Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230913

Algorithm structure for real-time general game playing agents

Chenming Ge

School of Software, Tongji University, Shanghai, China, 201804

2150236@tongji.edu.cn

Abstract. General Game Playing is an examination of AI agents that are designed without any explicit knowledge of a particular game. Instead, they are required to play a variety of games that they have never seen before by merely reading the description of the rules at runtime, without any intervention from a human being. Previous successful agents have concentrated their efforts on turn-based games, employing either generic heuristics or the Monte Carlo/UCT simulation technique. Real-time games have dominated the market in recent years, and as a result, agents that have more reflexibility have emerged as an important area of research. In this paper, the author evaluates whether or not it is possible to implement the already-existing algorithms into a real-time playing GGP agent, and we present a potential algorithm structure that, on the basis of this evaluation, could be able to solve the challenge.

Keywords: Algorithm, Structure, Real-time, General Game Playing

1. Introduction

Approximately fifty years before the advent of the first computers, the study of generic game playing emerged. General Game Playing (GGP) aims to construct intelligent agents that automatically learn how to play a wide variety of games at an expert level by comprehending the rules and determining the optimal strategy.

GGP provides players with a formal description of the target game that describes the game's rules and reward system in a language comparable to Game Description Language. They are then forced to play the game without aid from a human player. After comprehending the Game Description Language description of a game, it is possible to design a propositional network with input, output, and transition nodes to reflect the game's dynamics as well as a number of nodes that perform logical operations.

The creation of the International General Game Playing Competition (IGGPC), which has been held annually since 2005, may be one of the most significant historical developments in the vast history of GGP agents. During each race, a game server that administers the matches is connected to game-playing agents. In each game, a start-clock and a play-clock are the two timers that are utilized. During the start-clock, the agent has the opportunity to review the game description prior to the commencement of play. By controlling the play-clock, the agent has the opportunity to assess each move after the game has began. The server, on the other hand, directs play, reports moves, maintains track of time, and saves game results [1].

DOI: 10.54254/2755-2721/6/20230913

The initial focus of the research was the development of broad game-playing methods, but the focus gradually shifted to the development of high-performance game-playing systems that could compete with the smartest humans in individual games.

2. General game playing

2.1. Algorithms

Construction of the algorithm for GGP agents begins with the concept of reinforcement learning. The field of RL explores agents that can learn to maximize a concept of reward by acting, watching the consequences, and then calculating their next move based on the current circumstances.

2.1.1. Minimax. The standard method of embedding domain-specific information using a minimax-based game-tree search complemented by an autonomously learnt heuristic evaluation function formed the basis for all of the first successful GGP agents [1]. For instance, figure out the level of truth of the formulas defining goal and terminal predicates in the state to evaluate [2]. However, GGP programs often rely on a limited set of generic features that apply to a wide variety of games rather than manually creating a collection of domain-specific features for evaluation [1]. Next, a real-time determination is performed regarding the applicability and relative usefulness of the characteristics for the present game.

2.1.2. Monte-Carlo. GGP agents that calculate their actions using Monte-Carlo simulations have shown considerable potential in recent years. Throughout the process, it anticipates a set of outcomes based on an estimated range of input values as opposed to predefined values. It makes far less use of heuristic evaluation functions and may even be able to operate without them [3].

2.1.3. UCT. The field of GGP entered a new era when the Upper Confidence Bounds applied to Trees (UCT) algorithm was introduced. It is a generalization of the UCB1 method that works with game trees. The method determines the average return for each state-action combination performed by simulating the progressive construction of a game tree in memory. It gives a practical and effective method for balancing the exploration. While in the simulation's tree, it chooses the action to investigate by

$$a^* = argmax_{a \in A(s)}\{Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}}\}$$

The innovative component of UCT is UCT bonus, which is the second term. In contrast, the Q function is identical to the action value function in an MC algorithm. Instead of returning the winning percentage of this action, nothing is returned. The N function returns the number of times a state has been visited or the number of times a single action has been sampled in that state [1]. If an unexplored action happened in A(s), it will be added to the list of possible actions in states with no estimated value that have not yet been discovered. To learn more about it, the algorithm selects it by default before any action that has been previously sampled. This selection rule enables the UCT algorithm to strike a balance between using the deemed best action and recognizing the inferior ones in order to obtain the highest compensation.

2.2. GGP agents

Due to the progress, ever more GGP agents can be discovered on the internet. A minimum of three components are required for an agent to compete in the GGP: an HTTP server to communicate with the GGP game server, the ability to interpret the rules using GDL, and artificial intelligence (AI) for playing the supplied games strategically [1].

DOI: 10.54254/2755-2721/6/20230913

There are other GGP agents employing various search algorithms in a variety of settings. A typical example is the 2006 winner of the AAAI GGP Competition, the FLUXPLAYER agent. It generates game states and leverages calculus-based action reasoning. The planning phase employs common game-tree search techniques, including two well-known improvements to the iterative deepening depth-first search: history heuristics for rearranging the moves in accordance with the values discovered in lower-depth searches and transposition tables for caching the value of states already visited during the search [4]. At greater depths, non-uniform depth-first search is utilized. The evaluation of the heuristic function is based on fuzzy logic, in which static structures in the game descriptions are identified by employing the semantic properties of the game predicates [2].

