Experiments With The NegaC* Search An Alternative for Othello Endgame Search

Jean-Christophe WEILL

Université Paris 8 – Vincennes
Département d'informatique – Institut d'Intelligence Artificielle
2, Rue de la Liberté
93526 Saint-Denis (FRANCE)

Abstract

This paper presents experiments made by the author with the NegaC* search in Othello endgames. NegaC* search is mathematically analysed and compared to the α-β search and NegaScout in real game positions.

1 Introduction

Since the number of possible moves in a game is limited at Othello, it is possible to determine the game-theoric value of a position sufficiently close to the end of the game. Most Othello programs have two different searches [Lee and Mahajan, 1990]:

1. an *outcome* search : in fact an α-β search with a win/loss/draw evaluation function;
2. an *exact* search : an α-β search with a disc count differential evaluation function.

The first search is quite faster than the second. So Othello programs generally perform the first search and if enough time is left do the second one.

The main drawback in this scheme is that the second search does not use (or only a little) the results of the first search.

To bypass this, I adopted another algorithm: the NegaC* search (a negamax approach of the C* algorithm [Coplan, 1982]) in my program "Othello Master", which is, as far as I'm aware, the only program to use such a search.

2 α-β Search — Principal Results

In this section, I will recall the principal results of the standard search: the α-β search.

2.1 The α-β procedure

There are two usual approaches for searching a game tree. The *Min-Max* approach and the *Nega-Max* [Knuth and Moore, 1972] one. For this paper I chose to use the latter.

Figure 1 shows the α-β procedure using this approach.

```
integer procedure ALPHABETA(position P, integer alpha, integer beta)
  begin integer j, t, n;
  determine the successor positions: P1, ..., Pn;
  if n = 0 then
     ALPHABETA := f(P)
  else
    begin
    for j:=1 step 1 until n do
      begin
      t := - ALPHABETA(Pj, -beta, -alpha);
      if t > alpha then alpha := t;
      if alpha >= beta then goto done
      end;
 done: ALPHABETA:=alpha
    end
  end
```

Figure 1: The alpha-beta procedure from [Knuth and Moore, 1972].

2.2 The Expected Performance of α-β

First of all, some definitions:

A uniform game tree of degree w and depth d is a tree in which all internal nodes have exactly w sons, and all terminal nodes are at depth d.

A random game tree is a tree for which the values assigned to all terminal nodes are independent identically distributed random variables.

A directional algorithm is an algorithm with the following property: for some linear arrangement of the terminal nodes it never selects for examination a node situated to the left of a previously examined node.

We have the following Theorem:

Theorem 1 ([Baudet, 1978]) *Let $f_0(x) = x$, and for $i = 1, 2, \ldots,$, define:*

$$\begin{aligned} f_i(x) &= 1 - \{1 - [f_{i-1}(x)]^w\}^w, \\ r_i(x) &= \frac{1 - [f_{i-1}(x)]^w}{1 - f_{i-1}(x)}, \\ s_i(x) &= \frac{f_i(x)}{[f_{i-1}(x)]^w}, \\ R_i(x) &= r_1(x) \times \cdots \times r_{\lceil i/2 \rceil}(x), \\ S_i(x) &= s_1(x) \times \cdots \times r_{\lfloor i/2 \rfloor}(x). \end{aligned}$$

The average number, $N_{w,d}$, of terminal nodes examined by the α-β pruning algorithm in a random uniform game tree of degree w and depth d for which the bottom value are drawn from a continuous distribution is given by:

$$N_{w,d} = w^{\lfloor d/2 \rfloor} + \int_0^1 R'_d(t).S_d(t)dt$$

2.3 Minimal Window Search

It has been shown [Fishburn, 1983, Pearl, 1980, Reinefeld et al., 1985, Nagl, 1989] that the use of minimal windows improves the α-β search.

