1327866

Cannon, Gregg Eugene

A MICROCOMPUTER TEST CONSTRUCTION PROGRAM APPLIED TO "CRACK THE CODE"

The University of Arizona M.A. 1986

University
Microfilms
International 300 N. Zeeb Road, Ann Arbor, MI 48106

A MICROCOMPUTER TEST CONSTRUCTION PROGRAM
APPLIED TO "CRACK THE CODE"

by

Gregg Eugene Cannon

A Thesis Submitted to the Faculty of the

DEPARTMENT OF EDUCATIONAL PSYCHOLOGY

In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF ARTS

In the Graduate College

THE UNIVERSITY OF ARIZONA

1 9 8 6

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowed without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Gregg Cannon

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Darrell Sabers
DARRELL L. SABERS
Professor of Educational Psychology

3/25/86
Date

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

ABSTRACT

J. P. Das and J. A. Nagalieri are currently working to develop a Cognitive Assessment System (CAS) as an alternative to traditional psychometric IQ tests. The CAS is based on the Luria-Das model of information processing. This model has several components, they are: Arousal, Coding (simultaneous and successive), and Planning. The game "Master Mind" is considered to be an excellent measure of planning. However, the standard game has several problems which prevent it from being used as a subtest in the CAS battery. In response to these problems the game was revised to become what is now called "Crack the Code".

After an initial tryout of Crack the Code it was discovered that some of the items had more than one solution. It was decided that human error must be eliminated from the procedure of generating items. The objective of this project was to write a computer program that would generate all unique items with only one solution.

CHAPTER 1

THE NEUROPSYCHOLOGICAL APPROACH TO MEASURING INTELLECTUAL ABILITY

The neuropsychological approach to measuring cognitive functions is becoming a popular alternative to traditional psychometric intelligence tests. The information-integration approach of J.P. Das is based on Luria's neuropsychological model (1966) of cognitive processes. One major difference detween the neuropsychological and the information-integration approach is the way in which tests are validated. The validity criterion for neuropsychological measures is based on the tests ability to discriminate between normal and neurologically impaired subjects; whereas tests based on the information-integration approach are validated in much the same way as traditional psychometric tests.

The shift to the neuropsychological approach is partially a result of the accumulation of evidence indicating cultural and socioeconomic bias with traditional tests (Jensen, 1980). The bias is thought to be largely due to the heavy loading in areas of scholastic achievement and language. As far back as 1951 Halstead noted that tests of biological intelligence were "relatively independent of cultural considerations and had a wide generality" (p.121). As the information-integration approach is based on a neuropsychological model, it is hoped that this approach will also provide a relatively unbiased measure of intellectual ability.

The Kaufmans (1983) developed the first popular alternative to intelligence tests with the K-ABC. Another alternative to traditional tests is being worked on by Das and Nagalieri now reported to be under contract with the Psychological Corporation to develop the Cognitive Assessment System (CAS) which they started with Merrill Publishing Company prior to the purchase of Merrill's Division of Tests and Services by the Psychological Corporation (personal communication, March, 10, 1986).

The Luria-Das model

The basis for the Cognitive Assessment System is the Luria-Das model of information processing. This model contains three major components: Arousal (attention), Coding (simultaneous and successive), and Planning. Simultaneous coding is thought to be quasi-spatial in nature, having the characteristic that all parts are immediately accessable. Successive coding on the other hand is temporal in nature. It is accessable only in a linear way. Both successive and simultaneous coding can be used to process verbal and nonverbal information, as well as information from any sense modality. Planning is described by Das and Heemsbergen (1983) as "...the selection and generation of programs, execution of programs, and evaluation of the executed action" (p.1). Because the popular game "Master Mind" is considered to be an excellent measure of planning (Das & Heemsbergen, 1983), Das asked Darrell Sabers to develop a test format that would tap the same processes as Master Mind. Sabers related the request to the author. With feedback from Das, Naglieri, and Sabers, the author revised the game to become what is now called "Crack the Code".

In Crack the Code there are four colors and three positions to place the colors. The subject is presented three rows of colors where no color is in the correct position. The subject's task is to generate a row of colors such that each color is in the correct position. There can be no duplication of colors in a given row.

