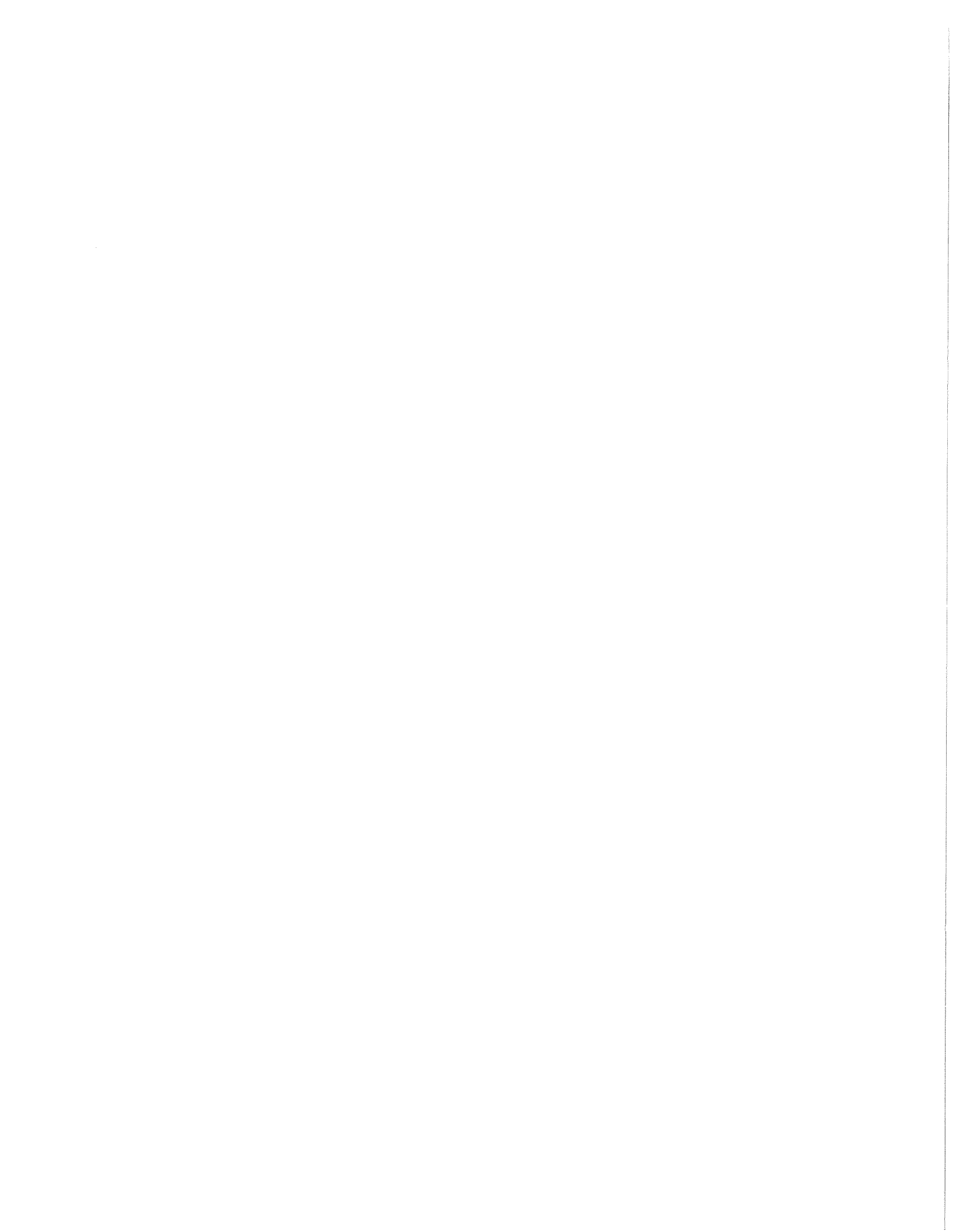


A SELF-DIRECTING TEACHING PROGRAM THAT
GENERATES SIMPLE ARITHMETIC PROBLEMS

Jonathan D. Wexler

Computer Sciences Technical Report #19

May 1968



Abstract

This paper describes a digital computer program (COACH) written in extended Algol that is currently operating on the Burroughs B5500. The program automatically (1) generates its own problems in simple arithmetic, (2) evaluates responses to questions by students, (3) provides hints when possible, and (4) uses probability switches and the student's past history in order to decide the type of question to ask him next.

The specific topic metrics, problem generation methods, and teaching strategies used in COACH are explained in detail. The program operates in a dynamic environment in which teletypes are used for communication between the student and the computer.

