is the case with MYCIN. For example, with a particular gift, the system will never recommend both outright and trust forms and let the user choose between them. However, TAXADVISOR can recommend conflicting courses of action based on different searches for recommendations. For example, the system can recommend that a client give his farm to his son and then, later in the consultation, recommend that he bequeath this same farm to his son.

Another peculiarity in TAXADVISOR is the method of backward chaining. In MYCIN each of the premises in a rule is traced in an attempt to make a diagnosis. In TAXADVISOR, no tracing is instituted from a rule until after the questions that the rule generates from its premises have been answered in the desired manner. For example, no search for a donee, form, etc. of a gift is instituted until after it is determined that a gift is desirable. A single gift control rule (often referred to in expert systems as a metarule) first determines desirability and then institutes tracing. In other words, different rule premises are involved with controlling the flow of the consultation (tracing parameters) and with domain knowledge (consultation parameters). As Aikins points out, this separation of control knowledge and domain knowledge is desirable because either can be altered without inadvertently affecting the other. This in turn makes a larger knowledge base possible because the system is much easier to change (Aikins, 1980).

The successful verification of TAXADVISOR implies a proper adaptation of system structure to the domain of tax planning. If this is so, the unique structural characteristics of TAXADVISOR mentioned above should be helpful to anyone planning to construct an expert sytems in a domain that may be similar to tax planning, such as other areas of taxation, law, and resource management, and may also be helpful to those constructing expert systems in unrelated domains.

Conclusion

As is the case when any field of inquiry is in its infancy, each new successful expert system sheds additional light on the domains that are amenable to this type of modeling, and on the manner in which expert systems should be developed and structured. Hopefully, this article is a useful addition to that body of knowledge.

References

Aikins, Janice, "Representation of Control Knowledge in Expert Systems," Department of Computer Science, Stanford University, 1980.

Hayes-Roth, Frederick, D.A. Waterman, & Douglas B. Lenat, "Principles of Pattern-Directed Inference Systems," *Pattern-Directed Inference Systems*, New York: Academic Press, Inc., 1978.

Melle, W. van, "A Domain-Independent Production-Rule System for Consultation Programs," Heuristic Programming Project, Department of Computer Science, Stanford University, 1979.

Michie, D., "Machine models of Perceptual and Intellectual Skills," from *Scientific Models and Man: The Herbert Spencer Lectures 1976*, (Edited by Henry Harris). Oxford University Press, 1979.

Shortliffe, E., *Computer-Based Medical Consultations: MYCIN*, New York, Elsevier, 1976.

Yu, V.L., L.M. Fagan, S.M. Wraith, W.J. Clancey, A.C. Scott, J. Hannigan, R.L. Blum, B.G. Buchanan, & S.N. Cohen, "Antimicrobial Selection by a Computer: A Blinded Evaluation by Infectious Disease Experts," *Journal of the American Medical Association*, Sept. 21, 1979, Vol. 242, No. 12, pp. 1279-1282.

Please turn to page 37 for Figure

AN ALGORITHM TO PLAY THE GAME OF MASTERMIND

by

T. Mahadeva Rao

Dept. of Math/Computer Science

SUNY College at Brockport

Brockport, NY 14420

1. INTRODUCTION

The game of Mastermind (INVICTA) is played by two persons. It is quite popular and claimed to be quite complex. Recently, we were interested in computerizing this game. This paper summarized the method we discovered and implemented. Our method uses a simple list data structure to represent the knowledge gained. The algorithm was programmed in Pascal on a Burrough B-6800 machine. Some experimental results are presented at the end of the paper.

There are several versions of the game - such as Master Mind, Mini Master Mind, Super Master Mind and so-on. We thus briefly explain the game we are considering, the Super Master Mind. The first of the two players is called the code-maker and the second is called the code-breaker. The code-maker makes a secret code and the code-breaker breaks this code. The accessories for the game consist of:

A decoding board with 12 rows of large holes (Code Peg holes); 12 rows of small holes (Key Peg holes); 5 shielded holes (for the hidden code); approximately 160 round headed colored pegs called "code pegs" (about 20 each of 8 colors) and approximately 40 small headed pegs called "Key Pegs" (about 20 each black & white).

