Enhance Performance of Action Evaluation Functions with Stochastic Optimization Algorithms

Nguyen Quoc Huy[1,2(✉)], Dao Duy Nam[1,3], and Dang Cong Quoc[4]

[1] SaigonTech, SaigonTech Tower, Lot 14, Quang Trung Software City, District 12, Ho Chi Minh City, Vietnam
{huy.nq, namdd}@saigontech.edu.vn
[2] Saigon University, District 5, Ho Chi Minh City, Vietnam
[3] High school for the gifted, VNUHCM, 153 Nguyen Chi Thanh, District 5, Ho Chi Minh City, Vietnam
[4] Hue University, 03 Le Loi, Hue City, Vietnam
dangcongquoc@hitu.edu.vn

Abstract. In this paper, we describe how to optimize the weights of board cells from data set of game records, the weights of board cells are applied in the action evaluation function which usually uses to enhance Monte Carlo Tree Search programs. The general optimization process is introduced and discussed, and one specific method is implemented. We use Othello as a testing environment, and experiment results is better if the action evaluation function is better.

1 Introduction

The emergence of Monte Carlo Tree Search (MCTS) has led to considerable result in resolving the difficult problems of board games with a very large search space as well as the games that are difficult to build a board evaluation function. MCTS need not a board evaluation function, but it needs a good action evaluation function to enhance its performance. Static knowledge combined with an action evaluation function is a popular method when implementing a Monte Carlo framework in order to improve the quality of the simulated games of simulation phase as well as for knowledge bias of selection phase. Many such systems have been developed, and used domain knowledge encoded from game records to provide a good probability distribution for random games. Weighting of board cells is one of many methods to incorporate domain knowledge into board game programs. Historically, Artificial Intelligence game programmers learned the target game, and weighted the board cells by their experience. Today, the weighting of cells can be optimized automatically by many methods, if game records are available.

The optimization process has three main elements: variables to be optimized, the objective function, and the optimization method. This process will optimize the variables, the optimized variables will be applied in an action evaluation function. There are many representations of variables, many kinds of objective functions, many optimization methods. The optimization process is introduced detail in Sect. 3.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2017
P. Cong Vinh et al. (Eds.): ICCASA 2016, LNICST 193, pp. 183–192, 2017.
DOI: 10.1007/978-3-319-56357-2_19

In this paper, the variables that we want to optimize are the weights of board cells. There are many board cells in a game board, and each cell has an important degree for game players. How to find the best weights of board cells based on the huge game records is an interesting problem. Evolutionary computing is an option of solving effectively for the optimization problems. From that, our problem can be formulated as finding a solution maximizing a criterion among a number of candidate solutions.

Othello is a popular game, its rules are simple. Moreover, we have already the high quality Othello game records. Thus, we select Othello as a testing environment for our research. Because of the symmetry of game board, so we need to find the weights of nine positions {A1, B1, C1, D1, B2, C2, D2, C3, D3} such that the winning probability of a selected move from data set of game records is maximal (see Fig. 1(b)), instead of weighting the board cells by experts (see Fig. 1(a)) [9]. Because the weighting by experts is not precise in complex board such as Shogi, Go. Besides, we have many levels of game records, the automatic weighting is very comfortable in adjusting the strong level of board game program.

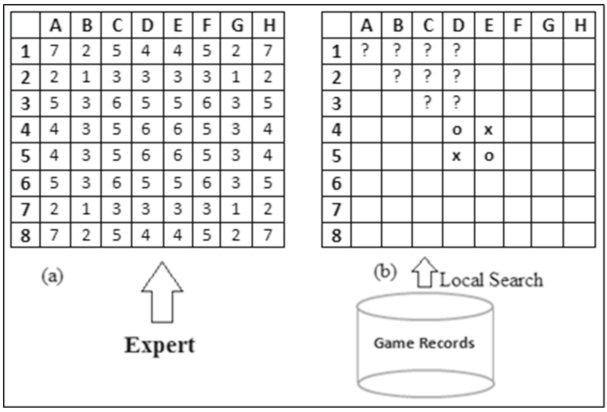

Fig. 1. The board cells to be optimized

This paper is organized as follows: Section 2 discusses related work about MCTS and knowledge encoding, Sect. 3 explains the problem description and our method in details, Sect. 4 presents experimental results and evaluating them, and Sect. 5 presents our conclusions and future work.