The main theorem is the following:

Theorem 2 *If $\alpha_0 \leq \alpha_1 < \beta_1 \leq \beta_0$, then the search $\alpha-\beta(P, \alpha_1, \beta_1)$ explores a sub-tree of the tree explored by the search $\alpha - \beta(P, \alpha_0, \beta_0)$.*

This theorem shows the motivations behind searching with a minimal window search, that is to search a smaller tree, so as to gain time.

2.3.1 Fail Soft α-β

The main idea is the use of an improved α-β search, the fail-soft-α-β [Fishburn, 1983] search showed in figure 2.

```
integer procedure IFALPHABETA(position P, integer alpha, integer beta)
  begin integer j, t, n, bestmove;
  determine the successor positions: P1, ..., Pn;
  if n = 0 then
     IFALPHABETA := f(P)
  else
    begin
    bestmove := -infinity;
    for j:=1 step 1 until n do
      begin
      t := - IFALPHABETA(Pj, -beta, -alpha);
      if t > bestmove then bestmove := t;
      if t > alpha then alpha := t;
      if alpha >= beta then goto done
      end;
 done: IFALPHABETA:=bestmove;
    end
  end
```

Figure 2: The fail-soft-alpha-beta procedure.

The main property of this search is that it can return values outside the α-β bounds without any extra work.

This leads to the following theorem:

Theorem 3 ([Nagl, 1989]) *We define by V, the negamax value of the root of a game-Tree, by α and β, the bounds of the window and by W, the value returned by fail-soft-α-β search then*

- *if* $\alpha < W < \beta$, *we have* $W = V$
- *if* $W \leq \alpha$, *we have* $V \leq W \leq \alpha$
- *if* $\beta \leq W$, *we have* $\beta \leq W \leq V$.

2.3.2 Search With Minimal Window

The fail-soft-alpha-beta procedure can be called with a minimal window, that is a window where $\beta = \alpha+1$. The previous theorem can be summarized in this case by:

Theorem 4 *We define by* V, *the negamax value of the root of a game-tree, by* α, *the lower bound of a minimal window search and by* W, *the value returned by the fail-soft-*α*-*β *search then*

- *if* $W \leq \alpha$, *we have* $V \leq W \leq \alpha$
- *if* $\alpha < W$, *we have* $\alpha < W \leq V$.

This shows that we can write a procedure which tells whether or not the negamax value of the root of a game-tree is greater or not than a value α. This is the base mechanism of some searches like Scout [Pearl, 1980] or C* [Coplan, 1982].

3 The NegaC* Algorithm

3.1 Definition

In [Coplan, 1982], Coplan proposed the C* search in order to replace the α-β search. In this section I reformulate the C* search in a Negamax approach which leads to a program structure conceptually simple and easy to implement in computer-game programs.

In fact, I show that the NegaC* algorithm is easy to implement using the fail-soft-alpha-beta search, see figure 3

The NegaC* algorithm search the *negamax* value of the root in a dichotomous way.

We need to know the bounds of the evaluation function, then using minimal windows it is possible in a dichotomous way to find the value of the root.

```
integer procedure NEGACSTAR(position P, integer min, integer max)
  begin integer v, t;
   while (min <> max) do
     begin v := (min + max) / 2;
       t := FAILSOFTALPHABETA(P, v, v + 1);
       if t > v then min := t else max := t;
     end;
     NEGACSTAR := t;
  end
```

Figure 3: The NegaC* procedure.
min and *max* are the bounds of the evaluation function.

For example, in an Othello endgame, the evaluation function returns values in the range $[-64, 64]$, the first value to be tested will be 0, and the first return will mean whether or not there is a winning move, modifying the window, and so on ...

Theorem 2 shows that the NegaC* algorithm explores a sub-tree of the tree explored by the α-β search. (The tree explored by the NegaC* search is a finite union of sub-trees of the tree explored by the α-β search)

3.2 An Upper Bound On The Expected Complexity of The NegaC* Search

The expected complexity of the call fail-soft-α-$\beta(P, v, v+1)$ First of all, we need to compute the expected complexity of a fail-soft-α-β search called with a minimal window on a game-tree of root J, width w and depth d.

