УДК 519.8

D. Klyushin, K. Kruchinin

ADVANCED SEARCH USING ALPHA-BETA PRUNING

D. Klyushin, K. Kruchinin. *Advanced search using Alpha-Beta pruning*, Matematychni Studii, **25** (2006) 108–112.

This paper presents a new algorithm of search of the best move in computer games like chess, the estimation of its complexity is obtained.

Д. Клюшин, К. Кручинин. *Эффективный поиск с использованием альфа-бета перебора* // Математичні Студії. – 2006. – Т.25, №1. – С.108–112.

В статье приведен новый алгоритм поиска наилучшего хода в игре, близкой к шахматам. Получена оценка сложности этого алгоритма.

Introduction. Most computer games like chess work by examining large game trees of fixed depth. It is necessary that search has been executed as soon as possible without loss of the information.

A.Bitman [6], the founder of well-known chess program "Kaissa", said: "There are two possible ways to help a machine to reduce search. The first is only technical – to use various heuristics and algorithms (like an alpha-beta pruning) without loss of the best decision. The second one consists in attempt to imitate the human behaviour and try to carry out selection of moves – candidates at least in some positions of a tree of game for making advanced search."

The purpose of this paper is to develop the reduced search tree by imitation thinking of the human (in the second way).

The classical work about the alpha-beta pruning is the paper of Knuth and Moore [1]. The majority of scientific works in the field of computer games like chess are directed to improvement of alpha-beta procedure. Among the basic directions it is possible to distinguish the followings: change in order of moves, creation of various auxiliary tables, etc. Some result of the done work is brought in T.Marslanda's paper [2].

Many recent works on the game theory (in particular, chess) offer training algorithms and use neural network for this. Among such papers it is possible to distinguish such as [4], [5]. An interesting training model (adjustment of estimated function) was proposed by G.Kendall and G.Vitvell [3] (Two functions from the set of the estimated functions are selected. Using these functions, two parties are played by results of which the function-winner remains in the set, and the lost function is excluded, and the mutated clone of the winner is inserted on its place. If the contest is a draw then both functions remain in the population but they are mutated. The procedure continues until population convergence is achieved.)

2000 *Mathematics Subject Classification:* 91A35.

Algorithm. The purpose of the chess program is to imitate thinking of the good chess player. From what does the human-chess player start to compute the moves? As a rule the first stage is an analysis of possible removes figures of the opponent, activization of figures, . . . , another positive moves. Then opportunities of the opponent in the given position are analyzed. As a result of these calculations the best move is done. On the basis of these reflections the algorithm is also constructed.
Algorithm:

1) The program starts to compute moves using alpha-beta search, missing its move (in other words, computation begins from a move of the opponent). Thus, the main threat of the opponent is defined. If some threats are present, we pass to item 2a. Otherwise, transition is carried out to item 3a.

2a) The computation of moves which protect from threat of the opponent (like protection of the figure which stay under fight, remove a figure from fight, cover a figure by another, etc.), begins using alpha-beta pruning. We pass to item 2b.

2b) Can the opponent put mate? If yes, then the protecting move is done (if it exists). If no, then we pass to item 3a.

3a) All positive moves or moves under the plan of the game are computed (positive moves are the moves bringing the material overweight (like remove figures or pawns of opponent) or positional overweight (like activization of figures), the victim is considered as a move under the plan of the game) using alpha-beta search. Moves bringing material overweight are examined on depth in 2-3 times bigger than bringing positional overweight or moves under the plan of the game (for example, in variants with removing figures of opponent it is important to what extent a figure is protected, because n moves can carry overweight, but the $(n+1)$-th move can deprive it). We pass to item 3b.

3b) If it is possible to put mate then the program puts mate. Otherwise, transition is carried out to item 4.

4) We select the best move among positive and protecting moves or moves under the plan of the game.

The note. When we use alpha-beta search it is possible to use almost all known modernizations of it (change the order of computation moves, creation of various auxiliary tables, progressive and iterative deeping, etc.)

Thus, as a result of performance of the algorithm all aimless moves (moves not bringing overweight, moves made not under the plan of game) will not be examined. This will essentially improve the algorithm of alpha-beta search (without using the algorithm described above). In fact, the best algorithm is the algorithm which visits the least number of nodes. Our algorithm does not demand sorting of moves with division into good and bad courses, it is enough to save at generation of the list of possible moves their advantage (benefit) (an estimation of only one program move without answer of the opponent, for example, a capture of a horse of the opponent is $+3$, a strong positional move is $+0.5$, etc. Good moves have advantage (benefit) $> 0.$xxx, where xxx defines on programming stage).