2 Related Works and Background

This section introduces some related works, and we focus on three things: (1) What is MCTS, (2) how to obtain knowledge from game data before adding it into MCTS, and (3) how an action evaluation function improve MCTS.

2.1 Learning from Game Records

There are many methods to extract knowledge from professional game records. A popular approach to automatically generate patterns is supervised learning such as a pattern extraction scheme for efficiently harvesting patterns of a given size and shape [3], using the relative frequencies of local board patterns observed in game records to generate a ranked list of moves [2], using a neural network approach to generate local moves [6], using the K nearest-neighbor representation to generate local moves [1], or automatic acquisition of tactical patterns for eyes or connections [7]. In this approach, expert knowledge is used to choose some relevant pattern shapes and pattern features, and then a machine-learning algorithm is used to find the patterns corresponding to these shapes and features, and then to evaluate them. The advantage of automatic pattern learning over hand-made patterns is that thousands of patterns may be generated and evaluated with little effort, and little domain expertise [4]. Also, Michael Buro proposed a Generalized Linear Evaluation Model (GLEM) [10] for building pattern-based evaluations. In this model, feature weights are optimized by using linear regression, and then GLEM combines automatic feature space exploration with fast numerical parameter tuning by building patterns from atomic features and assign pattern weights by linear regression. This model is suitable for algorithms that need an evaluation function such as minimax, alpha-beta, and negamax. In this paper, the knowledge to apply into the action evaluation function to enhance MCTS programs is the weight of board cells. The weight will be optimized by optimization process. It will search the best value of each cell such that all values of cells will be the most matching with the data set of game records.

2.2 Improving the MCTS with Static Knowledge

Monte Carlo Tree Search is a method of finding optimal decisions by taking random samples in the decision space. With static knowledge, MCTS can use an action evaluation function to enhance the quality of simulated games better than that of random simulations as well as the accuracy of selection policy.

Simulation Improvement: Probabilities and selections are two characteristics this step for the quality of simulated game. The selection in simulation phase is different with the selection step of MCTS, we can choose one of approaches such as Roulette-wheel, tournament selection, Reward-based selection, stochastic universal sampling, and Rank-base selection.

Progressive Strategy: The UCB + MCTS will be done accurately if the number of playouts is high. In case of the branching factor is high but the number of playouts is low, the UCB + MCTS will be inaccurate. Using a probability model with static knowledge to improve the move search more efficient is called progressive strategy. There are two progressive strategies: (1) Progressive widening first reduces the branching factor, and then increases it gradually to limit the number of search moves. (2) Knowledge bias uses knowledge to direct the search to give a bonus to make UCT

be more accurate if the number of playouts is low. Adding a bias in UCB formula is applied in several MCTS programs such as Erica and Zen. Our paper is based on the integration between knowledge action evaluation function with UCB formula [11]. There are many formulas, but the following formula is the sophisticated one.

$$UCB_{bias}(i) = \frac{w_i}{n_i} + C \times \sqrt{\frac{\ln n}{n_i}} + C_{BT} \times \sqrt{\frac{K}{n+K}} \times P(m_i) \quad (1)$$

where CBT is the coefficient to tune the effect of bias, and K is a parameter that tunes the rate at which the effect decreases. P(m_i) is the action evaluation function [11].

3 Problem Description and Our Method

The main purpose of this paper is finding the optimized weights of board cells. This is an optimization problem, and has three elements: (1) variables to be optimized, (2) the objective function, and (3) the optimization method. Figure 2 shows the optimization process of variables following an objective function.