A problem with Crack the Code

Using the format developed by the author, the test development staff at Merrill Publishing Company's Tests and Services Division and the authors of CAS produced items for an initial tryout of Crack the Code. This subtest was included in a battery of tasks and administered to 632 sixth and tenth grade students in Columbus, Ohio in September, 1985. During that tryout it was found that the item format was very efficient and that the average time per item was less than 30 seconds for these students. It was later discovered that one item developed for the first Crack the Code tryout had more than one correct response. This problem became the impetus for this thesis project: How can one be certain that there are no other correct solutions to an item?

The best way to check that there are unique correct solutions to each item is to try each possible combination of and show that only one fits the array. To do this for a large number of items presents a trying task, and when an employee of the Merrill Test Division was assigned the task, one item with at least two correct solutions was mistakenly allowed in the test. It was decided that human error must be eliminated from the procedure.

As originator of the task which is now known as Crack the Code, the author became the logical individual to take responsibility for producing the procedure for checking the items. Instead of developing a program to simply check previously-developed items, the resulting program generates all possible items and checks to ascertain that each has only one solution.

Objective of the project

The objective of the project was to develop a program that would generate and check items with one unique solution. A subgoal of the project was to produce all unique items to fit certain specifications, namely four colors, three rows, and three columns with a score of zero for each of the initial three rows.

CHAPTER 2

LURIA'S CONCEPTION OF BRAIN ORGANIZATION AND FUNCTIONS

The Cognitive Assessment model is based to a large extent on Luria's conception of brain organization and functions (1966). The scheme postulates three blocks corresponding to functional systems of the brain. The first block regulates body functions (e.g., arousal and attention). Anatomically it is thought to correspond to lower brain regions including the brain stem and midbrain. The second block is involved with input, recall, and the storage of information (coding). Coding is of two types, successive and simultaneous. Luria's hypothosis is that successive processes are associated with the frontotemporal regions of the brain, and simultaneous processes are associated with the occipital-parietal areas of the cortex. The third block of the brain is thought to be involved with more complex human behavior such as planning and decision making. There are several components of planning (Das & Heemsbergen, 1983). These components include: "...generation of hypotheses, selecting and testing hypotheses, and the evaluation of feedback" (p.9). The third block of the brain is thought to involve in the frontal lobes.

Luria's conception of brain functions has several parallels with that of Reitan (personal communication, April, 20, 1984). Reitan postulates three hierarchical levels of brain function. Reitan's first level of brain functions is congruent with Luria's block one. At the second level of brain functions Reitan makes the distinction between right and left hemisphere functions (verbal & nonverbal), whereas Luria makes the distinction between modes of processing (successive, & simultaneous). It appears that both hemispheres are capable of processing information either successively and/or simultaneously depending on the demands of the task. There is some evidence that the left hemisphere is more involved with successive processing, and the right hemisphere more simultaneous processing (Bogen, 1969; Levy, 1972; Nebes, 1974; Naglieri, Kamphaus, & Kaufman, 1983). Reitan would probably not argue the importance of the frontal lobes at the third level, but would emphasize the importance of many other cortical areas in higher mental functions, such as planning.

Planning dependent upon lower levels of processing

The view that higher cognitive processes are dependent on lower levels of processing was supported by data collected from Halstead's Category Test. All of the elements of planning enumerated by Das and Heemsbergen (1983) are contained in the Category Test. Halstead (1951) believed that the Category Test was specifically sensitive to the functions of the frontal lobes, and "It will not detect reliably primary brain lesions occurring elsewhere" (p. 123). However, Doehring and Reitan (1962) demonstrated that the Category Test was sensitive to lesions elsewhere in the brain, and that groups with lateralized cerebral lesions did not differ significantly in total errors on the test. It appeared that "...concept attainment on the Category Test may require the combined application of a number of primary abilities, both verbal and nonverbal" (p.32). That is, complex tasks such as the Category Test seem to involve the integration of several primary abilities, including subordinate processes such as coding and attention.

Although higher cognitive functions such as planning are dependent upon lower levels of processing it is possible to separate planning from coding. This is only possible because of the additional variance above that which can be accounted for by coding alone. Additional variance must be taken into account by other factors. It is possible to measure planning independent of coding by utilizing tests which load high on planning and low on coding. Loadings can be determined through factor analysis as was done by Das and Heemsbergen (1983).

The fact that planning requires the integration of many lower level abilities is what makes tests of planning so valuable. The lack of tests which measure planning is a major weakness with most intelligence batteries. For this reason it is very important to have a good measure of planning included in the CAS battery.