The lower bound of the algorithm. Let us assign to every position on level n a sequence of positive integer $a_1a_2a_3 \dots a_n$. For example, position 537 is received in the following way: in an initial position the fifth move has been made, then the third move has been made and then the seventh move has been made. We shall call a position *critical*, if $a_n = 1$ either for all even values of n or all odd values of n.

Let us consider a tree of game in which an estimation of an initial position is not equal to $\pm\infty$ and the first move in every position is the best. Then the alpha-beta pruning tests only the critical positions. From this statement it follows that if there is a tree of game where amount of branches (moves) from each node is equal W and depth is equal to D, the estimation of an initial position is not equal to $\pm\infty$ and the first move in every position is the best then the alpha-beta search estimates

$$V = W^{\lceil D/2 \rceil} + W^{\lfloor D/2 \rfloor} - 1$$

terminal positions (final positions which located at level (depth) D). Where $\lceil x \rceil$ is the smallest integer $\geq x$, $\lfloor x \rfloor$ is the largest integer $\leq x$ [1]. In other words, we computed V variants.

Let us consider the case, where in each position there are W moves. It is necessary to note that in average in a randomly selected position there are approximately 35 moves (de Groot, 1965). In other words, every move of the program generates about 35 branches (moves) of the opponent (W approximately is equal 35).

In our case in the first item of algorithm depth is equal to $D - 1$. So

$$V_1^{(0)} = W^{\lceil (D-1)/2 \rceil} + W^{\lfloor (D-1)/2 \rfloor} - 1.$$

Let us consider performance of items 2 and 3.

On the average among W moves there are less than half positive (not protecting) ones, protecting and moves under the plan of the game. We shall denote the amount of these moves by p. Then

$$V_2^{(0)} = p^{\lceil D/2 \rceil} + p^{\lfloor D/2 \rfloor} - 1.$$

The general amount of passes to nodes (computed variants) $V = V_1^{(0)} + V_2^{(0)}$, that is much less than $W^{\lceil D/2 \rceil} + W^{\lfloor D/2 \rfloor} - 1$.

It is necessary to note that the smaller amount of figures on the board the smaller set of "good" moves and, consequently, p will be less and the algorithm will work faster. Besides the algorithm reduce the amount of only the first moves (moves of the first level). If it is use recursively and for all moves of other levels in chains of computed variants, then

$$V_1^{(0)} = V_1^{(1)} + V_2^{(1)} = V_1^{(2)} + V_2^{(1)} + V_2^{(2)} = V_2^{(1)} + V_2^{(2)} + V_2^{(2)} + \cdots + V_2^{(D)} + 1,$$

where $V_2^{(k)} = p^{\lceil (D-k)/2 \rceil} + p^{\lfloor (D-k)/2 \rfloor} - 1$, since $V_1^{(0)}$ is the amount of estimated terminal positions of depth $D-1$, $V_1^{(1)}$ is the amount of estimated terminal positions of depth $D-2$.

$$V = V_1^{(0)} + V_2^{(0)} = V_1^{(1)} + V_2^{(0)} + V_2^{(1)} = \cdots = \sum_{k=0}^{D} V_2^{(k)} + 1,$$

where $V_2^{(k)} = p^{\lceil (D-k)/2 \rceil} + p^{\lfloor (D-k)/2 \rfloor} - 1$, the lower bound of the algorithm is $O\left(p^{\lceil D/2 \rceil}\right)$ (instead of $O\left(W^{\lceil D/2 \rceil}\right)$).

The upper bound of the algorithm. We have the tree of degree (the amount of branches from each node) d and height (depth) h. Terminal positions have random estimations. How many terminal positions will be examined at work of alpha-beta search? Let us denote the

answer to this question as $T(d,h)$. Then $T(d,h) < \mathrm{c}^*(d)(\mathrm{r}^*(d))^h$, where $r^*(d)$ is the largest eigenvalue of the matrix

$$M_d^* = \begin{pmatrix} \sqrt{p_{11}} & \sqrt{p_{12}} & \cdots & \sqrt{p_{1d}} \\ \sqrt{p_{21}} & \sqrt{p_{22}} & \cdots & \sqrt{p_{2d}} \\ . & . & . & . \\ \sqrt{p_{d1}} & \sqrt{p_{d2}} & \cdots & \sqrt{p_{dd}} \end{pmatrix},$$

where p_{ij} is the probability that $\max\limits_{1\le k<l}(\min(y_{k1},\ldots,y_{kd})) < \min\limits_{1\le k<j}(y_{lk})$ in the sequence of $(i-1)d+(j-1)$ independent identically distributed random variables $y_{11},\ldots,y_{l(j-1)}$, in other words whether the j-successor of research node will be examined. c^* is appropriate constant. Besides $\lim\limits_{h\to\infty} T(d,h)^{1/h} = r(d)$, where $c_1\frac{d}{\log d} \le r(d) \le c_2\frac{d}{\log d}$, c_1, c_2 are certain positive constants [1].