We first assume that the evaluation function returns only two values -1 and 1.

Let P_0, the probability for the value of a terminal node to be 1.

Let P_d, the probability for the negamax value of the root of a depth d tree to be 1.

For a depth d tree, the value of the root is 1 if, and only if all the successors of the root are valued by -1, so

$$1 - P_d = P_{d-1}^w$$

.

Now, we compute the expected number x_d of terminal nodes explored by the call fail-soft-α-$\beta(P, -1, 1)$.

The probability to have a pruning[1] after the computation of the *kth* successor is $P_{d-1}^{k-1}(1 - P_{d-1})$. Such a possibility requires $(k-1)x_{d-1} + x_{d-1}$ computations of terminal nodes.

On the other hand, the probability for the value of J to be -1 is $1 - P_d$ and require wx_{d-1} computations of terminal nodes.

Then,

$$\begin{aligned}
x_{d-1} &= \sum_{k=1}^{k=w}(P_d^{k-1}(1-P_d)kx_d) + wx_dP_d^w \\
&= x_d(wP_d^w + (1-P_d)\sum_{k=1}^{k=w}(kP_d^{k-1})) \\
&= x_d\frac{1-P_d^w}{1-P_d} \\
&= x_d\frac{P_{d+1}}{1-P_d} \\
&= \prod_{k=0}^{k=d}\frac{P_{k+1}}{1-P_k}
\end{aligned}$$

If we now consider that P_0 is the probability for the value of a terminal node of the tree $\mathcal{J}$ to be greater or equal than a value V, we can modify the game-tree $\mathcal{J}$ in a game-tree $\mathcal{K}$ by the following process:

- If the value of a terminal node of $\mathcal{J}$ is greater or equal than $(-1)^d v$, then we assign to the correspondent terminal node of $\mathcal{K}$, the value 1.

- All the others terminal nodes of $\mathcal{K}$ are assigned the value -1.

Then it can be shown, by easy considerations, that fail-soft-α-$\beta(\mathcal{J}, v, v+1)$ explores the same tree that the call fail-soft-α-$\beta(\mathcal{K}, -1, 1)$. *The expected number of terminal nodes is therefore the same.* This leads to the following theorem:

[1] meaning the root J is winning

Theorem 5 *The expected number of terminal nodes explored by the call failsoft-α-$\beta(\mathcal{J}, v, v+1)$ for a uniform tree (w, d) is:*

$$N_{w,d} = \prod_{i=1}^{i=d} \frac{P_i}{1 - P_{i-1}}$$

where $P_i = 1 - P_{i-1}^w$ and P_0 is the probability that the value of a terminal node is greater than v.

A bound on the complexity of NegaC* If we call M, the number of possible values returned by the evaluation function, then we need a maximum of $\lceil \log_2 M \rceil$ call of fail-soft-α-β with a minimal window for a complete NegaC* search.

Moreover, it can be shown that $\frac{P_i}{1-P_{i-1}}$ is lower than $\frac{\zeta_w}{1-\zeta_w}$ where ζ_w is the smallest positive root of $x^w + x - 1 = 0$. This leads to the following result:

Theorem 6 ([Weill, 1989]) [2] *The expected complexity of the NegaC* search for a uniform tree (w,d) and a number of possible values returned by the evaluation function M is bound by*

$$\lceil \log_2(M) \rceil \left(\frac{\zeta_w}{1 - \zeta_w}\right)^d$$

where ζ_w is the smallest positive root of $x^w + x - 1 = 0$.

3.3 Comparison of the complexity of α-β search and NegaC*

The figure 4 shows the ratio between the expected complexity of the α-β as given by the theorem 1 with the complexity of the NegaC*.

So, we see that the NegaC* search is theorically better than the α-β search for game-trees where the depth is great and the number of the values returned by the evaluation function is relatively small.