CHAPTER 3

MASTER MIND AND CRACK THE CODE

Master Mind

There are many versions of the game Master Mind which allow for adapting the difficulty of the game to the skill level of the players. The game was developed by Parker Brothers as a game of strategies. In the standard version of the game there are two players, a codemaker and a codebreaker. The codemaker's task is to secretly place four colored pegs in the four holes at the end of the decoding board. The pegs are hidden from the codebreaker. The codemaker can use any combination of six colors to fill the holes. The task of the codebreaker is to reproduce the same colors in the same positions on the board. A move is made when a row is filled with pegs. After each move the codemaker provides feedback in the form of small black and white (key) pegs. A white peg denotes a correct color in the wrong position, a black peg denotes a correct color in the correct position. There can be as many as four key pegs placed per row. There are ten rows or opportunities for the codebreaker to reproduce the code. The score is simply the number of moves it took for the codebreaker to reproduce the code.

Problems with Master Mind

There are several problems with Master Mind as a measure of planning. They are:

a) The difficulty of the code depends on the skill of the codemaker in fooling the codebreaker.

b) The amount of feedback varies from subject to subject, depending on the codebreaker'smoves.

c) The game takes too long for inclusion in a test battery.

d) The game is copyrighted.

Crack the Code

Crack the Code was designed to be administered in a standardized manner to produce a measure of planning in a relatively short period of time. Crack the Code differs from Master Mind in several respects. First it presents a game that has already progressed to its last move. All the information needed to produce the correct code is given. The examinee is given only a short period of time to respond with one row of colored chips. A second difference between Master Mind and Crack the Code is that there are fewer colors in Crack the Code. The number of colors is always one more than the number of columns. Since there are fewer colors from which to choose, a correct move is indicated only if the correct color is in the correct position. That is, unlike Master Mind, no credit is given for a correct color in the wrong position. In addition, there can be no duplication of colors in a row with Crack the Code.

The following is an example of an item which might be presented to an examinee:

RED	-	BLUE	-	GREEN	0
GREEN	-	YELLOW	-	RED	0
BLUE	-	GREEN	-	YELLOW	0
?	-	?	-	?	3

All the entries in the first three rows are wrong, as indicated by the zeros to the right of each row. Since there are only four admissible colors and no duplication of colors in a row, the response YELLOW - RED - BLUE would result in three correct placements, as indicated to the right of row four.

In order to make the items more challenging, an attempt was made to duplicate some colors in some columns. The following is an example of such an item:

RED	-	BLUE	-	GREEN		0
BLUE	-	GREEN	-	YELLOW		0
RED	-	YELLOW	-	RED		0
?	-	?	-	?		3

It should be noted that for column two the correct response is still RED, and for column three the correct response is still BLUE. There are two possible responses for the first column GREEN or YELLOW. As there are two possible responses to this item it is considered inadmissible. A better item would be:

BLUE	-	YELLOW	-	RED	0
YELLOW	-	GREEN	-	BLUE	0
BLUE	-	RED	-	YELLOW	0
?	-	?	-	?	3

For column two the correct response must be BLUE and for column three it must be GREEN. For column one there are two possible responses GREEN or RED. Since GREEN must be used in column three, the correct response to column one must be RED. This item has only one possible solution so it is considered an admissible item.

Crack the Code can be thought of as consisting of three types of admissible items: items with no repeats in any of the columns, items with repeats in one column, and items with repeats in two columns. It is not possible to have repeats in three columns as this would result in the item having more than one correct solution.

CHAPTER 4

METHODS AND RESULTS

The author had some previous experience in writing similar types of programs. One program in particular involved the eight queens problem. The task was to place eight queens on a chess board where no queen could attack another. The program was written to find all possible solutions to the problem. There were over 100 solutions. These solutions included mirror images, and various flips of the board. The eight queens problem was similar to the Crack the Code problem in that many combinations had to be checked, a task for which the computer is well suited. Both programs used similar types of procedures. One difference between the two programs was that with eight queens all permutations were checked, and with Crack the Code only unique solutions or combinations were checked. That is, mirror images and flips were not considered unique items to the Crack the Code problem.

Overview of the program

The program was written in PASCAL then compiled and run on a Tandy 1000 PC. For programing reasons colors were converted to integers. The main procedures were: pattern_generator, pattern_check, domain_generator, item_generator, item_check, and print_item. Figure 1 presents a flow chart of the main procedures. There were 11 procedures in all.