In our case in the first item of the algorithm depth is equal to $h-1$. So

$$V_1^{(0)} = T(d,h-1) < c^*(d)\,(r^*(d))^{h-1}\,.$$

As known, $\max\limits_{1\le i\le n}\lambda_i(A) \le \max\limits_{1\le i\le n}\sum\limits_{j=1}^{n}|a_{ij}|$, where λ_i are the eigenvalues of the matrix A.

Let us consider performance of items 2 and 3. It is W (approximately 35) moves in each position. Positive, protecting moves and moves under the plan of the game average hardly less than half of W. We shall denote the amount of these moves by p.

We construct the matrix M_d^{**} for our case

$$M_d^{**} = \begin{pmatrix} \sqrt{p_{11}^*} & \sqrt{p_{12}^*} & \cdots & \sqrt{p_{1d}^*} \\ \sqrt{p_{21}}^* & \sqrt{p_{22}^*} & \cdots & \sqrt{p_{2d}^*} \\ . & . & . & . \\ \sqrt{p_{d1}^*} & \sqrt{p_{d2}^*} & \cdots & \sqrt{p_{dd}^*} \end{pmatrix}$$

and denote the largest eigenvalue of M_d^{**} by $r^{**}(d)$.

In our case the bigger part of moves is not examined, consequently the bigger part of elements in each line $\sqrt{p_{ij}^*} = 0$, and consequently $\max\limits_{1\le i\le n}\left(\sum_{j=1}^{n}\sqrt{p_{ij}^*}\right)$ is much less $\Rightarrow$

$$r^{**}(d) \le \max_{1\le i\le n}\sum_{j=1}^{n}\left|\sqrt{p_{ij}^*}\right| < \max_{1\le i\le n}\sum_{j=1}^{n}\left|\sqrt{p_{ij}}\right| \le r^*(d)$$

$\Rightarrow$

$$r^{**}(d) < r^*(d)$$

$$V_2^{(0)} = T(d,h) < c^*(d)(r^{**}(d))^h$$

$$V = V_1^{(0)} + V_2^{(0)} = V_1^{(1)} + V_2^{(0)} + V_2^{(1)} = \cdots = \sum_{k=0}^{D} V_2^{(k)} + 1$$

The upper bound of the algorithm is $V_2^{(0)}$, that is less than $O\left(c^*(d)(r^{**}(d))^h\right)$, that in turn less than $O\left(c(d)(r^*(d))^h\right)$.

Experimental results. We took 10 different positions from chess party Botvinnik-Donner (Amsterdam, 1963) and 10 different positions from chess party Fisher-Matanovich (Lugano, 1968). The essence of the experiment is the following: first we input position to program with clear Alpha-Beta search and second we input position to program with our algorithm. Depth of search is 4. We receive the same results in both cases. But on the average Alpha-Beta search examined 11008 terminal positions and our algorithm examined only 331 terminal position on the average.

The following table contains the results of the experiments:

number	our algorithm	Alpha-Beta	number	our algorithm	Alpha-Beta
1	135	3326	11	225	10528
2	265	8203	12	274	10681
3	176	13090	13	349	9692
4	364	14712	14	307	12832
5	134	2923	15	368	9222
6	844	38058	16	686	12507
7	129	4305	17	544	11686
8	450	12073	18	161	9378
9	356	10995	19	163	5802
10	652	16357	20	36	3758

Conclusion. This paper presents a new algorithm of search of the best move in computer games like chess. The proposed technique is based on the Alpha-Beta pruning. Also lower and upper bounds of the working of algorithm have been presented.

Acknowledgement. We wish to thank T. Marsland for his cooperation.

REFERENCES

1. Knuth D.E., Moore R.W. *An analysis of Alpha-Beta pruning*, Artificial Intelligence, **6** (1975), 293–326.
2. Marsland T.A. Computer chess and search, Encyclopedia of Artificial Intelligence, 1992.
3. Kendall G., Whitwell G. *An evolutionary approach for the tuning of a chess evaluation function using population dynamics*, Proceedings of the 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, May 27-30, 2001, 995–1002.
4. Moriaty D.E., Shultz A.C. , Grefenstette J.J. *Evolutionary algorithms for reinforcement learning*, Journal of Artificial Inteligence Research, **11** (1999), 241–276.
5. Thrun S. *Learning to play the game of chess*, In Tesauro G. , Touretzky D., Leen T., editors, Advances in neural information processing systems 7, San Fransisko, Morgan Kaufmann, 1995.
6. Битман А. Игра, похожая на жизнь, Компьютерра, 2003.

National Taras Shevchenko University of Kyiv
Faculty of Cybernetics
ddddd19@rambler.ru
kruch@bigmir.net

Received 18.11.2004
Revised 22.12.2005