Adaptive Genetic Algorithm to Optimize the Parameters of Evaluation Function of Dots-and-Boxes

Fangming Bi, Yunchen Wang, and Wei Chen(✉)

School of Computer Science and Technology,
China University of Mining and Technology,
Xuzhou City 221116, Jiangsu Province, China
{bfm, wyc, chenw}@cumt.edu.cn

Abstract. Designed an evaluation function with parameters, and used genetic algorithm to optimize the parameters. This paper considers the objective function's variation trends in searching point and the information is added to the fitness function to guide the searching. Simultaneously adaptive genetic algorithm enables crossover probability and mutation probability automatically resized according to the individual's fitness. These measures have greatly improved the convergence rate of the algorithm. Sparring algorithm is introduced to guide the training, using gradient training programs to save training time. Experiments show skills in playing Dots-and-Boxes are greatly improved after its evaluation function parameters are optimized.

Keywords: Adaptive genetic algorithm · Evaluation function · Game

1 Introduction

Machine game is an important branch of artificial intelligence research and has been hailed as drosophila in the field. Computer Game System can be divided into four parts: situation represents, action set, the evaluation function and game tree search. Search algorithm is the basic method to solve the problems in artificial intelligence. Minimax theorem proposed by Von Neumann and Boerl in the 1920s is the mathematical basis for searching algorithm. Alpha-beta pruning algorithm which began in the 1950s is a big step forward in the search efficiency. PVS (Principal Variation Search, also known as NegaScout) search algorithm in 1980 has higher search efficiency than alpha-beta search algorithm when the game tree is strong orderly [1]. MTD(f) algorithm appeared in 1994, always with an empty window detection approaching true value, complete search with the help of "Transposition Table", is slightly better than the PVS search. Both are the current mainstream method. Most game trees are so large that unable to

F. Bi—Fund Project: National Natural Science Foundation and Shanxi Provincial People's Government Jointly Funded Project of China for Coal Base and Low Carbon (No. U1510115), the Qing Lan Project, the China Postdoctoral Science Foundation (No. 2013T60574).

J.-H. Lee and S. Pack (Eds.): QShine 2016, LNICST 199, pp. 416–425, 2017.
DOI: 10.1007/978-3-319-60717-7_41

complete the search. Conventional measure is to search for a certain depth, then the leaf node is approximately evaluated by evaluation function. The quality of the evaluation function directly affects the development of the situation [2]. If search engine is the game system's eyes, the evaluation function would be the system's brain. Evaluation function generally must contain elements of five aspects, that is fixed pawn value, pawn position value, pawn flexibility value, threats and protect value, dynamic adjustment value, and the value of each aspect is composed of a number of parameters. Combination of the above parameters in the evaluation function is often dependent on the programmer's own knowledge and experience which makes it difficult to achieve the optimal. Some scholars have introduced genetic algorithm into the evaluation function, trying to learn the dynamic relation between the pieces, but the effect is not very satisfactory. In this paper we take the dots-and-boxes as example, design an evaluation function, use adaptive genetic algorithm [3] to optimize evaluation function parameters. In order to enhance the local searching ability of genetic algorithms, knowledge of problem domain is added to the fitness function [4]. Our study has been organized as followings: Sect. 2 specify the challenges for genetic algorithm to be applied to optimization of evaluate function parameters and gives the scheme of genetic algorithm that we used in this paper and describes the adaptive genetic algorithm; Sect. 3 describes the experimental strategies and presents the experimental results; Sect. 4 is the conclusion of this paper.

2 Solutions of Genetic Algorithm Applied to Optimization of Evaluate Function Parameters

2.1 Challenges for Genetic Algorithm to Use in Game

The genetic algorithm was proposed by professor Holland at the University of Michigan in American in 1969 and formed a type of simulated evolutionary algorithm after summarized by Dejong, Goldberg et al. In recent years, Genetic Algorithm as an important part of the intelligent computing (neural networks, fuzzy processing and evolutionary computation), has been a focus of research and made very good results in the field of application such as more extreme value function optimization problems, combinatorial optimization problems, scheduling problems and so on.

Traditional hill-climbing algorithm finds the optimal solution by comparing with neighboring nodes, and is restricted by ranges of initial samples and single direction searching, and is easy to fall into local optimum [5].

Simulated annealing is a stochastic optimization algorithm based on the Monte-Carlo iterative solution strategies. It attempts to simulate the high temperature object annealing process to find the global optimal solution or the approximate global optimal solution. It can avoid local minima, but the fatal flaw is too slow, running too long [6].