The first step in generating items was to generate all possible rows or patterns (pattern_generator). Then each pattern was checked (pattern_check) for inclusion in the domain. The domain contained only those patterns where there was no match with a particular solution (e.g.,1-2-3). For example the pattern [2-4-1] would be included in the domain as there are no matches with the solution pattern[1-2-3]. On the other hand, the pattern [3-2-4] would not be included in the domain since 2 in column two matches the solution and would result in one correct placement. Construction of the domain was accomplished by the domain_generator.

Once the domain was established items could be generated (item_generator) such that a particular pattern was always a solution. To make sure there were no other solutions a procedure (item_check) checked all remaining patterns. If another solution was found the item was thrown out. If the item had only one solution that item was printed (print_item), and counted as an admissible item.

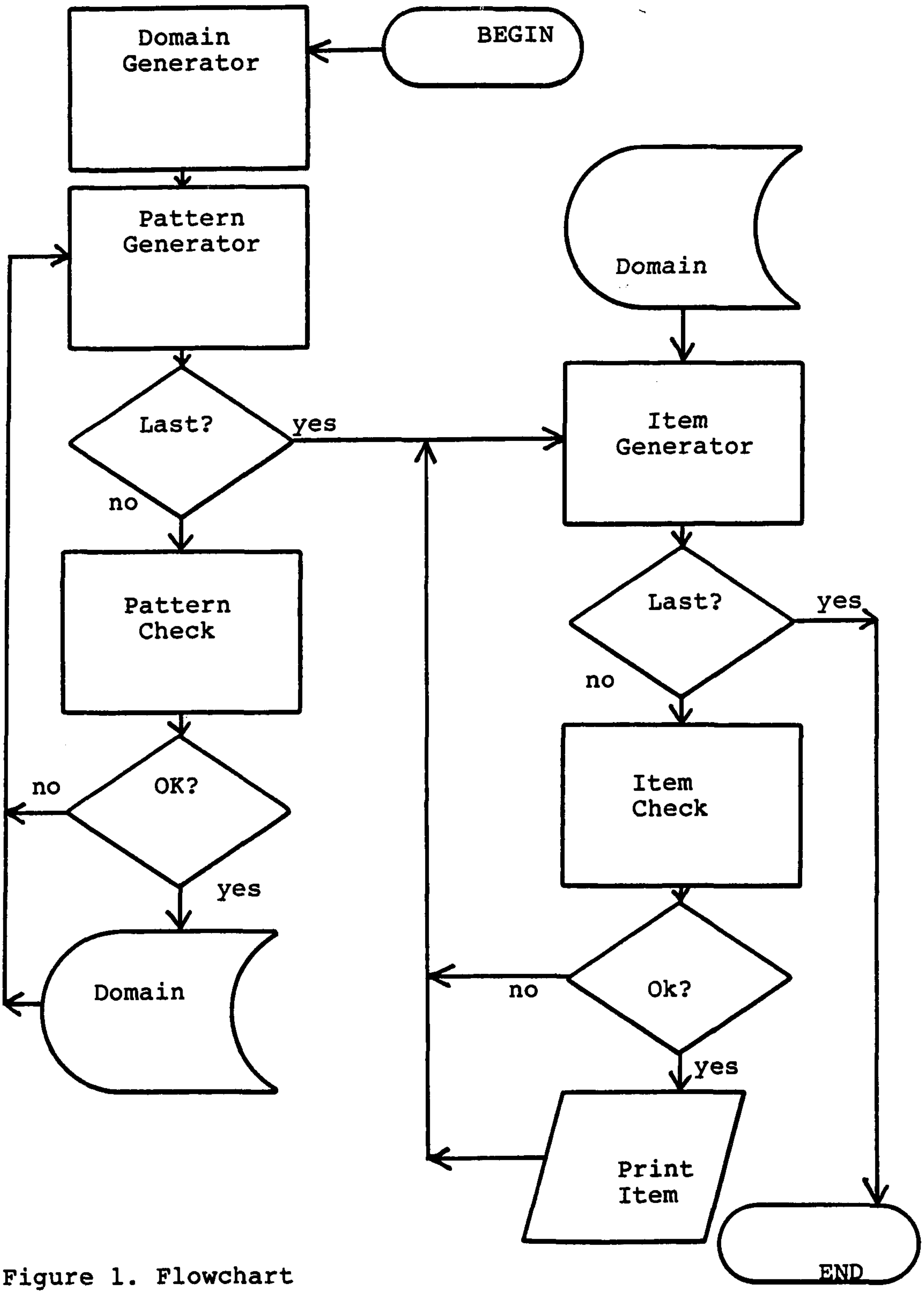

Figure 1. Flowchart

<u>Program output</u>

```
The total number of patterns was:          24
The number of patterns in domain was:      11

The following is a list of good items, or items with
only one solution [1-2-3].

(1)    2 - 1 - 4     (2)    2 - 1 - 4     (3)    2 - 1 - 4
       2 - 4 - 1            2 - 4 - 1            2 - 4 - 1
       4 - 1 - 2            4 - 3 - 1            4 - 3 - 2

(4)    2 - 1 - 4     (5)    2 - 1 - 4     (6)    2 - 1 - 4
       3 - 4 - 1            3 - 4 - 1            3 - 4 - 1
       4 - 1 - 2            4 - 3 - 1            4 - 3 - 2

(7)    2 - 1 - 4     (8)    2 - 1 - 4     (9)    2 - 1 - 4
       3 - 4 - 2            3 - 4 - 2            3 - 4 - 2
       4 - 1 - 2            4 - 3 - 1            4 - 3 - 2

(10)   2 - 3 - 4     (11)   2 - 3 - 4     (12)   2 - 3 - 4
       2 - 4 - 1            2 - 4 - 1            3 - 4 - 1
       4 - 1 - 2            4 - 3 - 2            4 - 1 - 2

(13)   2 - 3 - 4     (14)   2 - 3 - 4     (15)   2 - 3 - 4
       3 - 4 - 1            3 - 4 - 1            3 - 4 - 2
       4 - 3 - 1            4 - 3 - 2            4 - 1 - 2

(16)   2 - 3 - 4     (17)   2 - 3 - 4     (18)   2 - 4 - 1
       3 - 4 - 2            3 - 4 - 2            3 - 1 - 4
       4 - 3 - 1            4 - 3 - 2            4 - 1 - 2

(19)   2 - 4 - 1     (20)   2 - 4 - 1     (21)   3 - 1 - 4
       3 - 1 - 4            3 - 1 - 4            3 - 4 - 1
       4 - 3 - 1            4 - 3 - 2            4 - 1 - 2

(22)   3 - 1 - 4     (23)   3 - 1 - 4     (24)   3 - 1 - 4