[2] This shows also that NegaC* is asymptotically optimal over the class of directional game-searching algorithms

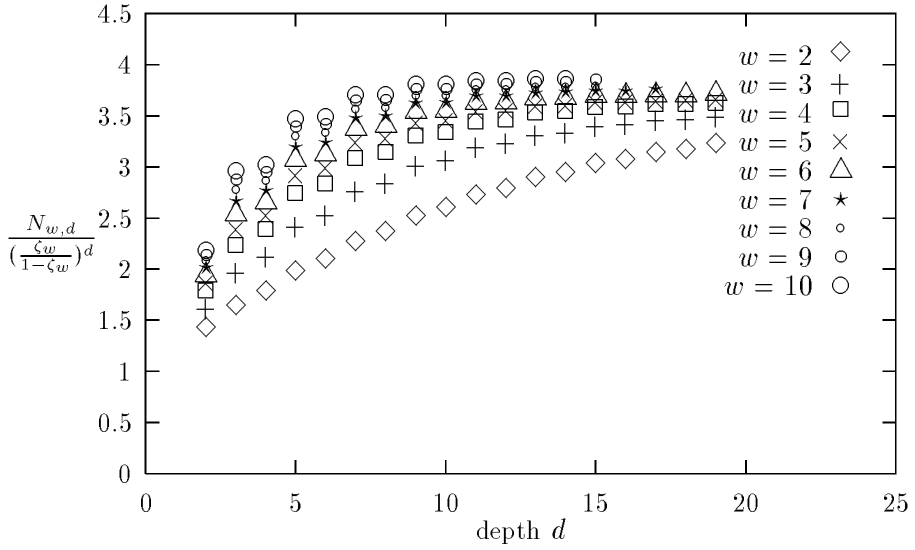


Figure 4: Ratio between the complexity of the α-β search with $(\frac{\zeta_w}{1-\zeta_w})^d$.

3.4 Refinements of NegaC*

We saw that the NegaC* algorithm explores a subtree of the tree explored by the α-β search, but some nodes are explored several times.

So, it would be helpful to save more information than just the fact of falling high or low. It can be done by modifying the fail-soft-α-β search, in the same way as was done for the NegaScout algorithm in [Reinefeld et al., 1985], see figure 5.[3]

Another possibility is to use transposition tables[4] to store informations, this is the best alternative because it handles transposition of moves.

---

[3] The search can be stopped when the root have only one successor left, then the exact *negamax* value of the root is unknow, but the best move is know like in [Nagl, 1989]

[4] also inappropriatly called hash-tables

```
integer procedure IFSAB(position P, integer alpha, integer beta)
  begin integer j, t, n, bestvalue;
  if P is all ready expansed then
  begin
     if P.min <= beta then begin bestvalue := beta; goto done; end;
     if P.max >= alpha then begin bestvalue := alpha; goto done; end;
     if P.min = P.max then begin bestvalue := P.min; goto done; end;
  end;
  else begin
    P is being expansed.
    attach the successor positions: P1, ..., Pn to P;
    P.min := -infinity;
    P.max := +infinity;
    if n = 0 then begin
       P.min := P.max := bestvalue := f(P);
       goto done;
    end;
  end;
  bestvalue = P.min;
  for Pj attached to P do
    begin
    t := - IFSAB(Pj, -beta, -alpha);
    if t > bestvalue then begin
      bestvalue := t;
      if t > alpha then alpha := t;
      if t >= beta or t=P.max then begin
        P.min := v;
        detach Pi where i < j;
        goto done;
        end;
      end;
    end;
  p.max := bestvalue;
  done: IFSAB := bestvalue
    end
  end
```

Figure 5: The informed-fail-soft-alpha-beta procedure.

4 Empirical comparison of NegaC*, NegaScout and α-β

In this section, I report experiments I made with Othello Endgame positions. The main aim was to compare the tree searches, α-β NegaScout [Reinefeld et al., 1985] and NegaC* in real endgame situations for my Othello program "Othello Master".

First of All, I will define the conditions of the experiments.