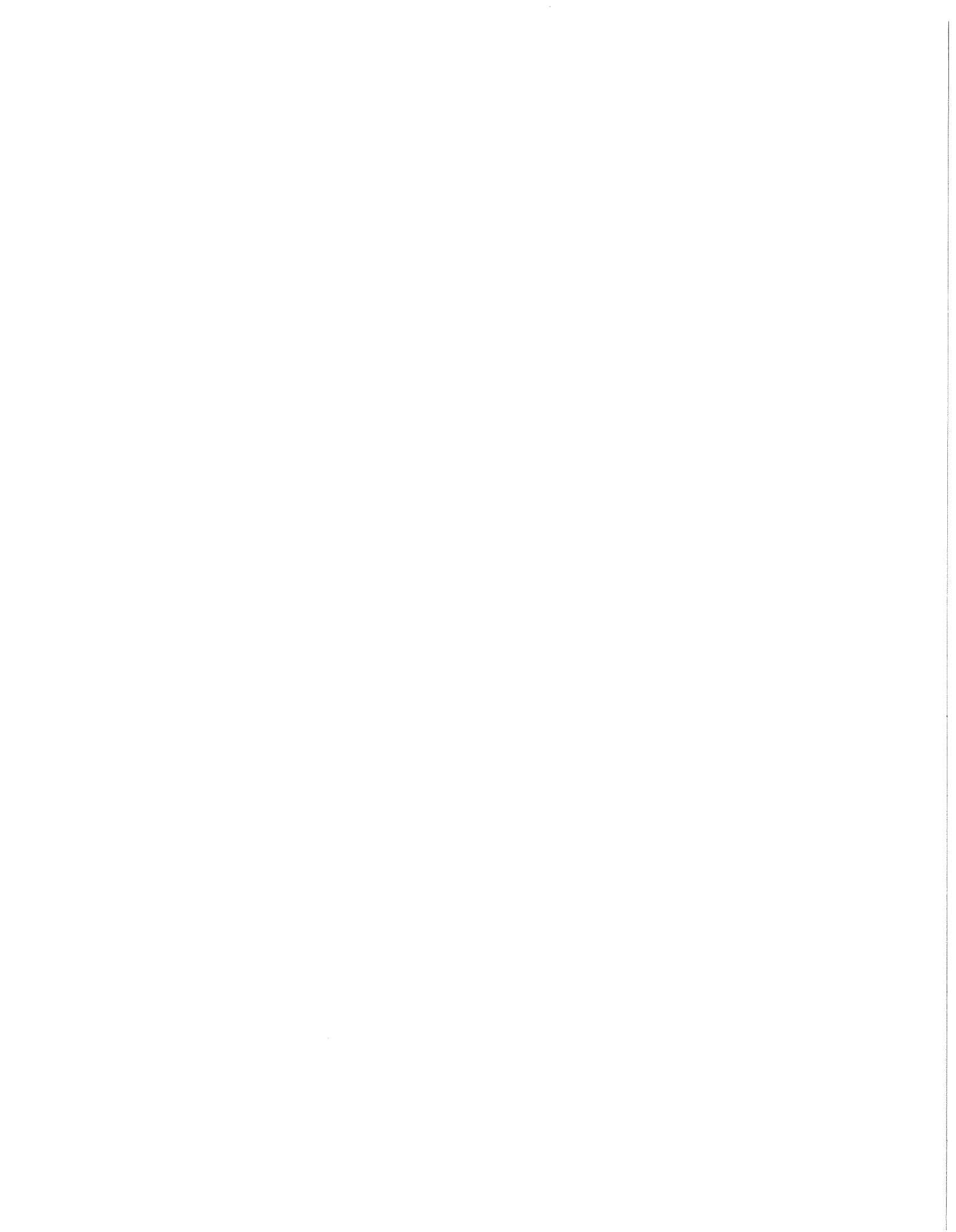


Table of Contents

	page
1. Introduction and Background	1
2. Program Operation	2
3. Levels of Difficulty and Ranges	6
4. Teaching Strategies	8
4.1 Acceleration Factor	10
4.2 Special Question Lists	12
5. Probability Switches and the Flow of Control	13
6. Metrics and Generators	19
6.1 Topic Metrics	19
6.1.1 Digit Scores	20
6.1.2 Position Scores	20
6.1.3 Operand Difficulty Scores	20
6.1.4 Problem Difficulty Scores	21
6.2 Question Generators	22
6.2.1 The Question Generator for Addition Problems ...	23
6.2.2 The MAKENUM Routine	27
7. Answer and Hint Modules	30
8. Miscellaneous Features	34
8.1 The Practice Option	34
8.2 The Monitor Printer	34
8.3 Control Mode Options	35

Table of Contents (Continued)

	page
8.3.1	Setting Probability Switches 35
8.3.2	Setting INCREMENTSPERLEVEL 36
8.3.3	Recovery Procedure 36
8.3.4	Debugging Aids 37
9.	Current Results 38
10.	Conclusions 41
11.	Acknowledgements 43
12.	Bibliography 44
Appendix A	Special Conventions in COACH 45
A.1	String/List Storage 45
A.2	Internal Representation of Questions 46
A.3	Format Specifications for Teletype Questions 49
Appendix B	Student/Teacher Operation of COACH 52
Appendix C	Sample Teletype Output 55
Appendix D	Sample Monitor Printer Output 60

1. Introduction and Background

The use of digital computers in instructional environments provides an opportunity to study explicit teaching strategies and to help students learn course materials on an individual basis.

Early work in programmed instruction on teaching machines concentrated on the construction of specific sequences of question and subject matter frames. Much of the recent work in CAI (Computer Assisted Instruction) continues along the same lines, although the frames for questions and information are now stored in computers instead of teaching machines. Special purpose languages (e.g. PLANIT, reference 1) have been developed to facilitate the construction of frames and to allow a more sophisticated analysis of student responses.

There is a one to one correspondence between the questions the computer can ask a student and the questions explicitly constructed for it a priori. Frame sequences containing these questions must be written with certain "typically average" students in mind. This in turn limits the sensitivity of the computer to individual differences, which is undesirable. If a student needs more practice in some special area he will be limited by the finite size of the question pool for that area.

It is natural to consider the idea of having a digital computer in a student/computer environment automatically generate additional appropriate questions when needed, however, there has been relatively little work done on this subject. In an earlier monograph (reference 3), Smallwood used a statistical model to deal with the problem of selecting

the next question or information frame from among several specific possibilities, however this is considerably different from the generative problem.

A subsequent paper by Uhr (reference 4) dealt with some of the problems associated with automatic question generation in very simplified situations. This paper is the most immediate precursor to the present report and has influenced the design of COACH.

In order to gain familiarity with the organization and techniques that might be useful in an automatic instructional environment, a well structured problem domain was selected for study, namely arithmetic.

The program (COACH - Constructed Arithmetic and Controlled Help) described in the following pages constructs its own questions for arithmetic using pre-selected topic metrics and a generative technique which is explained in detail. An attempt has been made to have the internal organization as modular as possible in order to clearly delimit specific areas. By giving the program (i.e. the system) the capability of generating its own problems the limitation of a finite question pool is removed. It is also possible to examine and alter specific teaching strategies as well as handling deep remedial sequences automatically.

2. Program Operation

The student is initially asked to choose the course or courses for which problems should be generated. He specifies a level of difficulty for each course, which is a digit from 1 to 8. Examples are given in

Figure 1 of the types of problems generated by the program for various levels of difficulty.

	Addition	Subtraction	Multiplication	Division
Level 2	$\begin{array}{r} 86 \\ + \underline{69} \end{array}$	$\begin{array}{r} 822 \\ - \underline{651} \end{array}$	$\begin{array}{r} 20 \\ \times \underline{8} \end{array}$	$63/7$
	DS=32 (30-33)	DS=50 (50-53)	DS=21 (20-22)	DS=60 (60-64)
Level 4	$\begin{array}{r} 648 \\ 887 \\ 667 \\ + \underline{827} \end{array}$	$\begin{array}{r} 7124 \\ - \underline{6960} \end{array}$	$\begin{array}{r} 974 \\ \times \underline{37} \end{array}$	$455/35$
	DS=123 (120-123)	DS=102 (100-108)	DS=87(80-84)	DS=156 (150-155)
Level 6	$\begin{array}{r} 5222 \\ 2280 \\ 5820 \\ 2222 \\ 2600 \\ + \underline{694} \end{array}$	$\begin{array}{r} 21013670 \\ - \underline{3517521} \end{array}$	$\begin{array}{r} 7977 \\ \times \underline{679} \end{array}$	$5778/835$
	DS=207 (200-205)	DS=240 (240-245)	DS= 155 (150-155)	DS=270 (260-268)
Level 8	$\begin{array}{r} 72601 \\ 62915 \\ 280255 \\ 600682 \\ + \underline{251212} \end{array}$	$\begin{array}{r} 6662229525 \\ - \underline{2008176872} \end{array}$	$\begin{array}{r} 329991 \\ \times \underline{41206} \end{array}$	$602826/42002$
	DS=356 (350-355)	DS=352 (350-365)	DS=267 (250-254)	DS=414 (400-430)

Sample Problems with Associated Difficulty Scores and Ranges

Figure 1

The difficulty score of the level 2 addition problem in Figure 1 is 32.