At the beginning of the game, the code-maker sets up a row of five code pegs in the five shielded holes. For example suppose the secret code is

(1.1) (Blue, Blue, Red, White, Yellow)

The code-breaker will now try and duplicate the exact colors and positions of the secret code. Each time the code-breaker places a row of code pegs (they are then left in position throughout the game), the code-maker must give information by placing the black and white key pegs in the key peg holes alongside the code pegs. For example, if the first guess of the code-breaker is

(1.2) (White White White Yellow Yellow)

the code-maker would respond by:

Bulls = 1 Cows = 1

A Bull (represented by a black key peg) implies a code peg which is the same color and in exactly the same position as one of the code pegs behind the shield and a Cow (white key peg) means a hidden code peg which is matched in color but not in position by a peg placed by the code-breaker. In a later section we summarize an algorithm to compute the response.

A version of this game is available under the name "guess number game" on some microcomputers. But in these implementations, it is the computer that plays the code-maker and the human that plays the code-breaker. This author has seen an implementation in which the machine plays code-breaker. In this paper, we describe a method for programming the machine to play code-breaker.

Although the game is a two person game, it is not a zero sum game. It appears that the traditional approaches such as using a Static Evaluation Function and Alpha-Beta Technique [1,2,3] to select the next move are not applicable to this situation. We had to use a somewhat ad-hoc technique to mechanize this game. Our approach is to build a knowledge base of inferences drawn from each response of the code-maker.

2. Mathematical Formulation

It is quite easy to formulate the problem mathematically. Let the colors be represented by numbers, 1, 2, ... M. Let the positions be denoted by 1, 2, ... N. The code-maker makes a secret code, an N-digit number, where each digit i is a number 1 .. N. Thus the secret code has the format

(2.1) $C = (C_1 \quad C_2 \quad \ldots \quad C_N)$.

The code-breaker's trials have the format

(2.2) $G = (g_1 \quad g_2 \quad \ldots \quad g_N)$.

The code-maker responds with

(2.3) Bulls = b Cows = c

and so on. The code-breaker's purpose is to get N bulls in as few trials as possible.

Before we discuss the algorithms to make the next guess and update the knowledge base, it is important first, to understand how to compute the number of bulls and that of cows for a given trial arrangement. Since our algorithms use list structures to represent information, we shall use the list structure for illustration as well. Suppose the secret code and trial arrangement are:

(2.4) C = (1 2 3 4 5)

and

(2.5) G = (2 1 3 4 4).

We first compute the number of bulls, which is 2 in the example. The positions where we counted bulls are then deleted from the two lists to obtain

(2.6) $C^1 = (1 \ 2 \ 5)$

$G^1 = (2 \ 1 \ 4)$.

The number of cows is easily computed now as 2. We remember that it is the human who does this and so we do not program the computation of the number of bulls and cows.

3. An Example

Our algorithm explores the possibilities systematically. We first illustrate the algorithm using an example before formally presenting the algorithm. The steps of the algorithm are explained in detail in the illustration. The example will also help to convey the general philosophy behind the algorithm. We assume that M = 8 and N = 5. Suppose the secret code

(3.1) $C = (C_1 \quad C_2 \quad C_3 \quad C_4 \quad C_5)$

is unknown to us. We are playing the code-breaker. We have to start making the guesses somewhere, so we let it be

(3.2) $G_1 = (1 \quad 1 \quad 1 \quad 1 \quad 1)$

We learn something from the code-maker's response and store it in a knowledge base called "Inferences." Inferences is a list with the following structure:

(3.3)

$((C_1 \ (P_{11} \ P_{12} \ldots))$

$(C_2 \ (P_{21} \ P_{22} \ldots))$

... ...

$(C_k \ (P_{k1} \ P_{k2} \ldots)))$

After the first trial, the code-maker replies with

(3.4) Bulls = 0 Cows = 0.