	C	A	B	B	A	C	
C	X	F	G	G	F	X	C
A	F	D	E	E	D	F	A
B	G	E	○	●	E	G	B
B	G	E	●	○	E	G	B
A	F	D	E	E	D	F	A
C	X	F	G	G	F	X	C
	C	A	B	B	A	C	

Diagram 1: squares types at Othello.

1. I chose 9 positions from High Level Human Othello Championship, 3 are taken from the World Championship 1985 and 6 of a Japanese Tournament [Ghirardato, 1982].
2. I chose to select the Move 47, so there are 14 moves left to play.
3. the evaluation is very simple, it is a disc count differential function.
4. the generation of moves is made in the following order: Corners,first then A-D-B-E-G-F-C-X squares [Frey, 1980, Kierulf, 1983] (see diagram 1).
5. α-β and NegaScout use the same killers as Chess 4.5 [Slate and Atkin, 1977].
6. the NegaC* algorithm uses a Partially Informed fail-soft-α-β, the positions are stored only near the root at depth 0,1,2,3,4,.. for reasons of memory-space.

The problem with Othello endgame searches, is that we want the fastest possible winning move if there is one, or else, the best-one.

So I chose to compute two parameters,

- The *outcome time* is the number of nodes needed by a search to find a winning move, if there is one, or else, it is the time needed to find the best move.

- The *exact time* is the number of nodes needed by a search to finish.

4.1 Exact Time

	Game 1	Game 2	Game 3	Game 4	Game 5	Game 6	Game 7	Game 8	Game 9
α-β	202955	123274	155307	176531	372265	256656	356434	332482	441883
NegaScout	138132	122270	145272	85505	457911	305004	202027	331460	346469
NegaC*0	115454	204761	209334	325389	337606	351304	434064	170550	615808
NegaC*1	104211	96079	197022	310905	218913	323993	434064	140889	459634
NegaC*2	92467	96079	188395	299326	217724	313089	397156	133638	442672
NegaC*3	75528	83388	175267	299626	215248	262076	392305	126589	423196
NegaC*4	75419	83388	167213	227837	213163	252479	371490	123165	419961

Table 1: Exact Number of Nodes for the Search

Table 1 shows that the results of the NegaC* search are very close to the resuts of α-β and NegaScout. We can see particulary that using an Informed-fail-soft-α-β, the NegaC* search completes searches in similar number of nodes.

	Winning Games	*Losing Games*	*Draw Games*	*Total Games*
α-β	1.00	1.00	1.00	1.00
NegaScout	0.78	0.88	1.23	0.87
*NegaC*0*	1.25	1.19	0.91	1.19
*NegaC*1*	1.17	0.92	0.59	0.99
*NegaC*2*	1.12	0.88	0.58	0.95
*NegaC*3*	0.91	0.82	0.58	0.83
*NegaC*4*	0.90	0.79	0.57	0.81

Table 2: Average Exact Time Normalized by α-β by Category of Games.

Table 2 shows that on average the time spent for the total search with an NegaC* using an Informed-fail-soft-α-β is almost better than the time spent by the α-β and NegaScout searches.

So in those examples, it is clear that NegaC* is a good improvement for Othello EndGame Search.

4.2 Outcome Time

	Game 1	Game 2	Game 3	Game 4	Game 5	Game 6	Game 7	Game 8	Game 9
α-β	185321	123274	155307	136389	372265	181350	200123	229162	141732
NegaScout	129386	122270	145272	63642	305567	213942	94240	281755	136170
NegaC*0	32921	150200	191000	40750	457911	42882	143620	150300	24405
NegaC*1	32921	83651	191000	40750	236324	42882	143620	130003	24405
NegaC*2	32921	83651	182310	40750	169201	42882	143620	125276	24405
NegaC*3	32921	78148	172365	40750	169201	42882	143620	120000	24405
NegaC*4	32921	78148	164000	40750	169201	42882	143620	119900	24405

Table 3: OUTCOME TIME

	Winning Games	*Losing Games*	*Draw Games*	*Total Games*
α-β	1.00	1.00	1.00	1.00
NegaScout	0.83	0.91	1.50	0.96
*NegaC*0*	0.22	0.96	0.77	0.61
*NegaC*1*	0.22	0.80	0.55	0.51
*NegaC*2*	0.22	0.78	0.55	0.51
*NegaC*3*	0.22	0.75	0.55	0.50
*NegaC*4*	0.22	0.73	0.55	0.48

Table 4: AVERAGE OUTCOME TIME NORMALIZED BY α-β RESULTS

Tables 3 and 4 show the NegaC* outperform α-β and NegaScout searches when considering the Outcome Time.