The numbers in parenthesis indicate the range of difficulty which the

generated problem should fall in.

After making his selection (s), the student is presented with a series of problems. If he chooses all four types at level 1 he would receive a sequence of problems like those in Figure 2.

$\begin{array}{r} 8 \\ -5 \end{array}$	$\begin{array}{r} 4 \\ +6 \end{array}$	$\begin{array}{r} 6 \\ -1 \end{array}$	$\begin{array}{r} 6 \\ +6 \end{array}$	$\begin{array}{r} 51 \\ -6 \end{array}$	$9/3$	$\begin{array}{r} 1 \\ \times 1 \end{array}$	$\begin{array}{r} 4 \\ +6 \end{array}$
$10/5$	$52/2$	$\begin{array}{r} 52 \\ -6 \end{array}$	$16/2$	$\begin{array}{r} 50 \\ -4 \end{array}$	50	$\begin{array}{r} 26 \\ -9 \end{array}$	$\begin{array}{r} 20 \\ +4 \end{array}$
$57/3$	$9/3$	$\begin{array}{r} 5 \\ \times 2 \end{array}$	$\begin{array}{r} 67 \\ -42 \end{array}$	$\begin{array}{r} 84 \\ -66 \end{array}$	$57/3$	$56/7$	$\begin{array}{r} 522 \\ -77 \end{array}$
$\begin{array}{r} 50 \\ +55 \end{array}$	$\begin{array}{r} 218 \\ -74 \end{array}$	$\begin{array}{r} 11 \\ +21 \end{array}$	$56/7$	$\begin{array}{r} 568 \\ -168 \end{array}$	860	$\begin{array}{r} 14 \\ +24 \end{array}$	$\begin{array}{r} 5 \\ \times 4 \end{array}$
$\begin{array}{r} 80 \\ +45 \end{array}$	$\begin{array}{r} 624 \\ -616 \end{array}$	$\begin{array}{r} 770 \\ -73 \end{array}$	$\begin{array}{r} 30 \\ +69 \end{array}$	$\begin{array}{r} 690 \\ -178 \end{array}$	46	$\begin{array}{r} 4 \\ +6 \end{array}$	$\begin{array}{r} 44 \\ +78 \end{array}$
$27/9$	$28/7$	$\begin{array}{r} 861 \\ -720 \end{array}$	$\begin{array}{r} 5 \\ \times 1 \end{array}$	$\begin{array}{r} 8 \\ \times 2 \end{array}$	4	$27/3$	$\begin{array}{r} 576 \\ -534 \end{array}$
$\begin{array}{r} 756 \\ -573 \end{array}$	$28/7$	$72/12$	$\begin{array}{r} 678 \\ -322 \end{array}$	$\begin{array}{r} 577 \\ -4 \end{array}$	$34/17$	$66/22$	$120/60$
$\begin{array}{r} 968 \\ -342 \end{array}$	$\begin{array}{r} 60 \\ \times 37 \end{array}$	$28/4$	$\begin{array}{r} 78 \\ +26 \end{array}$	$\begin{array}{r} 79 \\ \times 11 \end{array}$	33	$21/7$	$\begin{array}{r} 96 \\ \times 60 \end{array} \dots$

Intermixed Problems, Each Type Beginning at Level 1

Figure 2

As the student answers each question he is told whether he is correct or incorrect. If the student gives an incorrect answer he will be given a few hints, as shown in Figure 3.

ADD THESE NUMBERS

$$\begin{array}{r} 143 \\ 64 \\ \hline 295 \\ 503 \leftarrow \end{array}$$

PERHAPS SOME HINTS WOULD HELP
 YOUR ANSWER OF 503 IS INCORRECT
 YOU ARE JUST A LITTLE HIGH
 ONE OF YOUR DIGITS IS WRONG

A Wrong Answer and Some Sample Hints

Figure 3

The complete problem is then presented again, and if he gives another wrong answer he will receive the correct answer as well as any applicable hints. The program will then keep presenting the problem until he types in the correct answer.

When a student gets tired, or for other reasons, he may type in the message QUIT instead of an answer. The system will ask him if he wishes to continue at a later time and will save restart information on disk if so desired. At a later time he may begin at that restart point. If the problems are too difficult or if the student wishes to change his selection of topics or difficulty levels, he may type the message RESTART and then specify another curriculum.

A record of the conversation between the student and the computer is generated for the teacher on the line printer during a student's run. This record contains additional data such as the difficulty scores of the

generated problems and elapsed processor and input/output time. It should be noted that the timing information should not be used to estimate a student's latency of response time on particular problems because hardware considerations and the monitor system on the Burroughs B5500 invalidate such estimates.

3. Levels of Difficulty and Ranges

The selection of eight major levels of difficulty within each course instead of ten, twelve, . . . was completely arbitrary. It was originally intended to be suggestive of the eight levels or grades in grammar school. Subsequently it became useful to have a new level signal a change in the parameter specifications for particular problem types as well as specifying higher difficulty scores. This may be illustrated by observing the effect of moving from level three to level four problems in multiplication: Problems of the form $\begin{array}{r} XXX \\ \underline{XX} \end{array}$ can be generated in level 4, but not in level 3. (Level 3 multiplication causes problems only of the form $\begin{array}{r} XX \\ \underline{XX} \end{array}$ or $\begin{array}{r} X \\ \underline{X} \end{array}$ to be generated.)

There is a threshold difficulty score associated with each level for each type of problem. Originally the scores were the same for all four types of problems, but because of changes made to metrics for various courses this arrangement was found to be undesirable. The scores currently used for levels of difficulty in subtraction are given in Table 1.