       3 - 4 - 1            3 - 4 - 1            3 - 4 - 2
       4 - 3 - 1            4 - 3 - 2            4 - 3 - 1

(25)   3 - 1 - 4
       3 - 4 - 2
       4 - 3 - 2

The total number of items generated was:     165
The number of good items was:                 25

****** END OF PROGRAM CODER ******
```

As can be seen on the previous page, there were 24 rows or patterns generated. This number checks out since there are 24 permutations of four things taken three at a time. Of those 24 permutations eleven satisfied the requirements that there be no match in any column with the solution pattern [1-2-3]. This was checked by hand and found to be correct. From those eleven patterns 165 different items were generated and checked for alternate solutions. This number also checks out as there are 165 combinations of eleven things taken three at a time. Of those 165 items 140 were found to have more than one solution, leaving 25 admissible items or items with only one solution.

CHAPTER 5

DISCUSSION

Three types of items

The items generated by this program provide a means for constructing actual items for Crack the Code. As was mentioned earlier there are essentially three types of items. Those items with no duplication of colors in any of the columns might be considered easier items. The examinee can start with any column and there will be only one possible color for that column. That is, all columns can be determined through the process of elimination. An example of an item of this type is item number eight. Items where there are duplications of colors in one column may be more difficult. With these items the examinee must start with one of the columns which contains no duplication of colors. An example of an item of this type is item number three. For this item column three must be 3 and column two must be 2. Once these colors have been selected it leaves only one color for column one. If the examinee had started with column one he or she might have selected 3 for that column, which would have been more time consuming, since the person would have to go back and change that response once the remaining columns had been checked. The third type of item may be the most difficult. These items contain duplications in two of the three columns. Item number one is an example of an item of this type. At this point the order of difficulty is only speculation. Data on these 25 items is needed to to determine the true order of difficulty. Once determined, items could be presented to the examinee in ascending order of difficulty.