Ant colony algorithm (ACO) converges on the optimization path through pheromone accumulation and renewal. It has the ability of parallel processing and global searching and the characteristic of positive feedback. But the convergence speed of ACO is lower at the beginning for there is only little pheromone difference on the path at that time.

Genetic Algorithm has a high degree of parallel, weak dependence on initial value and the quick global searching ability. Its robustness is also significantly better than the previous two algorithms. But it has such disadvantages as premature convergence, low convergence speed and so on. To used in game, these disadvantages will be amplified in particular as the fitness of the generation depends on the game result. With the increase of the degree of evolution, it will take more rounds to fight it out. In view of its weakness, in this paper, the first-order differential information of adjacent two generations of the objective function was added into the fitness function to strengthen the search ability and prevent premature. The adoption of the adaptive genetic algorithm greatly improved the convergence speed. So it is most likely to succeed in Dots-and-Boxes.

2.2 Parameter Selection and Coding Scheme of the Evaluation Function

Dots-and-Boxes starting with an empty grid of dots, players take turns, adding a single vertical or horizontal line between two adjacent dots. A player who completes the fourth side of a 1×1 box earns one point and takes another turn. The game ends when no more lines can be placed. The winner of the game is the player with the most points.

The design of evaluation function of Dots-and-Boxes typically rely on several theorems [7], introduced as following.

Theorem 1. Regardless of the initial size of the board, there is always the following equation holds:

$$\text{Dots} + \text{Doublecrosses} = \text{Turns}$$

where Dots is the number of point of the initial board, Doublecrosses is the number of doublecross in the whole play and Turns is the total number of rounds to go through in the whole play.

Theorem 2. If total number of the board nodes is odd, then the Upper Hand side must form an odd number of Long Chain in order to win, and the After Hand side must form an even number Long Chain for win, and vice versa.

Theorem 2 is known as Long-Chain Theorem, and it is an important basis for the design of the evaluation function. Take 4×4 chessboard as example, according to Long-Chain Theorem, the Upper Hand side must form an odd number of Long Chain in order to win. Suppose that after the Upper Hand moves, it forms the situation (there are two Long-Chains marked as a, b) as shown in Fig. 1(a). So, no matter what strategy the After Hand to take, the final chain will be captured by the Upper Hand who is in a dominant position. However, relying solely on the Long-Chain theorem can not guarantee the accuracy of evaluation function. For example, suppose after the Upper Hand moves, it forms the situation as shown in Fig. 1(b) where there are also two long chains (marked as a, b), but the opponent simply select the edge numbered 1 in the Fig. 1(b), then the situation reverses, and the Upper Hand are in absolute disadvantage.

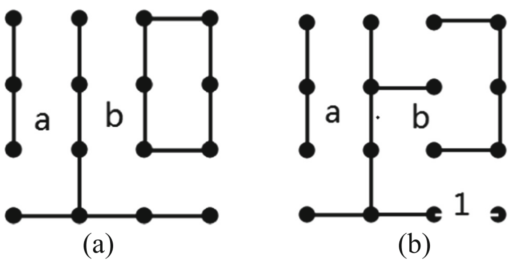

Fig. 1. An even number of Long-Chain case

The above analysis shows that the valuation depends on the combination of a variety of board elements and is sensitive to the parity of the number of board elements. So, we design the following evaluation function.

$$v(r_1, r_2, r_3 \ldots r_n) = \sum_{i=1}^{N} A_{i1}^{r_i} A_{i2}^{|1-r_i|} \tag{1}$$

where $r_i = n_i \bmod 2, n_i$ represents the number of the i-th type of board elements in the current chess game; A_{i1}, A_{i2} is the parameter of the i-th type of chess-type; N is the number of types of different board elements. In this article, we select some main board elements as Long-Chain, Short-Chain, DoubleCross, boxes with Freedom Degree of 3 and 4, so N equals 5.

Encode each parameter A_{ij} with 6 bit binary code, a total of 60 bits constitute a set of chromosomes string.

2.3 Calculation of the Fitness Function and Population Selection

Fitness is a main indicator that describes the individual performance and genetic algorithm select the fittest individuals based on it. Selection of the fitness function determines the algorithm optimization orientation and has a great impact on the convergence of the algorithm and the convergence rate.