This means to us (we are the computer) that the Color 1 is not present in the secret code. There is nothing so far in the Inferences list and our next move will be:

(3.5) G_2 (2 2 2 2 2)

The reply is

(3.6) Bulls = 2 Cows = 0.

We infer that there are two occurrences of the Color 2 in the secret code and our Inferences take the form

(3.7)

((2 (1 2 3 4 5))

(2 (1 2 3 4 5))).

In our next trial, we have two objectives: to try and find the right position for one of the Color 2's and try out a new color. We insert a 2 in the first possible position and fill the remaining positions with the next color. We use the terms - 'Being-Fixed' to denote a color which is known to be present and whose position is being determined; and 'Being-Considered' to denote the next color in the increasing order of colors. Our next trial then is

(3.8) $G_3 = (2 \quad 3 \quad 3 \quad 3 \quad 3)$.

The code-maker responds

Bulls = 2 Cows = 0

This implies that the Position 1 is correct for Color 1, for otherwise we should have had at least one cow. In future trials, Position 1 will always contain Color 2. We delete Positions 2, 3, 4, and 5 from the first sublist for Color 2, and Position 1 from all other lists. The fact that there are 2 bulls also implies that there is a 3 in the secret code somewhere in Positions 2, 3, 4, or 5. Thus we have:

((2 (1))

(3.9)

(2 (2 3 4 5))

(3 (2 3 4 5))).

We still have to fix the second Color 2. Thus Being-Fixed = 2 and Being-Considered = 4. In our next trial, we retain the Color 2 in Position 1, try the second Color 2 in Position 2, and fill up the remaining positions with Color 4. Thus our next trial will be

(3.10) G_4 = (2 2 4 4 4)

and there comes the response

(3.11) Bulls = 2 Cows = 1.

This tells us that there is one occurrence of the Color 4, and Position 2 is wrong for Color 2, for it it were correct there is no way the Color 4 could be wrong. It also follows that the Color 4 cannot be in Position 2. Thus the Inference takes the shape

((2 (1))

(2 (3 4 5))

(3.12)

(3 (2 3 4 5))

(4 (3 4 5))).

We still have not found the right place for the second Color 2, so Being-Fixed = 2 and Being-Considered = 5. Thus our next trial is

(3.13) G_5 = (2 5 2 5 5)

and the response is

(3.14) Bulls = 2 Cows = 0.

It now follows that the second Color 2 is in Position 3 (because of the two bulls). Also, there is no Color 5 in the secret code. Thus Being-Fixed will become 3, and Being-Considered becomes 6. Thus

(3.15) G_6 = (2 3 2 6 6)

is our next trial. We receive

(3.16) Bulls = 4 Cows = 0.

This tells us that the Color 3 is right in Position 3. We also conclude that there is a Color 6 in the secret code somewhere in Positions 4 and 5. Thus, the Inferences become

((2 (1))

(2 (3))

(3.17)

(3 (2))

(4 (4 5))

(6 (4 5))).

We know that we are quite close to the solution now. Being-Fixed is 4. We have discovered all the five colors. So, there is no need to try any new colors. We retain all the fixed colors in their positions. Being-Fixed will occupy the next possible position. The remaining positions will be occupied by the second color in the Inferences list which is not yet fixed. Thus our next trial will be

(3.18) G_7 = (2 3 2 4 6)

and we get the response

(3.19) Bulls = 5 and Cows = 0.

We halt, outputting the secret code G_7.

As the example illustrates, the algorithm tries out arrangements of colors in a systematic order and drawing all the possible conclusions at each trial. We are now ready to present the algorithms.

