Playing Mastermind Optimally

Peter Norvig, U.C. Berkeley

There have been several recent articles in the SIGART Newsletter about the game of Mastermind (T. M. Rao, #82, E. Shapiro, #85, P. Koppstein, #88). This note describes an algorithm that is different from the others in that it is specifically designed to be an approximation to the optimal strategy.

Consider the version of Mastermind where the target is a sequence of n pegs, chosen (perhaps with repetition) from a set of m colors. At each turn, one player (player G) guesses a sequence of pegs, and the other player (player R) replies with the number of "bulls" (pegs of the right color in the right position) and "cows" (right color, wrong position). Rather than try to analyze how R makes the initial choice of a target sequence, we assume R is free to make any response to a guess, provided the response is consistent with all earlier responses. Thus, the game can be pictured as an and/or tree where at each level G has m^n possible moves, and R can have up to (n+1 * n+2)/2 moves. The optimal strategy for G is simply:

S: At each move, make the guess that minimizes the maximum number of guesses until the end of the game.

There are well-known techniques for exhaustively searching the game tree under this strategy, but unfortunately the size of the tree is prohibitively large for even small values of m and n. We are forced to make two simplifying assumptions: (1) never guess a sequence that cannot possibly be the target sequence, and (2) use the heuristic that the number of guesses needed to find the target is directly proportional to the number of possible sequences. The strategy thus becomes:

S': At each move, make the guess (from among the possible target sequences) that minimizes (over all responses by the opponent) the maximum number of remaining possible target sequences.

Note that there is no need to decide on a class of "inferences" to extract from the responses: the nature of the and/or tree guarantees that all information is used. Also note that assumptions (1) and (2), while reasonable in most cases, could lead to sub-optimal play. For example, consider the game with n = 4 pegs and the color set (R G Y B O P). Suppose it has been determined that the target sequence matches (R R R ?). Then, following assumption (1) we could take up to 6 more guesses to determine the target, but if we allow guesses like (R G Y B) and (R R O P) we will take only 2 or 3 turns. Similarly, assumption (2) can lead to sub-optimal play. Perhaps a hybrid solution that used strategy S' at the root of the tree and S nearer to the leaves would yield better performance with acceptable run time.

I have written a Franz Lisp program that generates a program to play Mastermind using strategy S'. In the case where n = 4 pegs and m = 6 colors, S' has a worst case of 6 guesses, and an average of 4.47 guesses.

Sample Games

```
-> (master-mind)
Select a sequence of length 4 from the colors:
      (G Y B R O P)
Reply with the number of exact, and then inexact matches.
(P G P G)? 0 2
(G P Y B)? 1 2
(O P G Y)? 0 3
(G O B P)? 2 0
(G Y R P)? 4 0
Aha!

-> (master-mind)
Select a sequence of length 4 from the colors:
      (G Y B R O P)
Reply with the number of exact, and then inexact matches.

(P G P G)? 0 0
(Y O Y B)? 0 1
(O R R R)? 2 1
(R B R R)? 3 0
(R Y R R)? 4 0
Aha!
```

Generated Code

The generated code for the function master-mind with 6 colors, 4 pegs is about 21 pages long, so instead I show the code for 2 colors, 3 pegs. In this version the worst case is 3 guesses, and the average is 2.3 guesses. To understand what the function "?" does, and how "master-mind" was generated, see the lisp code below.

```
(defun master-mind ()
   (print-instructions '(B W) 3)
   (? (W B B)
      ((0 2) (? (B W W)))
      ((1 2) (? (B W B)
                ((1 2) (? (B B W)))))
      ((1 0) (? (W W W)))
      ((2 0) (? (W W B)
                ((1 2) (? (W B W)))
                ((1 0) (? (B B B)))))))
```