Varying the amount of feedback

The three components of planning advanced by Das should be kept in mind. As the items stand now they probably get a fairly good measure of an individual's ability to utilize strategies, the most effective being the elimination strategy. That is, by locating the column where three of the four colors have been tried the examinee should know which color to place in that column. As all the information needed to solve the problem is presented, the examinee need not generate and test different hypotheses. In the game Master Mind hypothesis testing is necessary as the codebreaker initially has no idea what the code is and must rely on feedback from the codemaker after each move. Although there is feedback in the form of zeros to the right of the initial three rows this does not vary from item to item. Feedback could be varied by inserting rows with one or more correct placements resulting in more than three rows. The following is an example of an item which contains one correct placement in one of its rows. It should be noted that this item is a variation on the last example item.

BLUE	-	YELLOW	-	RED	0
GREEN	-	BLUE	-	YELLOW	1
YELLOW	-	GREEN	-	BLUE	0
BLUE	-	RED	-	YELLOW	0
?	-	?	-	?	3

The item still has only one solution and the strategy would be essentially the same as it was before the extra row was inserted. Taking column three and eliminating only those colors which have a score of zero leaves GREEN as the only possible alternative for column three. The same procedure applied to column two leaves BLUE as the only possible response, which accounts for the one correct placement in row two. For column one there are two possible responses: GREEN or RED. Since GREEN must go in column three, column one must be RED. Adding rows with one or more correct placements may serve as a distractor making the items more difficult, especially for those who lack a firm grasp of the elimination principle. Due to the time estimated for a subject to solve such an item, Das decided not to use items like this in his initial tryout.

Varying the stimuli contained in the items

The ability to abstract the general rule or principle from a set of tasks is an important component of planning. This ability is probably tested to some extent by varying the feedback as suggested above. Another way of testing abstraction would be to vary the stimuli contained in the items. That is, colors could be converted to integers as was done in the program, or to geometric shapes. The examinee would have to realize that despite the change in stimuli the same principle or rule would apply.

Summation

The program provides a basis for constructing different items by conversion. In addition, rows varying in the number of correct placements could be added while retaining a single solution per item.

This would not have been possible without first establishing a set of items which are known to have one unique solution.

This program is unlike most test construction programs which build and perhaps administer actual test items. Many test construction programs build item banks which are useful in producing parallel, or alternate forms of a test. Some test construction programs are far more elaborate, as is the case with Computerized Adaptive Testing (CAT). Adaptive testing programs administer items to subjects based on their response to previous items. The result is a test which is tailored to each individual's ability level during the test administration process. Other programs provide scores and profiles as to a student's strengths and weaknesses. Some of these programs go as far as "directing the student to materials needed to learn aspects of a course on which he or she has not performed satisfactorily" (Libaw, 1974).

The sole purpose of this program was to solve what otherwise would have been an extremely tedious task in the test construction process.

Now that the task of generating items with one solution is done, it leaves test constructors to the problem of designing the test to obtain a good measure of planning in a relatively short period of time.

