ”General Video Game AI: Competition, Challenges and Opportunities,” Proc. of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 4335-4337, 2016.
[2] T. Schaul. ”A Video Game Description Language for Model-based or Interactive Learning,” Proc. of 2013 IEEE Conference on Computational Intelligence and Games, pp. 1-8, 2013.
[3] Stout, Bryan. ”Smart moves: Intelligent pathfinding,” Game Developer magazine 10, pp. 28-35, 1996.