Proceedings of the 5th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases, Madrid, Spain, February 15-17, 2006 (pp225-230)

Simulating a Human Playing Mastermind Introducing Noise in *Anti-Mind* Algorithm

JOSÉ BARAHONA DA FONSECA
Department of Electrical Engineering and Computer Science
New University of Lisbon
Monte de Caparica, 2829-516 Caparica
PORTUGAL
http://www.dee.fct.unl.pt

***Abstract*: -** When I started my career, in 1986, my main research interests were Artificial Intelligence(AI), Expert Systems and Decision Support Systems based on AI tools. The first experiment that I have done was the development of the *Anti-Mind* and *Master Mind with Feedback* programs written in the Basic language [1]. The *Anti-Mind* program simulates a *good* player of the Master Mind game, discovering the secret code defined by the human operator (a sequence of numbers in a pre-defined interval) very quickly. Then I used the algorithm of *Anti-Mind* to help and correct a human operator trying to discover the secret code defined by the computer resulting in the *Master Mind with Feedback.*

Let's take an example to clarify what I mean by the 'Computer *Thinks* better than the human' and *seems* to have a higher IQ:

Anti-Mind Program
CPC=Number of Correct Digits in Correct Position
CPE=Number of Correct Digits in Incorrect Position
3 Digits
Interval [0,3]

1. 103 CPC,CPE=1,1
2. 132 CPC,CPE=1,1
3. 120 CPC,CPE=0,2

Enough Information!
Secret code=?

The computer knows that the information is enough and it also knows the secret code. And you?

In this paper I will present the algorithms of *Anti-Mind* and *Mastermind with Feedback* with some worked examples and I will discuss, at the light of Cognitive Science, why is the computer a *better player* than the best human Mastermind players. Finally I will try to simulate a human player and his cognitive limitations introducing noise in the *good move* chosen randomly from the *good moves*. In the near future I am planning to introduce *logical processing limitations* simulated by discarding some previous moves and/or limiting the number of previous moves considered in the generation of the *good moves* coherent with them from which will be selected and altered the final move.

Key-Words: - AI, Cognitive Science, Simulation of Human Behaviour, *Anti-Mind* Algorithm, Simulating Human Cognitive Limitations Introducing Noise in *Anti-Mind*.

1 Introduction

During the last summer holidays I spent some time trying to put some order in my old books and I found my '89 CV [1] and when I read it I found the detailed description of Anti-Mind and Master Mind with Feedback programs. What I found interesting were some examples of the runs of that programs where the computer knows that the information was enough and knows the secret code but I had difficulties to reach the same conclusions.

Then I found another old book [2] which contains the Basic code of the Anti-Mind for the Sinclair 16k RAM ZX81 for secret codes with four digits varying between 1 and 6…Yes! They have done it without saving in memory the possible good moves coherent with the existent information, which makes the program very slow (they generate all the combinations till they find

one that is coherent with the actual information) and very difficult to understand. It is in this sense that my Anti-Mind program written in Basic in an Apple II with 48k of RAM memory is an original work: I didn't understand the Charlton *et al*'s algorithm behind a so tricky and '*spaghettian'* code and I invented the *Anti-Mind* algorithm.

There is an established idea that the Brain is a very powerful logic processor. I will show the contrary: the Brain is a very weak logic processor, and we have a great difficulty to combine (equivalent to logical conjunction AND) various incomplete informations like in the Mastermind game. The problem is that, if we don't repeat the digits in the first moves, the logical expressions that represent the possible hypotheses coherent with each move are more complex and much more their conjunction. For example, if the first move is

1333 cpc=1 cpe=0, the logical expression of the possible hypotheses coherent with this information is
(1,1)&(3,de) ⊕ (1,de) & [(3,2)⊕(3,3)⊕(3,4)] & (3 doesn't exist in position 1)
where ⊕ represents the exclusive or (XOR) logical operation and (i,j) means *digit i exists in position j* and (i,de) means *digit i doesn't exist*.