The 2008 GGP competition winner, CADIAPLAYER, is a gaming agent whose decisions are solely based on Monte Carlo simulation searches [3]. Moreover, it is governed by an efficient search-control learning mechanism.

The next notable GGP agent, and my personal favorite, is AlphaGo Zero, a generalization of AlphaGo. AlphaZero uses a single neural network to provide both a probability distribution over possible actions and a prediction of the winner. It plays games with itself by doing many Monte Carlo tree search simulations for each state [5]. In accordance with the outcomes of the current neural network, each simulation chooses a move in each state with a low visit count, a high move probability, and a high value [6]. The search returns a vector p representing a probability distribution over moves. Once the game has concluded, the neural network's parameters are modified to minimize the difference between the expected and actual outcome. After a self-play game, the winner is recorded, passed back to all states, and compared to the previous best player [7]. If the new player wins by a margin of at least 55 percent [8], the original player will be replaced.

3. Real-time assumption

Most GGP agents make an essential limiting assumption about the games they play, namely that the game is turn-based, zero-sum, and symmetric between the players. Breaking down these obstacles is an active area of research. Memory use was also seen as troublesome. For larger games, it was discovered that storing the entire state at each node of a Monte Carlo tree search required too much RAM [7]. All of these issues are essential when considering a real-time game.

3.1. Existing solution

The following approaches have already functioned as feasible solutions for some of the challenges. Maintaining separate neural networks individually trained for each player is one strategy to addressing player asymmetry [9]. Moreover, because these neural networks are all taught to infer attributes from the game state that are relatively similar, it is conceivable to integrate the initial layers of the neural networks of all the players. Comparable layer unification was utilized in AlphaGo Zero, where the networks of value and policy were merged, unlike in machine translation, each language had its own head, which had the effect of regularizing the network.

In multi-player games, it is difficult to determine if other players are aware of your movements. Therefore, it is sufficient to have everyone predict a move at the same time and then select the next state based on the combined prediction. This concept of having a head for each player has the added benefit of being easily generalizable to several players and permitting simultaneous play [7]. This method is quite straightforward and simple to comprehend, but it requires a great deal of time due to the large number of weights to be calculated. The author's opinion is that it is possible to reduce the workload by combining the identical weights of multiple players, hence avoiding repeated calculations.

Last but not least, for the issue with memory storage, the most important discovery is that the state frequently changes by something like a constant amount at each step, necessitating a method for capturing only the changes. It is currently recommended to utilize a persistent array to hold the state, which permits the use of more RAM each modification, and to generate new root nodes and new

DOI: 10.54254/2755-2721/6/20230913

nodes along the path to the modified element. Point to the original nodes for those subtrees that have not altered. Then, design a binary tree atop the state array to allow for updates [10].

3.2. Assumed structure

All of these solutions are only applicable to turn-based games, despite the fact that it appeared that they could fix nearly all of the problems. Even if all of these issues have been resolved, the development of GGP agents is still in its infancy. The proposed use of GGP agents should enable a single computer agent to play a variety of games, such as chess, riddles, and all types of real-time games.

As a result of the real-time nature of the challenge, it is no longer required for GGP agents to determine whether their opponents have made their moves; instead, they must take the opportunity by moving quicker than their opponents. In addition, memory capacity will become a much larger issue, as the game tree would expand if every move were recorded as it was previously, therefore the recording changes may not be as successful as they were in turn-based games.

3.3. UCT only

It is preferable to begin with what we already have in order to determine the most effective means of overcoming such obstacles. It is impossible to determine every move in a real-time game, even if the UCT algorithm, the previous most effective one, is utilized as a training approach. For each period of game time, conducting each stimulation will require a significantly bigger quantity of data than in a turn-based game, and the number of possible outcomes would be so great that it would be nearly impossible to depict every movement in a single tree.

Moreover, if real-time games involved competition, it would be necessary to account the actions of other players during the research phase. Therefore, the specific algorithm structure will differ between the single-player and multi-player versions.

Considering only the single player, the use of UCT will be nearly identical: discover the optimal move and maximize the prize. Due to the immutability of the surroundings, the optimal route will finally be printed down for the AI to follow. Simply stimulate more till we obtain the optimal response. For a competitive game, however, the self-playing component must keep two connected UCT algorithms running concurrently, one providing the other with the result of a stimulation. Different initial settings will be assigned to each UCT to represent the variances between players. After a given number of stimulations, there will be optimal motions for both parties.

This kind of framework may allow an agent to play the game for the first time, but it requires far more time and memory space than it can afford. Which leads to the following proposal, which aims to kill two birds with one stone.

3.4. Real-time training structure

According to the author, the primary structure of the agents will be divided into two portions, one for strategy formulation and the other for action. In lieu of regular turn-based stimulation, my proposed solution for memory storage issues is for the agent to make time-based decisions, deciding what to do during the following period of time. First, extract the possible movements from the description. Then, assign a fixed number of important elements to each movement and indicate the weights of each decisive factor. Focusing will eliminate the majority of insignificance, attaining the goal of saving time. Next, decide what to do by periodically selecting the optimal action based on the current circumstance. Each training step will continue to use the UCT method to construct a tiny tree and combine comparable parameters.