		Score
Level	1	0
	2	50
	3	75
	4	100
	5	175
	6	225
	7	275
	8	350
	Max	500

Threshold Difficulty Scores for Subtraction

Table 1

A given level of difficulty is subdivided into a number of ranges. The number of ranges within a level is specified by the variable INCREMENTSPERLEVEL. This variable is currently set to a value of ten, so within level 4 of subtraction the ranges are: 100-108, 108-116, 116-124, 124-132, 132-140, 140-148, 148-156, 156-164, 164-172, 172-175.

Higher score problems generated within a level of difficulty will gradually become more difficult. The first few subtraction problems generated in level 4 will fall within range 1 (100-108). As problems are answered correctly the range numbers will be changed to higher values until the last range in the level has been reached or passed. The reader should examine Figure 4 which gives complete sequence of level 4 subtraction problems. The sequence assumes the student answers each problem correctly.

Problem	Difficulty Score	Range Scores	Range Number
7124 - <u>6960</u>	102	100 - 108	1
7647 - <u>6151</u>	101	100 - 108	1
21282 - <u>16224</u>	117	116 - 124	3
15314 - <u>6961</u>	118	116 - 124	3
56722 - <u>16189</u>	135	132 - 140	5
51747 - <u>3699</u>	132	132 - 140	5
122266 - <u>23735</u>	157	156 - 164	8
212659 - <u>29667</u>	157	156 - 164	8

A Sequence of Level 4 Subtraction Problems

Figure 4

4. Teaching Strategies

The basic teaching criteria adopted are the following:

- 1) A student should correctly answer two of three consecutive questions from the same problem difficulty score range (henceforth

"range" will be used instead of "problem difficulty score range") in the course before the range is raised.

2) A student who misses two consecutive questions from the same range should be asked easier questions, i.e. his range should be lowered. A question is considered "missed" if it is necessary for the program to tell the student the correct answer.

3) A student who has not answered the current question correctly, but who has answered the preceding question correctly, should be asked another question in the same range, and similarly for the case when the current question was answered correctly but the preceding question was missed.

If the first answer supplied by the student is wrong the program will give him at most two hints on why his answer is wrong and then ask the question again. If his second answer is also wrong the student will receive the correct answer, and also all the hints applicable to his second incorrect answer. He will then be asked the question again and again until he types in the correct answer.

The sequence of steps in presenting a single question is the following:

1. Present question to student.
- 1A. If student is wrong, ask question again.
2. If answer is still wrong, give student up to two hints.
3. If answer is still wrong, give correct answer, hints, and cycle thru this step 3 until the correct answer is typed in.

Step 1A originally compensated for typing errors. It has been eliminated on the basis of actual teletype experience. (By the time step 3 was reached students were becoming bored with seeing the same problem.)

COACH has adopted the philosophy that the entire question should be repeated for the student rather than simply asking him to give another answer. This has the advantage of minimizing the chance for loss of data and other hardware difficulties in transmitting the problem from the computer to the teletype (particularly since echo-checking features are not available). The drawback is that additional time is spent typing out the same problem again, but this seems to be acceptable to students. It can easily be changed if it turns out to be undesirable on the basis of additional data.

One special situation will also cause the student's range to be lowered: if he misses the current question, and if the preceding question in this range was not answered correctly the first time it was presented, and if the second question back in this range was missed, then the program assumes the student is fumbling and will decrease his range.

4.1 Acceleration Factor

The subtraction problems in Figure 4 also illustrate the use of an acceleration factor. This factor allows a student to make faster than normal progress if his performance seems to warrant it. Normally, a student would be asked two questions in the first range of a level, two questions

in the second range, two questions in the third, etc. If he answers all these questions correctly, a slow rate of increase in difficulty may bore him. An acceleration factor is used in order to speed up this process. The factor is added to the last range number in order to produce the next range value which may yield a higher than normal problem range for his next question. For example, the acceleration factor might start at one at the beginning of a level (e.g. level 3). If two consecutive ranges have their questions answered correctly the acceleration factor is increased by one. If a student answers all the questions in a level correctly the following progression of ranges can occur:

Range in level 3	1	2	3	5	7	10
Acceleration Factor	1	1	2	2	3	3

In order to keep up the "momentum" when passing from one level to another, the acceleration factor is simply decreased by one instead of being reset to one. Thus the next level would have the following progression of ranges if the student continued to answer all the questions correctly:

Range in level 4	1	3	5	8	10
Acceleration Factor	2	2	3	3	

When the student misses several questions in a range the acceleration factor is used in deciding which lower number range to use next. At that time the factor may be decreased by one and succeeding range drops may cause

it to be further reduced to its minimum value of one. If the program needs to give the student the correct answer at any time during a particular range then the acceleration factor will not be changed when the range is raised, e.g. if a question was missed in level 3, range 7 (above) then the next range would be 9, instead of 10.

4.2 Special Question Lists

A question generated within BABYG may be viewed as an independent manipulable entity. Associated with each question is auxiliary information on the type of problem (addition, subtraction, ...), the difficulty score of the problem, the format to use in presenting the question on a teletype along with its operands and correct answer, a list of hints possibly applicable to incorrect student responses, and the last two responses made by the student to the question. A detailed schematic of a question is given in Appendix A.2.

There are two question lists specifically built into COACH, OPMISLIST and REDOLIST. When a question is missed it is added to the front of OPMISLIST. At a later time it may be asked again, as dictated by probability switch 1 (probability switches are discussed in the next section). REDOLIST holds questions that have been answered correctly but may be asked once more. Before a student is allowed to advance to the next level in a subject, OPMISLIST is scanned to see if it contains any questions on the same subject with a difficulty score lower than that of the next level. If any such questions are found they are removed from OPMISLIST and asked. Thus a student will encounter questions he missed

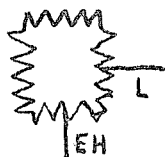
earlier, hopefully with better results the second time. More difficult questions will be retained in OPMISLIST even if the student's range drops. Ultimately he will encounter them again.

5. Probability Switches and the Flow of Control

By means of probability switches (PSW) a teacher or supervisor is able to exert a limited amount of control on some performance features of COACH.