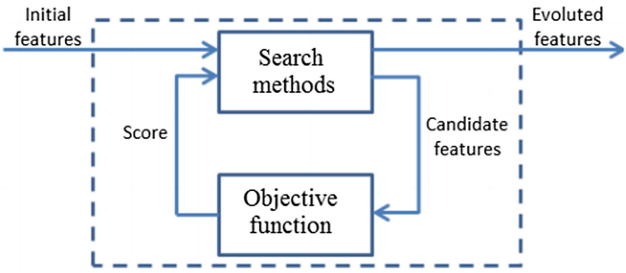

Fig. 2. Process of variables optimization.

3.1 Variables to be Optimized

In board games, the board cells have the different weights following the board game experts. However, the weighting from experts may be not optimal. Thus, we need an optimization method to find the optimal weights of board cells based on the game records. Beside the weights of board cells, there are many kids of other features such as patterns, local shapes, pattern shapes, etc. As the board of Fig. 1(b), we consider nine cells A1, B1, C1, D1, B2, C2, D2, C3, D3, these cells are the variables that need to be optimized in this paper. Let $x = \{x_1, x_2 \ldots x_m\}$ be the set of cell weights in the considered part of board. Element x_1 is the weight of board cell A1, element x_2 is the weight of board cell B1,.., element x_9 is the weight of board cell D3. The elements of x is called the variables that need to be optimized.

3.2 Objective Functions

To select the best move from legal moves based on the game records, we usually use some following formulas.

$f_1(H,x) = \frac{\sum_i^{\lvert H\rvert} prob(a_i^*)}{\lvert H\rvert}$	$f_2(H,x) = \frac{\lvert\{s,a^* \lvert prob(a^* < 0.1\}\rvert}{\lvert H\rvert}$
$f_3(H,x) = \frac{\sum_i^{\lvert H\rvert} rank(a_i^*)}{\lvert H\rvert}$	$f_4(H,x) = \frac{\sum_i^{\lvert H\rvert} \sum_{a \neq a^*} sigmoid(prob(a_i) - prob(a_i^*))}{\lvert H\rvert}$

where $prob(a_i^*) = \frac{value(a^*)}{\sum_{a \in A} value(a)}$ is a probability of a move on all legal moves in a state of board. In Fig. 1(a), we can calculate the probability of a move from the legal moves (B4, C1, F3, G2) in the state of board. The result are $prob(C1) = \frac{4}{3+4+6+1} = 28.57\%$, $prob(B4) = 21.42\%$, $prob(F3) = 42.85\%$, $prob(G2) = 7.14\%$

Then, probability of F3 is the highest, and the action evaluation function will select move F3.

Let H be a set of game records, | H | be the number of moves in set H. The move i^{th} is represented by a pair (s_i, a_i^*), where si is state of board at the move i^{th}. Let A be a set of legal moves in state *s*, $a \in$ A is a legal move, a* $\in$ A is a selected move. We see that the Eq. 2 is the most natural one, so it is selected in our method. From that, we can try can compare with other objective function by using Eqs. 3, 4, 5.

3.3 Optimization Method

Some following optimization methods can be used.

Random Search. Given a current solution x, if a new random solution x′ is better than current solution x, then $x = x'$. After many iterations, the solution is much better than the initial solution. The cost of this method is low, but the expected performance is not high.

Hill-Climbing. Given a current solution x, the new solutions are in neighborhood region of the current solution. If a new solution x′ is better than current solution x, then $x = x'$. After many iterations, the solution is much better than the initial solution. The cost of this method is higher than that of random search, and the expected performance is also higher. This method is always achieved the local optimum.

Simulated Annealing (SA). This method spends more cost than LS to overcome some traps of LS, then the global optimum can be achieved.

Genetic Algorithm (GA). It is difficult to compare the performance between SA and GA, but we can see the cost of SA is lower than GA in generating and evaluating the new solutions.