We use the following method to design objective function. Let each individual play with an existing game algorithm (sparring algorithm), then determine the value of the objective function according to the number of rounds per game takes. If we win, the fewer the number of rounds is, the greater the function of the value should be; if the other side wins, then the more the number of rounds is, the greater the value of the function should be. Therefore, the objective function is designed as follows.

$$f(x) = \begin{cases} 60 + \frac{(60-x)^2}{k} & \text{we wins} \\ x & \text{the other side wins} \\ 60 & \text{draw} \end{cases} \tag{2}$$

where x is the number of rounds of a game, k is adjustable coefficient that represents the importance of the number of rounds to fitness when we wins. Here take k = 6. The fitness function is mapped directly from the objective function, namely:

$$\mathrm{fit}(x) = f(x) \tag{3}$$

In order to fully considering the trend of the objective function, preventing the preferred chromosome hovering only in a relatively flat area in genetic process which would cause "premature", and we put the first difference of the objective function of adjacent generations to join in, using the following new fitness function [4]:

$$\mathrm{fit}(x)' = \varepsilon \cdot \frac{\mathrm{fit}(x) - \mathrm{fit}_{\min}}{\mathrm{fit}_{\max} - \mathrm{fit}_{\min}} + (1 - \varepsilon) \cdot \frac{\nabla f(x) - \nabla f_{\min}}{\nabla f_{\max} - \nabla f_{\min}} \tag{4}$$

where weights $\varepsilon \in [0, 1]$ called the control factor, to reflect the importance of values of the fitness function and the rate of change of the function in solving this problem. $\mathrm{fit}(x)$ is the original fitness function. $\mathrm{fit}_{\min}$ and $\mathrm{fit}_{\max}$ denote respectively the minimum and maximum value of individual fitness in the current generation of the population. $\nabla \mathrm{fit}_{\min}$ and $\nabla \mathrm{fit}_{\max}$ are defined as:

$$\nabla f(x) = f(x^p) - f(x^c) \tag{5}$$

$$\nabla f_{\min} = \min\{f(v_1^p) - f(v_1^c), \ldots, f(v_n^p) - f(v_n^c)\} \tag{6}$$

$$\nabla f_{\max} = \max\{f(v_1^p) - f(v_1^c), \ldots, f(v_n^p) - f(v_n^c)\} \tag{7}$$

where x^p, x^c denote the parent and offspring chromosomes respectively, n represents the size of the population.

2.4 Crossover and Mutation Operation

There are many crossover methods, single-point crossover, multi-point cross, sequence cross, cycle cross, etc. For convenience, we use single-point crossover, randomly select one of the binary bits in the chromosome as a cross point, then cross the two parameters of the parent individuals to form the sub's parameters [8]. Each parameter's cross rate is p_c.

Each parameter is mutated at variation rate p_m. Because of the use of the binary string representation, here just flip 0–1 for the gene according to the gene mutation rate. Mutation is a local random search, in conjunction with the selection/crossover to ensure the effectiveness of the genetic algorithm and maintain the diversity of the population at the same time, to prevent the emergence of non-mature convergence.

Here choosing to operate each parameter in the chromosome is due to that a chromosome contains many parameters. If directly operate the entire chromosome, some parameters may not get enough crossover and mutation which makes the convergence time become longer and the effect is not ideal.

2.5 Adaptive Genetic Algorithm

The convergence of the genetic algorithm is affected by crossover rate p_c and mutation rate p_m. The larger the cross rate, the faster the rate of new individuals to produce. But if the crossover rate is too large, it is not conducive to the protection of chromosome structure of the current high fitness individuals. But if the crossover rate is too low, the individual evolution is too slow and the search process is stalled. For the mutation rate, if it is too low, it's not conducive to generate new individuals and easy to fall into local optimum search; If it is too high, the genetic algorithm can easily degenerate into a random search algorithm. Since there is no fixed way to determine p_c and p_m, the optimization can only be optimized through continuous experiment and this process is very fussy. To this end, we introduce adaptive genetic algorithm which can automatically adjust with individual fitness [9, 10]. Here are the formulas of p_c and p_m:

$$p_c = \begin{cases} p_{c1} - \frac{(p_{c1}-p_{c2})(f'-f_{avg})}{f_{max}-f_{avg}} & f' \geq f_{avg} \\ p_{c1} & f' < f_{avg} \end{cases} \tag{8}$$

$$p_m = \begin{cases} p_{m1} - \frac{(p_{m1}-p_{m2})(f_{max}-f)}{f_{max}-f_{avg}} & f \geq f_{avg} \\ p_{m1} & f < f_{avg} \end{cases} \tag{9}$$

where $p_{c1} = 0.9$, $p_{c2} = 0.6$, $p_{m1} = 0.1$, $p_{m2} = 0.001$. f_{max} is the group's largest fitness value. f_{avg} is the average fitness value of each generation of the population. f' is the greater fitness value of the two individuals to cross. f is the fitness value to the mutation individual. Expression analysis shows that when it is closer to the maximum fitness value, the value of p_c and p_m is smaller, and this reduces the possibility of good genes being destroyed and ensure the convergence of the algorithm.