But if the first move is
0254 cpc=1 cpe=2, the logical expression of the possible hypotheses coherent with this information is much more complex (assuming a maximum digit of 6):

(0,1)& { (2,3)&(5,2)&(4,de)&[(3,4) ⊕(1,4) ⊕(2,4) ⊕(5,4) ⊕(6,4)] ⊕(2,3)&(5,4)&(4,de) &[(3,2) ⊕(1,2) ⊕(5,4) ⊕(6,4)] ⊕ (2,4)&(5,2)&(4,de)&[(3,3) ⊕(1,3) ⊕(5,3) ⊕(6,3)] ⊕ (2,3)&(4,2)&(5,de)&[(1,4) ⊕(3,4) ⊕(4,4) ⊕(6,4)] ⊕ (2,4)&(4,2)&(5,de)&[(1,3) ⊕(2,3) ⊕(4,3) ⊕(6,3)] ⊕(2,4)&(4,3)&(5,de)&[(1,2) ⊕(2,2) ⊕(3,2) ⊕(4,2) ⊕(6,2)] ⊕(5,2)&(4,3)&(2,de)&[(1,4) ⊕(3,4) ⊕(4,4) ⊕(5,4) ⊕(6,4)] ⊕(5,4)&(4,2)&(2,de)&[(1,3) ⊕(3,3) ⊕(4,3) ⊕(5,3) ⊕(6,3)] ⊕(5,4)&(4,3)&(2,de)&[(1,2) ⊕ (3,2) ⊕ (4,2) ⊕(5,2) ⊕(6,2) } ⊕

(2,2)& {…} ⊕

(5,3)& {…} ⊕

(4,4)& {…}

note that **A**⊕B=A&not(B) + not(A)&B

where + means logical OR.

Now imagine the conjunction of various expressions like this…only a very powerful logic processor would be capable to make the conjunction (logical AND) of various expressions so complex without getting lost…as it happens with us when we try to understand how the anti-mind algorithm reaches the conclusion that the information is enough and finds the secret code.

I have rewritten the anti-mind and master mind with feedback in matlab with some minor modifications in the algorithm and some more profound modifications in the implementation.

Since the mathematical analysis of the worst-case performance of the anti-mind algorithm in terms of the maximum number of moves for each combination of number of digits and maximum digit (assuming that the digit varies between 0 and digit_max) is very complex due to the random nature of my anti-mind, I have also made two programs anti_mind_auto.m and anti_mind_auto2.m that put the computer playing against itself for each possible secret code, selecting the more unfavourable situations of *enough information*.

I made some simulations with these latter programs but the results are not *reliable* since I used a number of repetitions with each secret code relatively small compared to the great number of possible combinations of good moves. For example [2] guaranteed that their anti-mind algorithm finds the secret code of 4 digits between 1 and 6 in no more than 9 moves, and my simulation points at a better worst case performance of 7 moves…but that doesn't mean that my algorithm is better, only that I didn't have access to a super computer!

2 JBF's Anti-Mind Algorithm

There are three main ideas behind my very simple anti-mind algorithm. The first one is to translate each move and its cpc and cpe not in a complex logical expression but in a set of *good* moves, that is, coherent with this information. The second one is that to select the subset from the actual good moves can be done simply considering the last move as the secret code and comparing it with the other good moves: the selected good moves must have cpc_i=cpc and cpe_i=cpe; this later condition guarantees that the selected good moves have cpc digits coincident with the last move and cpe digits that exist in the last move in different positions, that is, are *coherent* with the new information. The third one is that applying successively this rule of selection is equivalent to the conjunction (logical AND) of the logical expressions that define the good moves associated with each trial.

Before enunciating the algorithm in formal terms let's see an example, in order to acquire the intuition of what is going on.

My anti_mind.m has the syntax anti_mind(n_digits,max_digit, flag_trace, flag_n_h)

When we make flag_trace=1, the algorithm asks after each move if we want to see all the actual good moves, and if we

make flag_n_h=1, the algorithm will display the number of actual good moves before each trial. So let's consider 4 digits varying between 0 and 5, the secret code being 0535:

```
>> anti_mind(4,5, 1, 1)
Number of Hypothesis=1296
Move 1
Move=4  2  0  4
cpc=0
cpe=1
Number of Good Hypothesis=276
>>trace?0
Move 2
Move=2  1  5  3
cpc=0
cpe=2
Number of Good Hypothesis=52
>>trace?0
Move 3
Move=3  3  4  5
cpc=1
cpe=1
Number of Good Hypothesis=11
>>trace?
0  0  3  5
0  3  3  1
0  5  3  5
1  3  3  0
1  4  1  5
1  5  4  1
3  0  3  1
3  4  1  1
5  0  3  5
5  4  1  5
5  5  4  1
Move 4
Move=5  0  3  5
cpc=2
cpe=2
Number of Good Hypothesis=1
>>trace?0
**ENOUGH INFORMATION**, Secret Code:
0535
```