APPENDIX

THE PROGRAM

```
Program Coder(input,output);
const
     rowmax = 3;   { number of rows }
     colmax = 3;   { number of columns }
     colors = 4;   { number of colors or shapes }
     limit = 24;   { number of rows in domain }
type
     pattern = array[1..colmax] of integer;
     item = array[1..rowmax,1..colmax] of integer;
     domain = array[1..limit,1..colmax] of integer;
var
     pat: pattern;
     Im: item;
     dom: domain;
     k,           { index for rows in domain[] }
     item_count,{ total number of items }
     good_item, { number of items with one solution }
     sol_count, { number of solutions per/item }
     pat_count, { total number of patterns }
     domain_count: integer;{ number of patterns in
                              domain[] }
     last, L,     { indicates last item and pattern }
     ok: boolean  { indicates a good pattern }

{ initial_pat(pat):Generates initial
  pattern[1 - colmax].
}
  procedure initial_pat(var pat: pattern);
  var
       j: integer; { index for columns in pat[] }
  begin
       for j := 1 to colmax do
            pat[j] := j
  end;      { end of procedure initial_pat }
```

```
{ match(m,j,pat): Checks the pattern for a matching
  character and returns false if no match is found.
}
  procedure match(var m: boolean; j: integer; pat:
                    pattern);
  var
       t: integer; { target character to be matched }
  begin
       m := false;
       t := pat[j];
       while (j > 1) do
       begin
            j := j - 1;
            if (pat[j] = t) then
               m := true
       end
  end;    { end of procedure match }

{ pat_generator(pat,L): Generates all possible patterns
  given the constraints of colmax and colors.
}
  procedure pat_generator(var pat: pattern; var L:
                                boolean);
  var
       j: integer;   { index for columns in pat[] }
       m: boolean;   { value returned by match() }
  begin
       L := false;
       j := colmax;
       repeat
            if (pat[j] <> colors) then
            begin
                 pat[j] := pat[j] + 1;
                 match(m,j,pat);
                 if (j < colmax) and (m = false) then
                    j := j + 1
            end
            else    { if pat[j] = colors then }
            begin
                 pat[j] := 0;
                 j := j - 1
            end
       until (m = false) and (j = colmax) and
             (pat[j] <> 0) or (j = 0);
       if (j < 1) then
          L := true
  end;      { end of procedure pat_generator }
```

```
{ pat_check(pat,ok): Checks each pattern generated by
  pat_generator() for inclusion in domain[]. There can
  be no match with the target pattern[1-2-3].
}
  procedure pat_check(var pat: pattern; var ok:
                      boolean);
  var
       j: integer;  { index for columns in pat[] }
  begin
       j := colmax;
       ok := true;
       while (j >= 1) and (ok) do
       begin
            if (pat[j] = j) then
               ok := false;
            j := j - 1
       end
  end;   { end of procedure pat_check }

{ domain_generator(pat,pat_count,domain_count):
  Generates domain[].
}
  procedure domain_generator
   (var pat: pattern; var pat_count,domain_count:
    integer);
  var
       i,           { index for rows in domain[] }
       j: integer;  { index for columns in pat[] and
                      domain[] }
  begin
     { generate initial pattern[1 - colmax] }
       initial_pat(pat);
     { build domain[] }
       i := 1; l := false;
       while (L = false) do
       begin
            pat_generator(pat,L);
            if (L = false) then
               pat_count := pat_count + 1;
            pat_check(pat,ok);
            if (ok) and (L = false) then
            begin
                 for J := 1 to colmax do
                     dom[i,j] := pat[j];
                 i := i + 1;
                 domain_count := domain_count + 1
            end
       end
  end;    { end of procedure domain_generator }
```

```
{ load_item(pat,): Loads initial item from domain[].
}
  procedure load_item(var pat: pattern);
  var
       i,  { index for rows in item[] and domain[] }
       j: integer; { index for columns in item[] and
                     domain[] }
  begin
       for i := 1 to rowmax do
           for j := 1 to colmax do
               Im[i,j] := dom[i,j];
  end;    { end of procedure load_item }

{ load(i,k): Loads row k from domain[] into row i of
  item[].
}
  procedure load(i,k: integer);
  var
       j,          { index for columns in item[] }
       l: integer; { index for columns in domain[] }
  begin
       l := 1;
       for j := 1 to colmax do
       begin
            Im[i,j] := dom[k,l];
            l := l + 1
       end
  end;     { end of procedure load }
```

```
{ find_match(k,i): Finds a matching pattern in domain[]
  with row i of item[] and returns its position k.
}
  procedure find_match(var k: integer; i: integer);
  var
       m_count, { number of matching characters in a
                  row }
       j: integer; { index for columns in item[] and
                     dom[] }
       m: boolean; { true if all chars. in a row match
                   }
  begin
       m := false;
       k := 0;
       while (m = false) do
       begin
            m_count := 0;
            k := k + 1;
            for j := colmax downto 1 do
                if (Im[i,j] = dom[k,j]) then
                   m_count := m_count + 1;
            if (m_count = colmax) then
                m := true
       end
  end;    { end of procedure find_match }
```

```
{ item_generator(k,last): Generates all unique items
  from domain[], one at a time.
}
  procedure item_generator(var k: integer; var last:
                                  boolean);
  var
       i: integer; { index for rows in item[] }
  begin
       i := rowmax;
       last := false;
       repeat
            if (k < domain_count) then
            begin
                 k := k + 1;
                 load(i,k);
                 if (i < rowmax) then
                    i := i + 1
            end
            else    { if k = domain_count then }
            begin
                 Im[i,1] := 0;
                 i := i - 1;
                 if (i > 0) then
                    find_match(k,i)
            end
       until (i = rowmax) and (Im[i,1] <> 0) or
             (i = 0);
       if (i < 1) then
          last := true
  end;   { end of procedure item_generator }
```

```
{ item_check(col_count,pat,Im): Checks each item
  for alternative solutions.
}
procedure item_check(var sol_count:integer;
                     pat:pattern; Im:item);
var
     i,  { index for rows in item[] and pat[] }
     j: integer; { index for columns in item[] and
                  pat[] }
     ok: boolean;{ true if pat is a solution to item[]
                 }
begin
     i := rowmax;
     L := false;
     initial_pat(pat);
     while (L = false) do
     begin
            ok := true;
            while (i >= 1) and (ok) do
            begin
                 for j := colmax downto 1 do
                     if (Im[i,j] = pat[j]) then
                        ok := false;
                 i := i - 1
            end;
            if (ok) then
               sol_count := sol_count + 1;
            pat_generator(pat,L);
            i := rowmax
       end
  end;    { end of procedure item_check }

{ print_item(Im): Prints an item one at a time.
}
  procedure print_item(Im: item);
  var
       i,           { index for rows in item[] }
       j: integer; { index for columns in item[] }
  begin
       for i := 1 to rowmax do
       begin
            for j := 1 to colmax do
                if (j < colmax) then
                   write(Im[i,j]:2,' -')
                else
                   write(Im[i,j]:2);
            writeln
       end;
       writeln;  writeln
  end;  { end of procedure print_item }
```

```
{ PROGRAM CODER.
}
  begin
       item_count := 0;
       good_item := 0;
       pat_count := 1;
       sol_count := 0; domain_count := 0;
     { build item domain }
       domain_generator(pat,pat_count,domain_count);
       writeln('The total # of patterns is',pat_count);
       writeln;
       writeln('The number of pats in domain
               is',domain_count);
       writeln;  writeln;
       writeln('The following is a list of good
               items,');
       writeln('or items with only one solution');
       writeln;
     { build and check items for alternate solutions }
       load_item(pat);
       item_check(sol_count,pat,Im);
       if (sol_count = 1) then
       begin
            print_item(Im);
            good_item := good_item + 1
       end;
       last := false; k := rowmax;
       while (last = false) do
       begin
            item_count := item_count + 1;
            item_generator(k,last);
                 if (last = false) then
            begin
                 item_check(sol_count,pat,Im);
                     if (sol_count = 1) then
                 begin
                      print_item(Im);
                      good_item := good_item + 1
                 end
            end;
            sol_count := 0
       end;
       writeln('The total number of items
               is',item_count);
       writeln;
       writeln('The number of good items
               is',good_item);
       writeln;
       writeln('****END OF PROGRAM CODER ****')
  end.
```