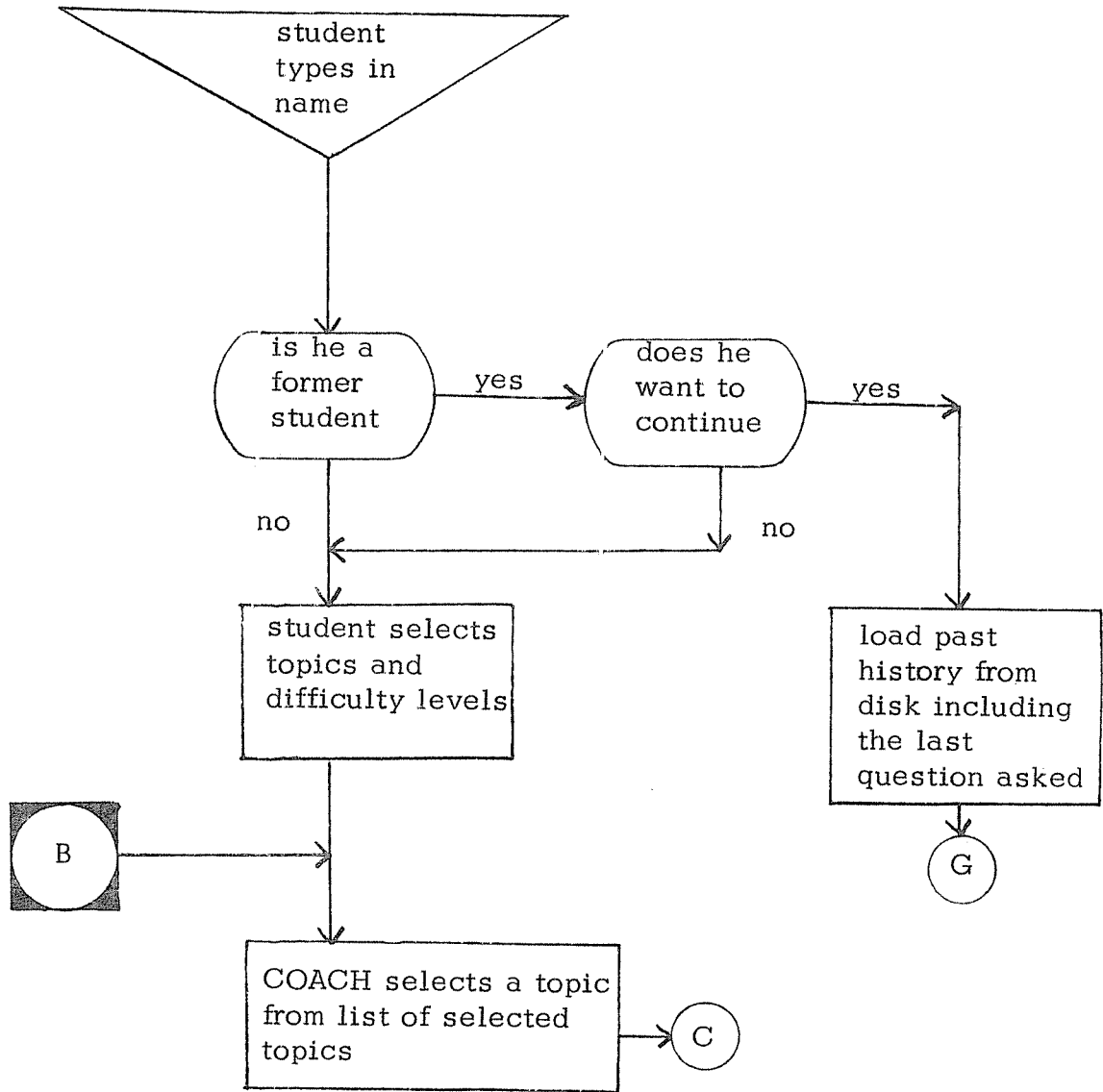
Probability switches make use of a pseudo random number generator that is part of COACH. Each time the flow of control reaches a probability switch a random integer between 0 and 99 is generated by the program. This integer is compared with the current setting of the probability switch and then one of two branches is taken by the program.

The L (lower) branch is taken if the randomly generated integer is less than the probability switch value; otherwise the EH (equal or higher) branch is used.

The symbol  is used in Figure 5 to indicate the points

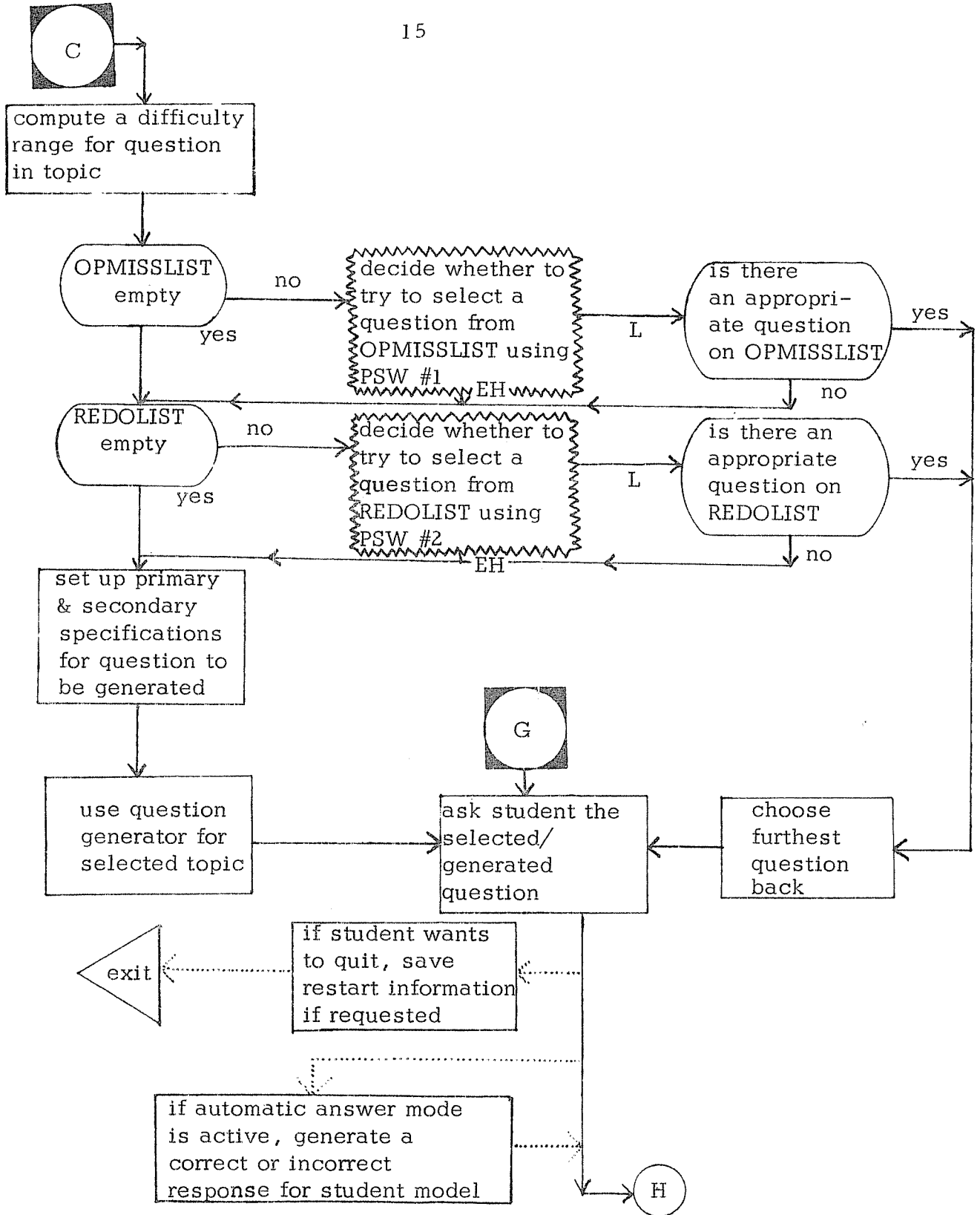
in the program where probability switches are used to make decisions.