If you compare the selected 11 *good moves* with move 3, you see that each *good move* has only one digit coincident with move 3, since cpc=1, and only one digit that exists in move 3 but in a different position, since cpe=1; the remaining two digits don't contribute to cpc and cpe. I regrouped the referred 11 *good moves* to make more clear how the algorithm translates the *logic condition* in a set of moves (*de* means *does not exist)*:

```
Move 3
Move=3  3  4  5 cpc=1 cpe=1
[considering (3,1)->cpc=1 &(3,3)->cpe=1 &(the remaining positions 2,4 occupied by 0,1∉{3345})&(4,de)&(5,de)]:
3  0  3  1
[considering (3,1)->cpc=1 &(4,2)->cpe=1 &(the remaining positions 3,4 occupied by 1∉{3345})&(5,de)]:
3  4  1  1
[considering (3,2)->cpc=1 &(3,3)->cpe=1 &(the remaining positions 1,4 occupied by 0,1∉{3345})&(4,de)&(5,de)]:
0  3  3  1
[considering (3,2)->cpc=1 &(3,3)->cpe=1 &(the remaining positions 1,4 occupied by 0,1∉{3345})&(5,de)]:
1  3  3  0
[considering (4,3)->cpc=1 &(5,2)->cpe=1 &(the remaining positions 1,4 occupied by 5 and 1∉{3345}) & (3,de)]:
5  5  4  1
[considering (4,3)->cpc=1 &(5,2)->cpe=1 &(the remaining positions 1,4 occupied by 1∉{3345}) & (3,de)]:
1  5  4  1
[considering (5,4)->cpc=1 &(3,3)->cpe=1 &(the remaining positions 1,2 occupied by 0∉{3345}) & (4,de)]:
0  0  3  5
[considering (5,4)->cpc=1 &(3,3)->cpe=1 &(the remaining positions 1,2 occupied by 5 and 0∉{3345})& (4,de)]:
0  5  3  5
[considering (5,4)->cpc=1 &(4,2)->cpe=1 &(the remaining positions 1,3 occupied by 1∉{3345}&(3,de)]:
1  4  1  5
[considering (5,4)->cpc=1 & (3,3)->cpe=1 &(the remaining positions 1,2 occupied by  5 and 0 ∉{3345}) & (4,de)]:
5  0  3  5
[considering (5,4)->cpc=1 & (4,2)->cpe=1 &(the remaining positions 1,3 occupied by  5 and 1 ∉{3345}) & (3,de)]:
    5   4  1  5
```

After this digression I think you are just guessing my Anti-Mind algorithm which I will present in pseudo-code:

1. input(n_digits,max_digit)
2. n_good_moves=$(max\_digit+1)^{n\_digits}$
3. move_order=1
4. good_moves(1,1 to n_digits)=zeros(1,1 to n_digits)

```
5.  while n_good_moves > 1
                if  move_order = 1

move=round(random('uniform',0,max_digit, 1 to
                    n_digits))
                        else
            move=

good_moves(index(round(random('uniform',1,n_
good_moves)),1 to n_digits)
    display(['Move ', move_order, ' ', move])
    input(cpc,cpe)
    if cpc = n_digits
      display(' I got it!')
      break
    if move_order=1
     [good_moves,
index]=generate_good_moves(n_digits,max_digi
t,move,cpc,cpe)
   else
      [index,n_good_moves]=

select_good_moves(n_good_moves,good_moves,
index,move,cpc,cpe)
6.      if n_good_moves=1
display(['**Enough Information**
            Secret Code=',
good_moves(index(1),1 to n_digits)])
7.      if n_good_moves=0
display('**You Didn't Respect the Rules!**')
```

3 So, So You Think You Think Better than *Anti-Mind*?!

As another example of application of the *Anti-Mind* algorithm let's see how it found the secret code and reached the conclusion that the given information was enough for the case presented in the abstract:

1.103 CPC,CPE=1,1
Good Hypothesis:
131, 132, 133, 100, 110, 120,
001, 101, 201, 300, 302, 303,
023, 033, 213, 313
2.132 CPC,CPE=1,1
131 compared to 132->CPC,CPE=2,0; so 131 cannot be the Secret Code!
132 compared to 132->CPC,CPE=4,0; so 132 cannot be the Secret Code!
133 compared to 132->CPC,CPE=2,0; so 133 cannot be the Secret Code!
100 compared to 132->CPC,CPE=1,0; so 100 cannot be the Secret Code!
110 compared to 132->CPC,CPE=1,0; so 110 cannot be the Secret Code!
(etc)
120 compared to 132->**CPC,CPE=1,1**; so **120 can be the Secret Code**!
(etc)
302 compared to 132->**CPC,CPE=1,1**; so **302 can be the Secret Code**!
(etc)

So, it only remains 2 Hypothesis: 120 or 302

3.120 CPC,CPE=0,2

So it only remains **302** and this must be the Secret Code if the *CodeMaker* did not lie!

The anti-mind algorithm can be seen as a simplification of Donald Knuth's algorithm [4] where he chooses the *good move* not *randomly* from all *good moves* but the *move* that *minimizes* the *maximum number of next good moves for all possible combinations of cpc,cpe*. In the literature, after 1976, appeared a lot of works to solve the mastermind game, but nobody did beat the worst case performance of 5 moves of Donald Knuth's algorithm [5]-[16], and most of these works did not refer Donald Knuth's work!

4 Mastermind with Feedback

There are two ways to compare the human mastermind player performance with the anti-mind performance. The more obvious is to verify if the human player make good moves, that is, such that move$\in${good_moves(index(1:n_good_moves),1:n_digits)}. This is the main idea behind *Mastermind with Feedback* implemented by anti_mind_real.m that has the syntax anti_mind_real(n_digits, max_digit). The other way is to compare human's and anti_mind's worst case performances. In the following section I will present some simulations that, although not 100% reliable, give an idea of the very good worst case performance of my anti_mind algorithm..

Again, before presenting the formal description of *Mastermind with Feedback* algorithm let's see some examples with 4 digits and maximum digit 5 and flag_bad_moves=1, which means that the computer only accepts *good moves*, and while the move is considered *bad* (don't belong to {good_moves(i,j)}) the computer continues to ask for a *good move*:

```
>> anti_mind_real(4,5,1)
Number of Possible Good Moves=1296
move 1='1530'
cpc=1
cpe=0
Number of Possible Good Moves=108
```

move 2='1421'
cpc=0
cpe=2
Number of Possible Good Moves=19
move 3='2512'
Bad Move!
Do you want to see the good moves?1

Actual Good Moves
0 2 4 0
2 0 4 0
2 2 3 4
2 2 4 0
2 3 3 4
2 5 4 2
2 5 4 4
2 5 4 5
2 5 5 4
3 2 3 4
4 2 0 0
4 2 3 2
4 2 3 3
4 2 3 4
4 2 4 0
4 3 3 2
4 5 4 2
4 5 5 2
5 5 4 2
move 4='2545'
cpc=0
cpe=2
Number of Possible Good Moves=6
move 5='2542'
Bad Move!
Do you want to see the good moves?1

Actual Good Moves
3 2 3 4
4 2 0 0
4 2 3 2
4 2 3 3
4 2 3 4
4 3 3 2
move 6='4233'
cpc=3
cpe=0
Number of Possible Good Moves=2
move 7='4234'
cpc=4
cpe=0
**You Found It! in 7 moves, with 2 bad moves **

As you can see, my performance is not bad…for a human!

A *very good human* master mind player would be someone that don't make *bad moves*.
After this digression I think you are just guessing my *Master Mind with Feedback* algorithm that I will present in pseudo-code:

1. **input(n_digits, maximum_digit)**
2. **secret_code=generate_secret_code(n_digits, maximum_digit)**
3. **cpc=cpe=0**
4. **while cpc < n_digits**
   flag_bad_moves=1
   while flag_bad_moves
   move=input('your move')

flag_bad_moves=search(move,good_moves)
cpc=calculate_cpc(move,secret_code)
cpe=calculate_cpe(move,secret_code)

good_moves=select_good_moves(good_moves,secret_code,move,cpc,cpe)

5 Introducing Noise in the Chosen *Good* Move

In this first approach we try to simulate a human playing mastermind introducing noise in certain digits and in some moves, i.e. altering the chosen *good move* for certain moves. The *good move* is always coherent with all the previous moves and *cpc* and *cpe*.
Let's see an example for secret code 0005, where we introduce noise in all 4 digits varying between 0..5 and only in moves 2, 4 and 6:

1. 1223 CPC=0, CPE=0
From this move we can infer that 1, 2 and 3 do not belong to the secret code, but in the second move:
2. 5211 CPC=0, CPE=1
In this move only the 5 makes sense! From this information we know that there is at least one 5 in the second and/or third and/or fourth positions of the secret code.
3. 0445 CPC=2, CPE=0
Once this time we did not introduce noise this is a *good move*, i.e. coherent with all the previous. Now we know that there exist a 5 in the last position and a 4 in the second or third positions or a 0 in the first position.
The next move, where we will introduce noise, is totally nonsense since it is a permutation of the first move:
4. 2312 CPC=0, CPE=0
The next move, were we will not introduce noise, is obviously a *good move*:
5. 0055 CPC=3, CPE=0

Now with this new information we know that exists a 0 in the first or second position and two 5's or exists one 5 and two 0's.
So the *good moves* are only 0005! We do not need more information to get the secret code, but since in the next move we will introduce noise, we will not reach that conclusion:

6. 5152 CPC=0, CPE=1

Since in the next move we will not introduce noise, the algorithm will reach *our* conclusion:

7. ENOUGH INFORMATION! SECRET CODE: 0005

6 Conclusions and Future Work

Although a little bit simplistic, our approach is an original first step toward the modelling of a very complex cognitive task, an human playing the mastermind game.

In the near future we also plan to consider the limitations of working memory and logical thinking which could be modeled by a varying limit in the number of previous moves to be considered to generate the next *good move* that would next also be altered.

Another vector of possible evolution of our work could be the development of a complementary psychological test based on the *Mastermind with Feedback* algorithm from which we could calculate various scores characterizing the level of logical thinking errors.

References:

[1] J. Barahona da Fonseca, *Curriculum Vitae*, (written in Portuguese), May 1989.

[2] G. Charlton, M. Harrison, M and D. Jones, *The Turing Criterion : Machine Intelligent Programs for the 16K ZX81*, Interface Publications, 1982.

[3] N. A. Stillings, S. E. Weisler, C. H. Chase, M. H. Feinstein, J. L. Garfield, and E. L. Rissland, *Cognitive Science: An Introduction*, MIT Press, 1995.

[4] D. Knuth, "The Computer as Master Mind", *Journal of. Recreational Mathematics*, Vol. 9(1), 1976, pp. 1-6.

[5] Bento, L and Pereira, L, "Mastermind por Algoritmos Genéticos", in *Proceedings of Workshop on Genetic Algorithms and Artificial* Life, pp. 43-47, (written in Portuguese), DEEC, IST, Technical University of Lisbon, Portugal, 1996.

[6] Bento, L, Pereira, L and Rosa, A C, "Mastermind by Evolutionary Algorithms", *Proceedings of ACM SAC 99*, pp. 307-311, San Antonio, Brasil, 1999.

[7] Chvátal, V, "Mastermind", *Combinatorica*, 3:325-329, 1983.

[8] Flood, M M, "Mastermind Strategy", *Journal of Recreational Mathematics*, 18(3):194-202, 1985-86.

[9] Flood, M M, "Sequential Search Sequences with Mastermind Variants- part 1", *Journal of Recreational Mathematics*, 20(2):105-126, 1988.

[10] Flood, M M, "Sequential Search Sequences with Mastermind Variants- part 2", *Journal of Recreational Mathematics*, 20(3):168-181, 1988.

[11] Irving, R W, "Towards an Optimum Mastermind Strategy", *Journal of. Recreational Mathematics*, 11(2):81-87, 1978-79.

[12] Koyoma, K and Lai, T W, "An Optimal Mastermind Strategy", *Journal of. Recreational Mathematics*, 25(4):251-256, 1993.

[13] Neuwirth, E, "Some Strategies for Mastermind", *Zeitschrift fürOperations Research*, 26:B257-B278, 1982.

[14] Rada, R, "Mastermind in SIGART", *SIGART Newsletter*, 89:24-25, July 1984.

[15] Rao, T M, Kazin, G and O'Brien, D, "Algorithms to Play Mastermind", *SIGART Newsletter*, 95:33-35, January 1986.

[16] Shapiro, E, "Playing Mastermind Logically", *SIGART Newsletter*, 85:28-29, 1983.