REFERENCES

Bogen, J. E. (1969). The other side of the brain: Parts I, II, and III. Bulletin of the Los Angeles Neurological Society, 34, 73-105; 135-162; 191-203.

Das, J. P., & Heemsbergen, D. B. (1983). Planning as a factor in the assessment of cognitive processes. Journal of Psychoeducational Assessment, 1, 1-15.

Doehring, D. G., & Reitan, R. M. (1962). Concept attainment of human adults with lateralized cerebral lesions. Perceptual and Motor Skills, 14, 27-33.

Halstead, W. C. (1951). Biological intelligence. Journal of Personality, 20, 118-130.

Jensen, A. R. (1980). Bias in mental testing. New York: The Free Press.

Kaufman, A. S., & Kaufman, N. L. (1983). Kaufman Assessment Battery for Children administration and scoring manual. Circle Pines, MN: American Guidance Service.

Levy, J. (1972). Lateral specialization of the human brain: Behavioral manifestations and possible evolutionary basis. In J. A. Kiger (Ed.) Biology of behavior. Corvallis, Ore.: Oregon State University Press.

Libaw, F. B. (1974)."Pedagogical Implications,"p.197 in G. Lippey (ed.) Computer Assisted Test Construction. Englewood Cliffs, NJ: Educational Technology Publications.

Luria, A. R. (1966). Human brain and psychological processes. New York: Harper & Row.

Naglieri, J. A., Kamphaus, R. W., & Kaufman, A. S. (1983). The Luria-Das simultanious-successive model applied to the WISC-R. Journal of Psychoeducational Assessment, 1, 25-33.

Nebes, R. D. (1974). Hemispheric specialization in commissurotomized man. Psychological Bulletin, 81, 1-14.