By setting a probability switch to 0 or 100 it is possible to force the EH or L branch to always be taken.



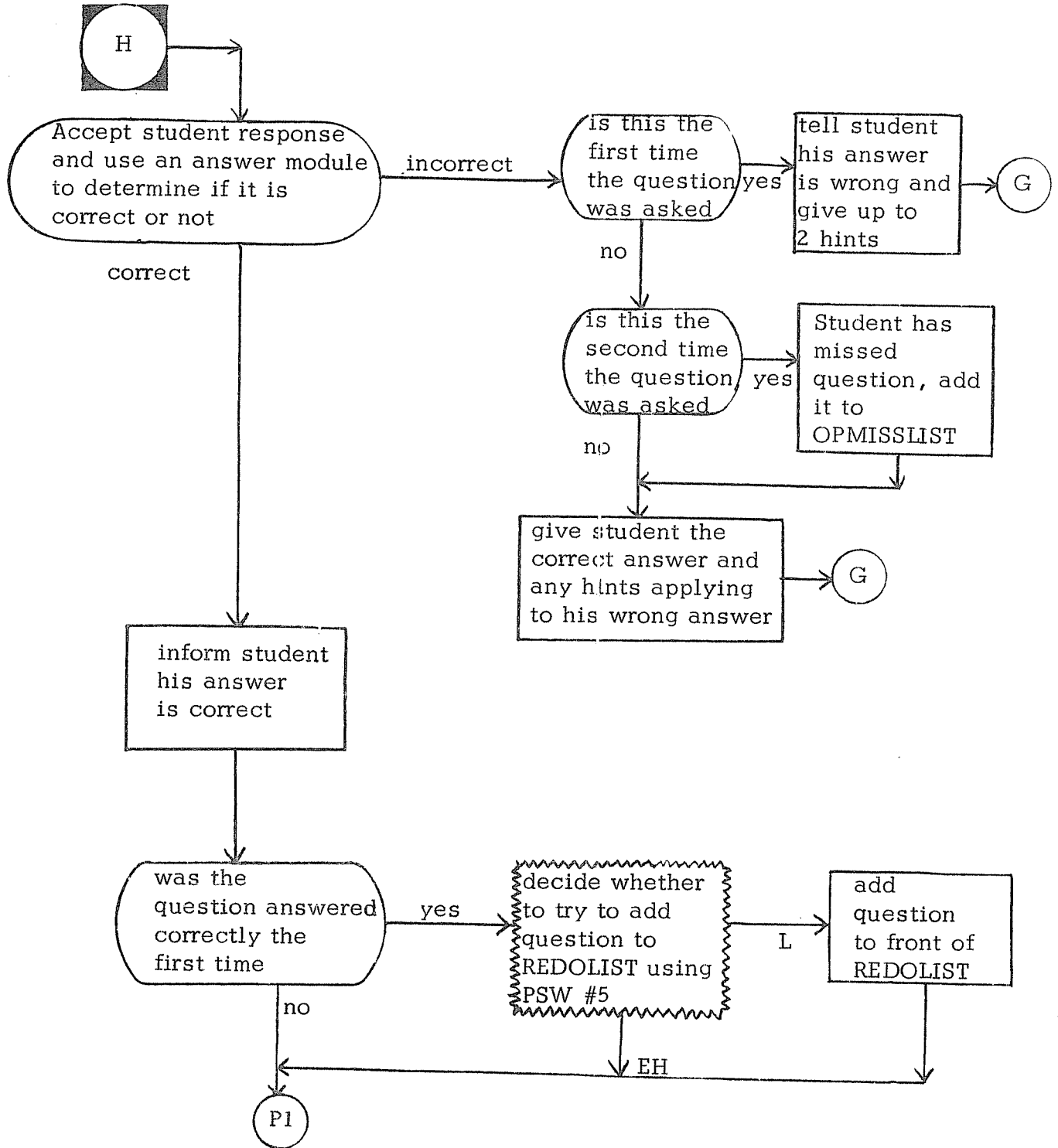
Flowchart of COACH

Figure 5



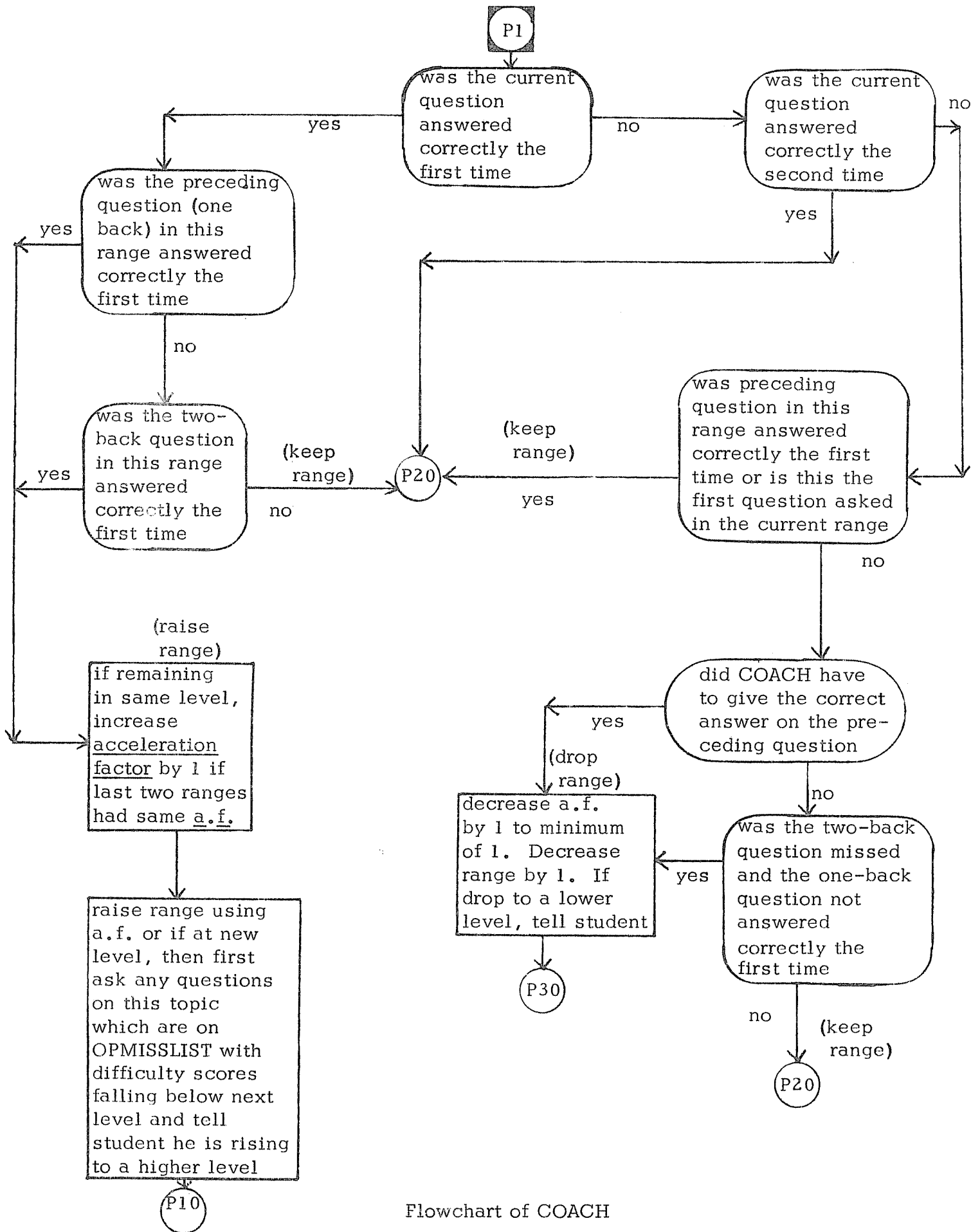
Flowchart of COACH

Figure 5 (cont.)



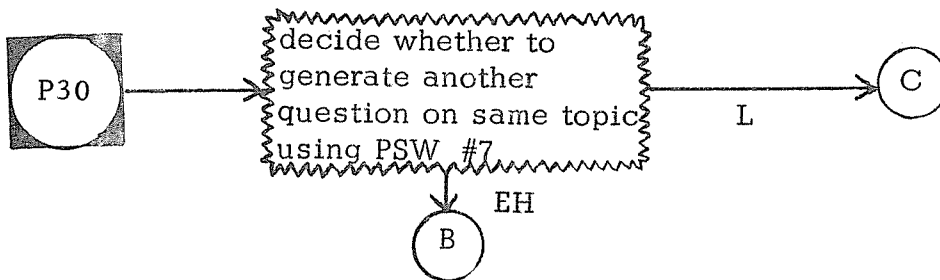
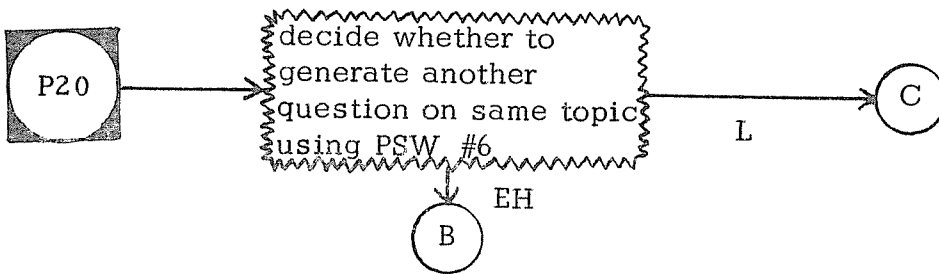
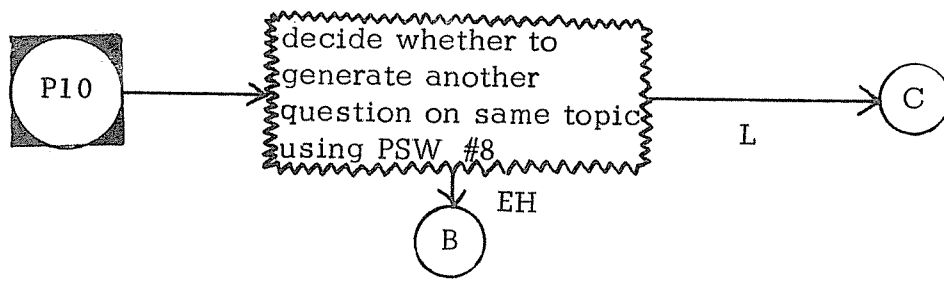
Flowchart of COACH

Figure 5 (Cont)



Flowchart of COACH

Figure 5 (Cont)



Flowchart of COACH

Figure 5 (Cont.)

6. Metrics and Generators

The following hypotheses are explicitly embodied in COACH:

1) By using a suitably chosen metric for each topic it is possible to assign numerical scores which represent relative difficulty for problems within each topic.

2) Problem generators can be constructed for topics with well-defined metrics.

6.1 Topic Metrics

The following considerations entered into the design of the heuristic metrics adopted for arithmetic:

1) The difficulty score for a problem should be a linear combination of the difficulty scores of its individual operands.

2) The linear combinations in 1) may be different for metrics in different topics.

3) The operand scores should be such that an n digit operand generally has a higher score than an m digit operand if $n > m$.

4) The fact that certain decimal digits are easier to work with than others should be taken into account. (It is easier for a student to manipulate the digit 2 in a problem than the digit 9.)

With these constraints in mind the actual metrics used in COACH were specified as follows:

6.1.1 Digit Scores

The digit scores assigned to decimal digits are the following:

	decimal digits		
digit score = 1		0	
digit score = 2	1	2	5
digit score = 3	4	6	8
digit score = 4	3	7	9

6.1.2 Position Scores

The digits in a number are identified by a particular position, for example the number 1492 has the digit 2 in position one, the digit 9 in position two, and so forth. Position scores are the following:

position	position score
1	2
2	3
3	5
4	7
≥ 5	9

6.1.3 Operand Difficulty Scores

The difficulty score for an operand is set equal to the sum of the products of the digit scores times their respective position scores for

6. Metrics and Generators

The following hypotheses are explicitly embodied in COACH:

1) By using a suitably chosen metric for each topic it is possible to assign numerical scores which represent relative difficulty for problems within each topic.