3 Experimental Testing and Results

3.1 Experimental Strategies

Adaptive genetic algorithm requires a lot of training to get a better race result, time is the key factor that must be considered. In this paper, we take gradient training method to achieve the purpose that takes less time to achieve better results. It specifically includes three aspects, gradually increase the search depth of the game algorithm; gradually increase the intensity of sparring algorithm and choosing an efficient game tree search algorithm.

Search depth has an enormous influence on the time, in the beginning, due to evaluation function parameters have not been effectively optimized, even if searching a great depth the estimates are still not accurate. Therefore, the time cost is too high. However, when the function parameters are effectively optimized, if the search depth is still not corresponding increase, chess will not be able to better enhanced, genetic algorithms will think this is because of the current individual's gene is not good, so it continue evolution and then convergence time will increase.

Sparring algorithm will also affect the accuracy of the training results and time. If sparring algorithm is too strong in the beginning, choice to the genetic algorithm tend to be simplify. If sparring algorithm is always weak, it's not conducive to choose the best individual. If always use the same kind of sparring algorithm, it's easy to fall into local optimum. In this paper, the different stages of the training use different intensity of sparring algorithm and in the training process, the evaluation function is interspersed with random parameters to prevent the search into the local optimum.

The strength of sparring algorithm is reflected in two aspects - the accuracy of the evaluation function and the depth of the search. At the start of training, we use the evaluation function designed according to our own experience. After the training reaches a certain stage, we use the evaluation that is consistent with the sparring algorithm, and at this time, we use the depth of search to distinguish the skill in playing chess. In the early, middle, late stage of the training, take 2, 5, 7 as the search depth of the genetic algorithm, and 2, 8, 10 as the sparring algorithm's. Now, we need choosing an efficient search algorithm.

In 1978, Stockman proposed SSS* algorithm and proved that it is a correct minimax algorithm and that it never explores a node that alpha-beta ignore. Moreover, for practical distributions of tip value assignments SSS* will explore strictly fewer game tree nodes than Alpha-Beta. However, SSS* algorithm has disadvantages of big storage requirement and the need to maintain the OPEN table [11, 12].

Paper [13] proposed that by reformulating the algorithm, SSS* can be expressed simply and intuitively as a series of calls to Alpha-Beta, yielding a new algorithm called AB-SSS*. AB-SSS* visits the same interior and leaf nodes in the same order as SSS* and resolves the problems mentioned above with SSS*. AB-SSS* has been implemented in high-performance game-playing programs for checkers, Othello and chess. In this paper, we using AB-SSS* as the searching algorithm.

3.2 The Experimental Results

Train the evaluation function optimized by adaptive genetic algorithm that has an improved fitness function with gradient training experiment. Champions of 50th generation, 100th generation, 150th generation, 200th generation, 250th generation, 300th generation formed a group to carry out round robin of successively hand. One is awarded 3 points for a win, 1 for a draw and 0 for a lose. Finally, the case is shown in Fig. 2.

It can be found by the line chart that for the evaluation function optimized by improved genetic algorithm and gradient training, the more generations to train, the

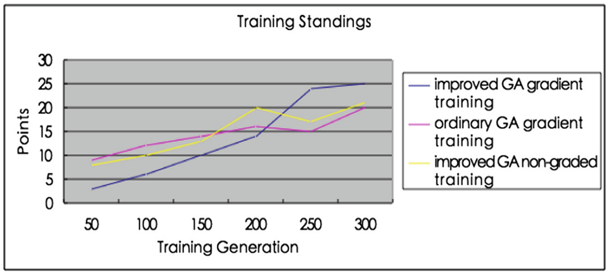

Fig. 2. Training standings

stronger the chess is and more obvious the effect is. However the differences between evaluation functions optimized by ordinary genetic algorithm gradient training and improved genetic algorithm non-graded training respectively is relatively small between generations. This shows that the improved genetic algorithm has faster convergence rate.

It was found that after the program is iterated to 300 generations, the optimized parameters are substantially no longer change. The parameter values at this time are shown in Table 1.

Table 1. Training results.

Parameter	A_{11}	A_{12}	A_{21}	A_{22}	A_{31}	A_{32}	A_{41}	A_{42}	A_{51}	A_{52}
Value	53	7	27	19	12	15	11	9	5	8

Experiment results show that game algorithm with evaluation function using the above parameter combination has greater possibilities to win compared with sparring algorithm.