5 conclusion

It is clear, so, that NegaC* is the most suitable algorithm for Othello EndGame, even if the endgame is split in two different searches, by exemple like in Bill [Lee and Mahajan, 1990] or in Comp'Oth [Aguillon, 1986].

The time used in the *outcome search* (see section 1) is the same as that used by the first call to *IFAILSOFTALPHABETA*. So NegaC* uses the same time as a normal *outcome search* to see if there is a winning move.

Then it was shown by table 4 that the time needed to find the best move in the case of a losing game is less than the time needed by an α-β or a NegaScout search, so it is less than the time needed by an *exact search*.

Acknowledgment I would like to thank M.-F. Baudot and C. Weill for their proof reading and their many constructive comments.

A Positions Used In The Comparison

In the following positions, *W,L,D* denote the best moves, *W* means a winning move, *L* a losing move, *D* a move which leads to a draw.

The first three games are taken from the World Championship 1985, the last six games are taken from the Japanese Grand Master 1986.

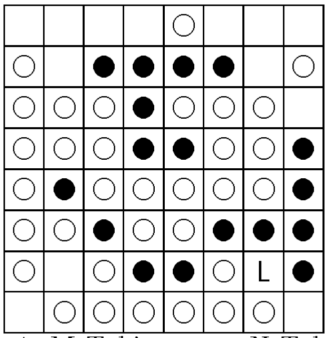
Game 1: Shaman vs Takizawa

	○		W	○	○	○	
		○	○	○	○		●
●	●	●	●	●	●	●	
●	●	●	○	○	●	○	○
○	●	○	●	●	○	●	○
○	○	●	●	●	●	○	○
○	○	●	●	●	●		○
		●	●	●			

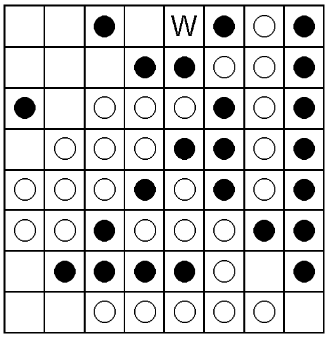
Game 2: Jones vs Takizawa

		●	●	●	L		
●		●	●	○	●		
●	●	●	○	○	●	●	●
●	●	●	○	○	●	●	●
●	●	●	○	○	●	○	●
●	●	●	○	●	●	●	●
		○	●	○	●		●
	○	○	○	○	○		

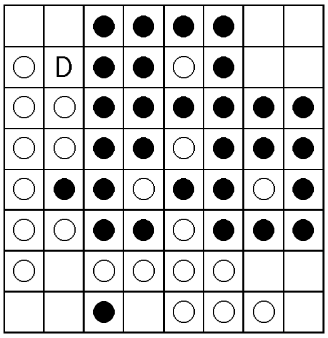
Game 3: Jones vs Kotkamaa



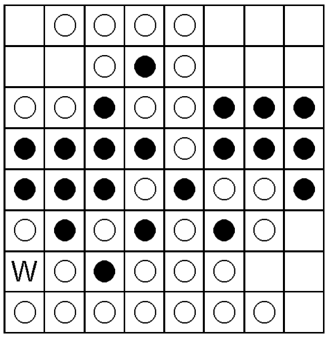
Game 4: M.Takizawa vs N.Takizawa



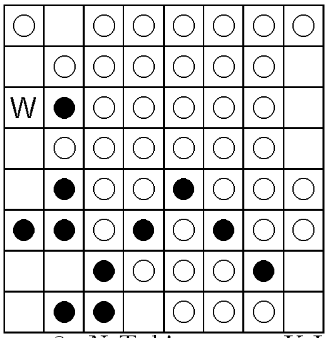
Game 5: Kaneda vs N.takizawa



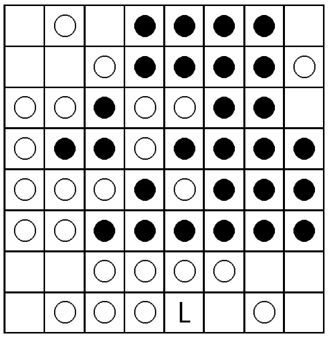
Game 6: N.Takizawa vs K.Ishii



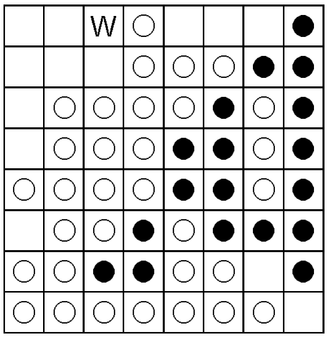
Game 7: K.Ishii vs N.Takizawa



Game 8: N.Takizawa vs K.Ishi



Game 9: K. Ishii vs N.Takizawa



References

[Aguillon, 1986] Aguillon, F. (1986). Le programme COMP'OTH. *FFORUM*, 6:8–11. (In French.).