Lisp Code

```
;;;; MASTER defines the function master-mind,
;;;; which in turn plays
;;;; the game Mastermind under the strategy S'

(defun master (colors n)
  (eval '(defun master-mind ()
     (print-instructions ',colors ,n)
     ,(partition (all-possibilities colors n)))))

;;;; PARTITION takes a list of possible sequences,
;;;; and finds the guess
;;;; that minimizes the maximum number
```

```
;;;; (over all possible replies) of
;;;; sequences that would still be possible.

(defun partition (possibilities)
  (cond ((length1? possibilities)
                '(? . ,possibilities))
   (t (let ((best-score 100000)
            best-guess score replies best-replies)
      (loop for guess in possibilities do
          (setq replies (give-replies guess possibilities))
        (setq score (apply #'max (mapcar #'length replies)))
         (cond ((lessp score best-score)
           (setq best-guess guess)
           (setq best-replies replies)
           (setq best-score score))))
      '(? ,best-guess .
          ,(loop for r in best-replies
          collect (list (car r) (partition (cdr r)))))))))

;;;; GIVE-REPLIES takes a guess and a list
;;;; of possible sequences and returns
;;;; a list of lists such that the first element
;;;; in each sublist is a reply
;;;; (that is a list of bulls and cows)
;;;; and the remaining elements of the
;;;; sublist are sequences that would lead to that reply.

(defun give-replies (guess possibilities)
  (let ((replies nil))
    (loop for possibility in possibilities do
      (cond ((not (equal possibility guess))
        (let* ((b&c (bulls&cows possibility guess))
          (lookup (assoc b&c replies)))
            (cond (lookup (push possibility (cdr lookup)))
             (t (push (list b&c possibility) replies)))))))
     replies))

;;;; BULLS&COWS gives the number of exact
;;;;  and inexact matches.

(defun bulls&cows (target guess)
  (let ((bulls 0) (cows 0) (target2 nil) (guess2 nil))
    (loop for a in guess for b in target do
      (cond ((eq a b)
        (setq bulls (1+ bulls)))
       (t (push a target2) (push b guess2))))
    (loop for a in guess2 do
      (cond ((memq a target2)
        (setq target2 (delq a target2 1))
        (setq cows (1+ cows)))))
    (list bulls cows)))

;;;; ALL-POSSIBILITIES returns a list of
;;;; all sequences of length n
;;;; whose elements are chosen from the set
;;;; colors, allowing duplicates.

(defun all-possibilities (colors n)
  (cond ((= n 1) (mapcar #'list colors))
   (t (loop for color in colors with results = nil do
         (loop for p in (all-possibilities colors (- n 1)) do
        (push (cons color p) results))
         finally (return results)))))

;;;; PRINT-INSTRUCTIONS tells the player what to do.

(defun print-instructions (colors n)
  (msg "You select a sequence of length "
       n " from the colors: " colors \N
       "Reply with the number of exact,"
        " and then inexact matches" \N \N))

;;;; The fexpr "?" makes a guess, reads a reply,
;;;; and dispatches on that.

(defun ? fexpr (args)
  (msg (car args) "? ")
  (let* ((bulls (read)) (cows (read))
    (program (cadr (assoc (list bulls cows) (cdr args)))))
   (cond ((equal bulls (length (car args))) 'Aha!)
         (program (eval program))
         (t 'Inconsistent))))
```

THE INTELIGENT SYSTEM

W. Fritz
Insituto de Investigacion en
Intelligencia Artificial
Uruguay 252 2 D, 1015 Bs. As
Argentina

ABSTRACT.

In this article, the brain is observed as a total system. A hypotesis of the various functions of the brain is set up and then partially tested through the workings of a computer program.

INTRODUCTION.

Today, much research is being done on various parts of A. I. such as picture recognition, knowledge representation, expert systems, natural language, but very little research is done on the total intelligent system as such.

We suspect that some manifestations, some aspects of intelligence, can only be demonstrated artificially, as a total system.

Therefore a hypotesis of the main functions of the brain is presented and a program shown, which embodies these functions. Running the program, the activity of the brain can be observed, and the results of that activity can be seen.

GENERAL CONSIDERATIONS.

In this paper, we use the word "function" as it is used in value analysis, and functional analysis. A function is the abstraction of how an object works, independent of the embodiment. Thus the function of an ashtray would be to hold ashes, independent of its shape and whether it is made of glass, metal or ceramics.