Brute Force. This method is useful if the search space is small. It is difficult to be used if the search space is large, and the cost of this method is always the highest.

3.4 Our Method

From three elements of optimization process, we can see that there are many methods that are combined by these elements. With the variables are vector x, our method uses the Eq. 2 as the objective function, Hill-Climbing or Simulated Annealing as an optimization method. We select Hill-Climbing or Simulated Annealing to balance between the cost and performance. Figure 3 shows the selected elements of our method.

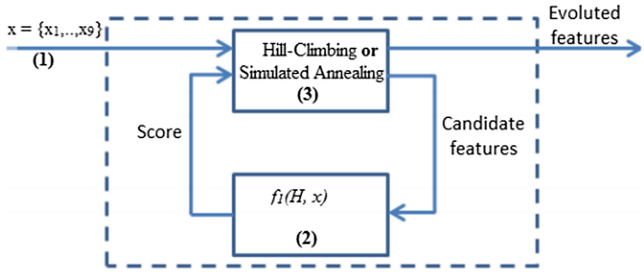

Fig. 3. Process of variables optimization.

Hill-Climbing procedure
Input: set the initial value of vector x, a set of game records H.
Output: the optimized value of vector x
1) The best value of x is the initial value
2) **repeat**
   a) Generate a neighbor x from x
   b) **if** $f_1(H, x) > f_1(H, x')$ **then**
   c) $x = x'$
3) **until** k iterations but no any x' be better than x
4) **return** x

Let $x = \{x_1 \ldots x_9\}$ be the variables to be optimized, $D = \{d_1 \ldots d_9\}$ be the neighborhood region of x, cho $d_i = x_i/10$. Let $x' = \{x'_1 \ldots x'_9\}$, such that $x_i - d_i \leq x'_i \leq x_i + d_i \cdot |H|$ is the number of moves in set of game records H. Each Riversi game record has maximum 60 moves, suppose that the average of each game records is 59 moves, set H has 1000 game records, so the total number of moves is 59000. Each selected move a^* leads to a new state of board, the board state includes the set of legal moves A. From that, we can calculate the probability for a selected move $prob(a^*)$ based on the weights of board cells. Average of 59000 probabilities is the value of objective function. After many iterations, the value of objective function is increased as the Fig. 4.

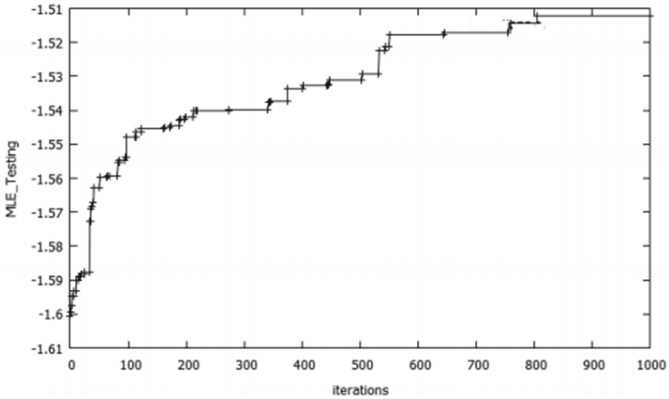

Fig. 4. An evolutionary behavior of Hill-Climbing.

Simulated Annealing Procedure
Input: Set the initial value of vector x, a set of game records H, T_{init}, T_{stop}, $gamma$
Output: the optimized value of vector x

1) $T = T_{init}$;
2) The best value of x is the initial value
3) **while** $T < T_{stop}$
   a) Generate a neighbor x' from x
   b) $\Delta = f_1(H, x) - f_1(H, x')$
   c) **if** $\Delta \leq 0$ **then**
   d) x = x'
   e) **else** x = x' with probability $e^{\Delta / T}$
   f) $T = T \times gamma$
4) **return** x

In Simulated Annealing, select randomly a state x' in the neighborhood region of x. If x' is better than $x(cost(x') > cost(x))$, then x′ is selected. On the contrary, the state x' is selected with some probability. The probability is decreased by the "badness of state x'. The probability depends on the temperature T. The higher the temperature T is, the more the bad state is selected. In searching process, temperature T decreases gradually to zero. When T is close zero, the behavior of Simulated Annealing looks like that of Hill-Climbing. The probability of selecting the bad state between x' and x is $e^{\Delta/T}$, where $\Delta = \cos t(x') - \cos t(x)$. Figure 5 is the evolutionary behavior of Simulated Annealing, it differs with evolutionary behavior of Hill-Climbing.