This strategy would reduce the amount of time required for stimulation, but it may result in a major issue, namely the inability to determine the optimal action for a long-term outcome. Therefore, developing a lot of long-term plans will be vital for the agent. This can be analyzed using either the available data or a computation that assigns varying weights to various actions based on the time period.

DOI: 10.54254/2755-2721/6/20230913

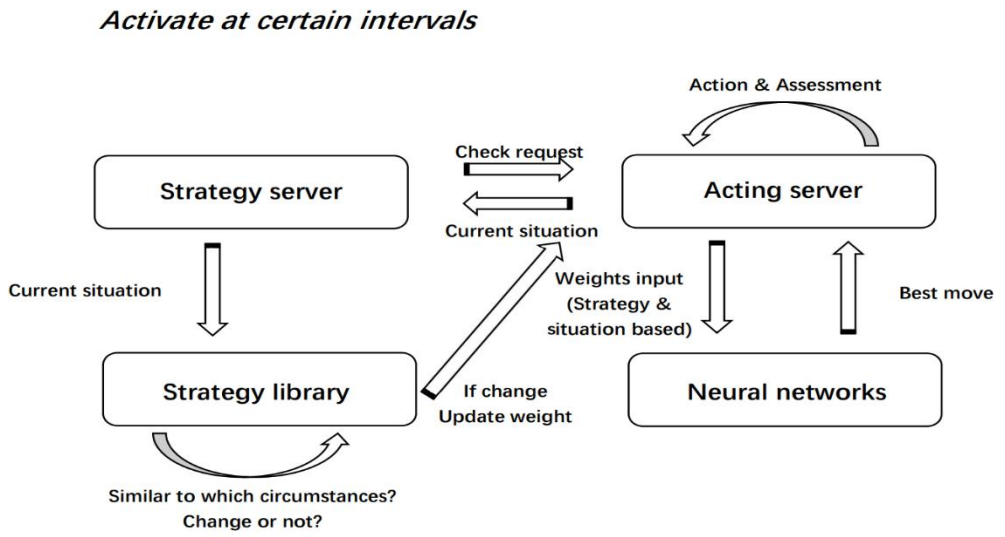

Figure 1. Con-structure of real-time agents.

At the training state, the strategy-making component will select which strategy to prioritize based on the overall scenario, while the action-deciding component will input each crucial aspect and the strategy into the neural network to determine which action should be taken. Then, decision-making and action will occur concurrently, reaching maximum efficacy.

4. Conclusion

Through this essay, we can have a general idea of game playing in general and the masterpieces created to overcome obstacles.

"A human should be able to change a diaper, plan an invasion, butcher a hog, con a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight valiantly, and die gallantly," said the brilliant author Robert Heinlein. As for robots, which are designed to function like humans, they should also utilize some of these skills. At least, they should be capable of playing a variety of games like humans.

To further improve the real-time algorithms in GGP, there are a number of promising future research directions to explore. The subsequent phase would involve documenting the precise parameters and constructing the entire algorithm structure based on existing methodologies, hence enhancing simulation playouts.

It is stated that games are miniature representations of actual occurrences. The author believes that with the development of GGP agents, we will sooner or later be able to implement all of these algorithms into a robot capable of coping with similar situations in the actual world.

References

[1] Finnsson, H., & Björnsson, Y. (2008). Simulation-based approach to general game playing. In Aaai. Vol. 8, pp. 259-264.

[2] Schiffel, S., & Thielscher, M. (2007). Fluxplayer: A successful general game player. In Aaai. Vol. 7, pp. 1191-1196.

[3] Bjornsson, Y., & Finnsson, H. (2009). Cadiaplayer: A simulation-based general game player. IEEE Transactions on Computational Intelligence and AI in Games, 1(1), 4-15.

DOI: 10.54254/2755-2721/6/20230913

[4] Schiffel, S., & Thielscher, M. (2006). Automatic construction of a heuristic search function for general game playing. Department of Computer Science, 16-17

[5] Whitehouse, D., Cowling, P. I., Powley, E. J., & Rollason, J. (2013, November). Integrating monte carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game. In Ninth Artificial Intelligence and Interactive Digital Entertainment Conference.

[6] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484-489.

[7] Goldwaser, A., & Thielscher, M. (2020, April). Deep reinforcement learning for general game playing. In Proceedings of the AAAI conference on artificial intelligence (Vol. 34, No. 02, pp. 1701-1708).

[8] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ... & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419), 1140-1144.

[9] Kamra, N., Gupta, U., Fang, F., Liu, Y., & Tambe, M. (2018, April). Policy learning for continuous space security games using neural networks. In Thirty-Second AAAI Conference on Artificial Intelligence.

[10] Rao, J., & Ross, K. A. (2000, May). Making B+-trees cache conscious in main memory. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (pp. 475-486).