4. The Algorithm

The main loop of the algorithm is called Algorithm Mastermind and is presented here:

4.1 Algorithm Mastermind

```
Begin
 Setup; (* Initialize *)
 Try(trial, bulls, cows);
   (* display the trial code *)
   (* and accept human response *)
 While (bulls < N)
   and (not gameover) do
  Begin
   (*N= Numbers of positions *)
   Update (inferences);
   Getnext (trial);
   If (Numfix (inferences) = N)
   (* Numfix returns the number *)
   (* of positions which are tied to colors *)
    then gameover := true
    else Try (trial, bulls, cows)
  end;
 display (trial);
end. (* mastermind *)
```

The most important routine are Getnext and Update which are presented below:

4.2 Algorithm Getnext

The purpose of the algorithm is to construct the next trial arrangement. Let (g_1 g_2 ... g_N) be the arrangement it returns. We need the following routines:

1. Tied(i):boolean; Returns true if the position i has a color tied to it.
2. Itscolor(i) : colors; when i is a tied position returns the color to which i is tied.
3. Length(L) : integer; Returns the length, the number of top-level elements in a list L.
4. Nextpos(i) : positions; Returns the next possible position for color i. That is, if i=3 and the corresponding sublist is (3(2 4 5)) then it returns 2.
5. Secondunfixed(inferences) : colors; Returns the second color which is not yet fixed

```
Begin
 For i := 1 to N do (* N= Number of positions *)
  Case
   Tied (i)
    : g_i :=itscolor (i);
   (i = nextpos(beingfixed))
    : g_i := beingfixed;
   (Length(inferences) = N)
    : g_i= secondunfixed(inferences);
   else
    : g_i= beingconsidered
  end
end; (* Getnext *)
```

4.3 Algorithm Update

This algorithm updates the knowledge base. The routines we need are:

1. Addlists(gain, beingconsidered, inferences); Adds sublists (equal in number to gain) to the list inferences; each sublist with header 'beingconsidered'.

2. Fix(beingfixed); 'Fix'es the beingfixed in its next possible position and deletes appropriate position from other lists. For example, if beingfixed = 3, and inferences is

   ((3 (2 4 5)
   (4 (2 3 4 5))
   (7 (1 2 3 4 5)))

   then 'fix'ing of 3 would result in

   ((3 (2))
   (4 (3 4 5))
   (7 (1 3 4 5))

3. Bump(beingfixed); BeingFixed will get an updated value. In the above example, beingfixed will become 4.

4. Del(i,j); Deletes the current position of the color i form the sublist for color j. For example, if i=2, j=3 and the sublists are

   (2 (3 4 5))
   (3 (2 3 4 5))

   then Position 3 will be deleted from the second list, so that it becomes (3 (2 4 5)).

5. Fix1(i,j); fixes the color i in the current position of the color j. In the above example, if i=3 and j=2, then the color 3 gets tied to Position 3, its sublist becomes (3 (3)), and the position 3 gets deleted from other sublists.

6. Cleanup(inferences); cleans up the inferences list. For example, if inferences was

   ((2 (3))
   (4 (3 4 5))
   (5 (3 5)))

   cleanup will first transform this to

   ((2 (3))
   (4 (4 5))
   (5 (5)))

   and then finally to

   ((2 (3))
   (4 (4))
   (5 (5)))

7. Nextcolor(beingconsidered) : Gets a new color for being considered. If all the five colors have already been detected then beingconsidered is simply set to zero:

```
Begin
 if beingfixed = 0
  then gain := (bulls + cows)
   - Numfix (inferences) - 1
  else gain := (bulls + cows)
   -Numfix (inferences);
 case
  cows = 0
   : Begin
    (* beingfixed is OK in current position *)
    Fix(beingfixed);
    Bump(beingfixed)
   end;
  cows = 1
   : Begin
   (*beingfixed is not OK in current position *)
   (*beingconsidered is not OK in current position *)
   (*of beingfixed *)
   If (beingfixed <>0)
    then Del(beingfixed, beingconsidered);
   Del(beingfixed, beingfixed);
  end;
 cows = 2
  : Fix1(beingconsidered, beingfixed);
 else
  : Reporterror;
end; (*inferences);
cleanup(inferences);
Nextcolor(beingconsidered)
end; (* Update *)
```