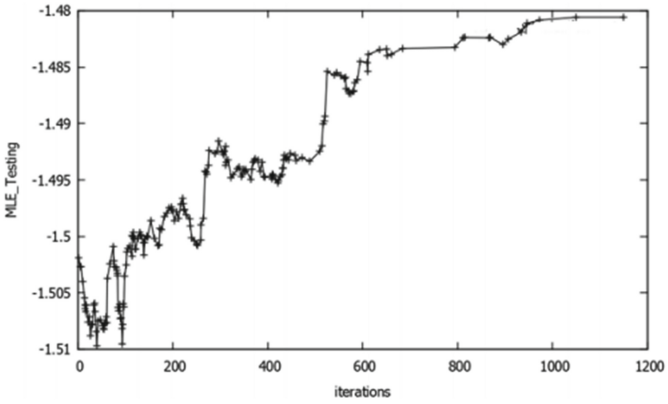

Fig. 5. An evolutionary behavior of Simulated Annealing.

4 Experiments and Evaluation

This section presents the experiments of Hill-Climbing, and Simulated Annealing. From that, we selected the best vector of weights, and applied it into the action evaluation function. The performance of Riversi MCTS is enhanced with the action evaluation function. The process of optimization was performed on game records played by strong players on a site of Michael Buro [5], author of Logistello program.

A. Experiment Results of Hill-Climbing
The parameters for Hill-Climbing:

- Initially, x = {496, 22, 161, 111, 4, 58, 39, 38, 89}
- The maximum iterations but no any candidate were found: 2000
- Number of game records: 480.000

Table 1. The evolutionary process of vector x

Iter	A1	B1	C1	D1	B2	C2	D2	C3	D3	$f_1(H,x)$
init	496	22	161	111	4	58	39	38	89	0.170907
1	503	17	161	88	8	44	59	61	114	0.172096
2	484	27	141	89	16	53	54	43	108	0.175772
4	507	13	149	86	18	43	49	35	92	0.177810
63	508	13	143	114	12	40	61	37	105	0.178249
162	506	23	138	112	17	35	52	39	108	0.178436
179	517	17	142	102	17	37	52	36	93	0.178454
631	510	26	148	113	10	35	60	36	108	0.178508
645	505	31	155	100	16	35	61	38	107	0.179525
967	498	20	144	90	17	38	64	36	103	0.180125
1350	502	16	153	106	18	36	64	35	106	0.180930
…										
3350	502	16	153	106	18	36	64	35	106	0.180930

Table 1 shows the evolutionary process of function f1. The value is evolved by Hill-Climbing method and stop at iteration 1350. We implement Hill-Climbing method many times, and Table 1 is the best one.

B. Experiment Results of Simulated Annealing The parameters for Simulated Annealing:

- Initially, x = {496, 22, 161, 111, 4, 58, 39, 38, 89}
- Number of game records: 480.000
- Initial temperature: T_{init} = 0.001
- Stop temperature: T_{stop} = 0.0001
- Coefficient of temperature decreasing: gamma = 0.999
- Number of iterations (for these parameters): 2301

The neighborhood candidate in Simulated Annealing procedure is changed two random elements in nine elements of vector *x*. The Simulated Annealing is implemented many times, and Table 2 is the top-4 data of many times.