In order to verify the gradient training method we proposed can reduce the training time and ensure the accuracy, our experiment take 8 as the search depth of both genetic algorithm and training algorithm in the training in different stages and using the same evaluation function. Due to the time cost is too high, we only train 50 generation. The results are shown in Table 2 and Fig. 3.

Table 2. Trainning time.

Generation	Time consumption (min)	
	Gradient training	Non-gradient training
10–20	10	50
20–30	23	66
30–40	21	80
40–50	25	103

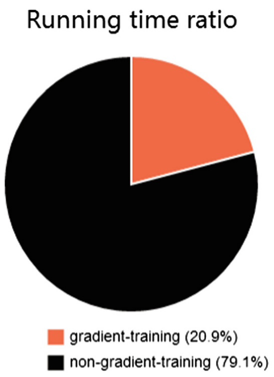

Fig. 3. Running time ratio

In the Table 2 and the Fig. 3 we find that the time gap of two is obvious. Then we used the same evaluation function of the same generation of both gradient training and non-gradient training to play with each other for 50 rounds. Winning statistics are shown in Table 3.

Table 3. Winning statistics.

Generation	Win (rounds)	
	Gradient training	Non-gradient training
10–20	27	23
20–30	24	26
30–40	25	25
40–50	23	27

In the Table 3 we find that increment of the search depth dose not bring obvious advantages. It validates our strategy to a certain extent.

4 Conclusion

There are precocious and other defects for classical genetic algorithm. In this paper, we introduced a new method of calculating the fitness function that considers the objective function's trends in search point, and the trend information is added to the fitness function to guide search. Simultaneously adaptive genetic algorithm enables crossover probability p_c and mutation probability p_m automatically resized according to the individual's fitness. These measures have greatly improved the speed of convergence of the algorithm. The genetic algorithm is introduced to optimize the evaluate function parameters which avoids manually adjust parameters in the traditional way and ensures the accuracy and objectivity of the parameters. Experiments show skill in playing Dots-and-Boxes greatly improved after its evaluation function parameters optimized. This also provides a new way of thinking for development of high-level game algorithm.

References

1. Hongkun, Q., Peng, Z., Yajie, W., et al.: Analysis of search algorithm in computer game of Amazons. In: 26th Chinese Control and Decision Conference, pp. 3947–3950. IEEE Press, New York (2014)
2. Duan, Z.: An improved evaluation function for Connect6. In: 24th Chinese Control and Decision Conference, pp. 1685–1690. IEEE Press, New York (2012)
3. Wei, X.-K., Shao, W., Zhang, C., et al.: Improved self-adaptive genetic algorithm with quantum scheme for electromagnetic optimization. IET Microw. Antennas Propag. **8**, 965–972 (2014)

4. He, X., Liang, J.: The objective function using genetic algorithms gradient. J. Softw. **12**, 981–985 (2001). (in Chinese)
5. Luo, B., Zheng, J., Yang, P.: GA-based directional climbing. Comput. Eng. Appl. **44**, 92–95 (2008). (in Chinese)
6. Qi, J.-Y.: Application of improved simulated annealing algorithm in facility layout design. In: 29th Chinese Control Conference, pp. 5224–5227. IEEE Press, New York (2010)
7. Li, S., Li, D., Yuan, X.: Research and implementation of dots-and-boxes. J. Softw. **7**, 256–262 (2012)
8. Deng, X.: Application of adaptive genetic algorithm in inversion analysis of permeability coefficients. In: Second International Conference on Genetic and Evolutionary Computing, WGEC 2008, pp. 61–65. IEEE Press, New York (2014)
9. Huang, Y.-P., Chang, Y.-T., Sandnes, F.-E.: Using fuzzy adaptive genetic algorithm for function optimization. In: Annual Meeting of the North American on Fuzzy Information Processing Society, NAFIPS 2006, pp. 484–489. IEEE Press, New York (2006)
10. Yanhong, P.: Wind power fitness function calculation based on niche genetic algorithm. In: International Conference on Sustainable Power Generation and Supply, pp. 1–5. IEEE Press, New York (2012)
11. Stockman, G.C.: A minimax algorithm better than alpha-beta? Artif. Intell. **12**, 179–196 (1978)
12. Ibaraki, T.: Generalization of alpha-beta and SSS* search procedures. Artif. Intell. **29**, 73–117 (1986)
13. Plaat, A., Schaeffer, J., Pijls, W., de Bruin, A.: SSS* = alphabet + TT. Technical report TR-CS_94-17, Department of Computing Science, University of Alberta, Edmonton, AB, Canada (1994)