5. A Computer Example

We present an example now, in which the machine guessed the secret code in 12 trials. For M=8 and N=5, this is the worst number of trials we have observed.

```
My Next trial : (1 1 1 1 1)
Enter Bulls Cows >> 0 0
My Next trial : (2 2 2 2 2)
Enter Bulls Cows >> 0 0
My Next trial : (3 3 3 3 3)
Enter Bulls Cows >> 0 0
My Next trial : (4 4 4 4 4)
Enter Bulls Cows >> 1 0
My Next trial : (4 5 5 5 5)
Enter Bulls Cows >> 1 1
My Next trial : (6 4 6 6 6)
Enter Bulls Cows >> 1 1
My Next trial : (7 7 4 7 7)
Enter Bulls Cows >> 1 1
My Next trial : (8 8 8 4 8)
Enter Bulls Cows >> 1 1
My Next trial : (6 5 6 6 4)
Enter Bulls Cows >> 2 1
My Next trial : (6 6 5 6 4)
Enter Bulls Cows >> 1 2
My Next trial : (7 8 6 5 4)
Enter Bulls Cows >> 3 2
```

Gotcha, your secret code is

(8 7 6 5 4)

And I took only 12 attempts.

6. Conclusion

From the systematic way in which the next trial arrangement is constructed it is clear that the number of Cows must be 0, 1, or 2. Simple arguments show that the conclusions drawn in each case are correct. It is interesting to find the worst number of trials for a given M (number of colors) and N (number of positions), called Mastersize. We have tried a large number of examples on our program and it has never taken more than 12 trials for M = 8 and N = 5. It appears that, in the worst case, the algorithm takes less than (M + N) attempts. It is quite pleasant to observe that repetitions in the secret code actually reduce the number of attempts.

References

[1] Nilsson, N., *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill (1971).

[2] Winston, P.H., *Artificial Intelligence*, Addison-Wesley (1977).

[3] Winston, P.H. and Horn, B.K.P., *Lisp*, Addison-Wesley (1981).

TURTLES AND DEFENSE

The following letter, which was the response to a request for information about the use of AI hardware for defense purposes, arrived through the ARPAnet after passing through many different sites (more than 4). We received permission from the authors to include it in the newsletter. They said that the time (3:05 AM) was the actual time the letter was written - *Ed.*

3:05am Tuesday, 19 January 1982

William Schubert
Center for Defense Analysis
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Dear sir:

We must admit to some initial puzzlement at receipt of your communique of 6 January regarding possible applications of our robotics and artificial intelligence products to military functions.

However, always eager to contribute to the defense of our country from the ever-present threat, we put our best minds right to work on the problem and put together the enclosed report. We hope it will aid in your analysis.

Please note that the information we are providing is to be used only in your analytical studies, and is *not* to be considered an official offer by Terrapin to supply the systems at the quoted prices.

Please call us at (617) 492-8816 if you have any questions.

Sincerely yours,

Patrick G. Sobalvarro
Leigh L. Klotz
Senior Software Engineers
Terrapin, Inc.

Introduction

At Terrapin, we feel that our two main products, the Terrapin Turtle®, and the Terrapin Logo Language[1] for the Apple II, bring together the fields of robotics and AI to provide hours of entertainment for the whole family. We are sure that an enlightened application of our products can uniquely impact the electronic battlefield of the future.

The Terrapin Turtle® is a small, versatile robot that can perform any number of complex tasks under computer control. A powerful AI programming language is necessary to realize the full potential of this advanced device.

The Terrapin Logo Language, developed at the Massachusetts Institute of Technology Artificial Intelligence Laboratory's Logo Group, is ideal for this application. The Logo language is a close relative of Lisp, the language used most widely in AI research. It fills the bill quite handily!

Descriptive Information

1. Functions the system might perform.

While somewhat limited in range,[2] Turtles show great promise as all-terrain, high-accuracy system with excellent survivability in hostile environments. The Turtle's low observability, low vulnerability to ECM, and its multidirectional sensing capabilities make it an ideal reconaissance vehicle. Its ability to perform complex terminal maneuvering while pushing a heavy payload makes it a superb delivery vehicle as well.

Survivability

The Turtle enjoys very low observability, due to a minimal radar cross-section and an almost non-existent infra-red signature.

[1]Logo trademark under license from Bolt, Beranek, and Newman, Inc.

[2]Extended-range Turtle Mark I variants are currently in testing, however. See the section on *Range*.