Table 2. Top-4 Data of Simulated Annealing

Stop at	A1	B1	C1	D1	B2	C2	D2	C3	D3	$f_1(H,x)$
2301	458	416	255	196	10	73	73	38	97	0.182423
2301	464	459	277	212	11	66	69	36	85	0.182348
2301	501	317	302	144	7	55	68	74	96	0.181988
2301	492	343	449	222	4	77	98	81	63	0.181574

C. Applying Monte Carlo Tree Search Program The Monte Carlo Tree Search program in this paper is Riversi MCTS, we implemented a Riversi program based on Monte Carlo Tree Search instead of Alpha-Beta. Besides, we also have the other strong Riversi which is implemented by Alpha-Beta. The Riversi is downloaded from the Internet (http://www.codeproject.com/Articles/4672/Reversi-in-C), we use it for comparing with our program Riversi MCTS.

The best set of weights x = {458, 416, 255, 196, 10, 73, 73, 38, 97} is selected, this set is the variables that make the objective function have the highest value $f_1(H, x)$ = 0.182423. Apply the weights into the action evaluation function $prob(a_i^*) = \frac{value(a^*)}{\sum_{a \in A} value(a)}$, the Riversi_MCTS program is improved, Table 3.

Table 3. Performance of Riversi_MCTS before/after using action evaluation function

	Before	After
Riversi_MCTS	343	432
Riversi (expert level)	657	568
Result	657/1000 = 34.3%	568/1000 = 43.2%

5 Conclusion

This section reviews our work, summaries the experiment results, and discusses the future work. The first, we introduce a process of optimization which has three important elements: Variables to be optimized, the objective function, and the optimization method. This process is used to optimize the features from a given set of data. The feature is considered in our paper is the weights of board cells. These weights are applied in the action evaluation function to enhance the MCTS program.

In experiment, we use Hill-Climbing and Simulated Annealing as the optimization methods to balance the cost and the performance of experiments. From that, the best set of weights is found out. These weights are applied into the action evaluation function of our Riversi MCTS program. Performance of MCTS program is improved in comparing with other strong Riversi. However, the performance of Riversi MCTS is still not better than that of Riversi. We have a plan to study in other features instead of the weights of board cells. Besides, this method can be applied in other board games.

References

1. Araki, N., Yoshida, K., Tsuruoka, Y., Tsujii, J.: Move prediction in Go with the maximum entropy method. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (2007)
2. Stern, D., Herbrich, R., Graepel, T.: Bayesian pattern ranking for move prediction in the game of Go. In: Proceedings of the 23rd international conference on Machine learning, Pittsburgh, pp. 873–880 (2006)
3. Coulom, R.: Computing Elo ratings of move patterns in the game of Go. In: Computer Games Workshop, Amsterdam, Netherlands (2007)
4. http://skatgame.net/mburo/ggs/game-archive/Othello (2012)
5. Werf, E., Uiterwijk, J.W.H.M., Postma, E., Herik, J.: Local move prediction in Go. In: Schaeffer, J., Müller, M., Björnsson, Y. (eds.) CG 2002. LNCS, vol. 2883, pp. 393–412. Springer, Heidelberg (2003). doi:10.1007/978-3-540-40031-8_26
6. Cazenave, T.: Automatic acquisition of tactical go rules. In: 3rd Game Programming Workshop in Hakone, Japan, pp. 10–19 (1996)
7. Chaslot, G., Bakkes, S., Szita, I., Spronck, P.: Monte-Carlo tree search: a new framework for game AI. AIIDE 2008
8. http://www.apld.co.uk/riscworld/volume3/issue5/agrm/chap09.htm
9. Buro, M.: From simple features to sophisticated evaluation functions. In: Herik, H.J., Iida, H. (eds.) CG 1998. LNCS, vol. 1558, pp. 126–145. Springer, Heidelberg (1999). doi:10.1007/3-540-48957-6_8
10. Ikeda, K., Viennot, S.: Efficiency of static knowledge bias in monte-carlo tree search. Computers and Games 2013 (2013)