[Baudet, 1978] Baudet, G. M. (1978). On the Branching Factor of the Alpha-Beta Pruning algorithm. *Artificial Intelligence*, 10(2):173–199.

[Coplan, 1982] Coplan, K. (1982). A Special-Purpose Machine For An Improved Search Algorithm For Deep Chess Combinations. In Beal, D. F., editor, *Advances In Computer Chess 3*, pages 25–43. Pergamon Press.

[Fishburn, 1983] Fishburn, J. P. (1983). Another Optimization of Alpha-Beta Search. *SIGART Newsletters*, 84:37–38.

[Frey, 1980] Frey, P. (1980). Machine Othello. *Personal Computing*, 4(7):89–90.

[Ghirardato, 1982] Ghirardato, P. (1982). Ishii colpisce : ancora! *Othello News*, 2(3):2–3. (In Italian.).

[Kierulf, 1983] Kierulf, A. (1983). Brand — an Othello program. In Bramer, M., editor, *Computer Game-Playing*, pages 197–208. Ellis Horwood.

[Knuth and Moore, 1972] Knuth, D. E. and Moore, R. W. (1972). An analysis of alpha-beta pruning. *Artificial Intelligence*, 3(3):145–163.

[Lee and Mahajan, 1990] Lee, K.-F. and Mahajan, S. (1990). The Development of a World Class Othello Program. *Artificial Intelligence*, 43:21–36.

[Nagl, 1989] Nagl, W. (1989). Best-Move-Proving: A Fast Game-Tree Searching Algorithm. In Levy, D. N. L. and Beal, D. F., editors, *Heuristic Programming in Artificial Intelligence*, pages 255–272. Ellis Horwood.

[Pearl, 1980] Pearl, J. (1980). Asymptotic Properties of Minimax Trees and Game-Searching Procedures. *Artificial Intelligence*, 14:113–138.

[Reinefeld et al., 1985] Reinefeld, A., Schaeffer, J., and Marsland, T. A. (1985). Information Acquisition in Minimal Window Search. In *Proceeding of The 9th IJCAI*, pages 1040–1043.

[Slate and Atkin, 1977] Slate, D. and Atkin, L. (1977). Chess 4.5 — The Northwestern University Chess Program. In Frey, P., editor, *Chess Skill in Man and Machine*, pages 82–118. Springer-Verlag.

[Weill, 1989] Weill, J.-C. (1989). Contribution à la programmation des jeux de réflexion. Mémoire de d.e.a., Université Paris 8 – Vincennes. (In French.).