2) Problem generators can be constructed for topics with well-defined metrics.

6.1 Topic Metrics

The following considerations entered into the design of the heuristic metrics adopted for arithmetic:

1) The difficulty score for a problem should be a linear combination of the difficulty scores of its individual operands.

2) The linear combinations in 1) may be different for metrics in different topics.

3) The operand scores should be such that an n digit operand generally has a higher score than an m digit operand if $n > m$.

4) The fact that certain decimal digits are easier to work with than others should be taken into account. (It is easier for a student to manipulate the digit 2 in a problem than the digit 9.)

With these constraints in mind the actual metrics used in COACH were specified as follows:

6.1.1 Digit Scores

The digit scores assigned to decimal digits are the following:

	decimal digits		
digit score = 1		0	
digit score = 2	1	2	5
digit score = 3	4	6	8
digit score = 4	3	7	9

6.1.2 Position Scores

The digits in a number are identified by a particular position, for example the number 1492 has the digit 2 in position one, the digit 9 in position two, and so forth. Position scores are the following:

position	position score
1	2
2	3
3	5
4	7
≥ 5	9

6.1.3 Operand Difficulty Scores

The difficulty score for an operand is set equal to the sum of the products of the digit scores times their respective position scores for

each digit in the number. Thus the number 1492 has an operand

$$\begin{aligned} \text{difficulty score} &= (\text{digit score of } 2) \times 2 + \\ &(\text{digit score of } 9) \times 3 + (\text{digit score of } 4) \times 5 + \\ &(\text{digit score of } 1) \times 7 = 2 \times 2 + 4 \times 3 + 3 \times 5 + 2 \times 7 \\ &= 45. \end{aligned}$$

Similarly the number 16 has an operand difficulty score =
 $2 \times 2 + 3 \times 3 = 13.$

6.1.4 Problem Difficulty Scores

The addition and subtraction metrics used the sum of the operand difficulty scores for their problem difficulty scores. Thus $1492 + 16$ and $1492 - 16$ each have problem difficulty scores equal to 58. The multiplication metric uses the sum of the operand scores times 1.5 and the division metric takes the sum of the operand score for the denominator and twice the numerator, and multiplies this sum by two. These metrics are summarized in Figure 6.

$$\begin{aligned} 0 &= \text{operand score} = \sum (\text{digit scores} \times \text{position scores}) \\ \text{metric score for addition } (a+b+c+\dots) &= 0_a + 0_b + 0_c + \dots \\ \text{" " " subtraction } (a-b) &= 0_a + 0_b \\ \text{" " " multiplication } (a \cdot b) &= (0_a + 0_b) \times 1.5 \\ \text{" " " division } (a/b) &= (2 \times 0_a + 0_b) \times 2 \end{aligned}$$

Topic Metrics for Arithmetic

Figure 6

In practice the following situation can occur: three problems may be produced with difficulty scores 110, 115, 120 and the problem with difficulty score 120 appears (on a subjective basis) to be easier than the problem with difficulty score 110. If the difficulty scores have a wider spread, such as 110, 140, 170 their subjective relative difficulty rankings coincide with the ordering of their scores more closely. The reader may observe this phenomenon by studying the eight subtraction examples in Figure 4.

6.2 Question Generators

Using the topic metrics in Figure 6, question generators were constructed for COACH. The function of a question generator is to produce a problem within some specified range of difficulty. For example, if the question generator for addition were asked to produce a problem with a difficulty score between 55 and 60 it might generate the problem $1492 + 16$.

The information required by question generators consists of primary and secondary specifications. The only primary specifications currently used are the two score specifications for a range (upper and lower difficulty score bounds). Secondary specifications are peculiar to the topic. In arithmetic secondary specifications specify the maximum number of digits to use in operands for a problem.

For purposes of illustration we will now consider the operation of a particular question generator in detail:

6.2.1. The Question Generator for Addition Problems

A sample set of specifications to the question generator for addition could be the following:

Maximum problem score = 33	}	Primary
Minimum problem score = 30		
Maximum number of digits in first operand = 3	}	Secondary
Maximum number of digits in second operand = 3		

The generator first decides on the number of operands to use in the problem. The maximum-minimum scores lie in a difficulty level and associated with each such level is the maximum number of operands to use with problems in that level. The 30-33 range falls in level 1 and so exactly two operands are specified. If the range were in level 7 then up to six operands would be possible and the program (using a pseudo random number generator) would decide on the number of operands to actually try to use. Based on current parameter values, the program would generate a six-operand level 7 problem 60% of the time and one with fewer operands 40% of the time. Similarly, within the 40% cases in which a six operand level 7 problem was rejected, it would generate a five-operand level 7 problem 60% of the time and one with fewer operands 40% of the time, and so forth.

After deciding on the number of operands to use the generator considers specific difficulty scores for operands. It will make up to five cycles thru an assignment process that attempts to allocate difficulty scores so the problem difficulty score falls within the desired range. If the generator does not succeed it will arbitrarily accept the operand scores and accompanying problem generated during its last cycle.

On each cycle a bias factor is computed by taking the difference between the maximum and minimum range values and dividing it by the number of operands and the cycle number. Thus the denominator will increase in size in later cycles and the value associated with the bias factor will decrease. In our example the bias factor on the first cycle = $(33 - 30) / (2 + 1) = 1$.

A guess is made at the score for the first operand by taking the average operand scores (minimum problem score (30) divided by the number of operands (2) = 15) and adding a random integer in the range 1, 2, ... bias. In our example this means the first operand would have a difficulty score of $15 + 1 = 16$.

The difficulty score for the first operand is subtracted from the maximum problem score and the difference represents the amount of "problem difficulty" left to be distributed among the remaining operands.

The score for each remaining operand is determined by computing the average operand score for the remaining operands using the undistributed amount of "problem difficulty" and then adding to it a random integer in the range 1, 2, ... bias. Thus it is possible to have four of the five operands absorb all the "problem difficulty" in which case the problem is changed from one having five operands to one having only four.

This process reflects the nature of the metric chosen for the topic of addition. Similar considerations apply to other topics. Obviously other allocation schemes could be used, but the one described above appears to function well and it produces a nice variety of problems.

The generator is almost ready to use the MAKENUM routine (which is described in detail in section 6.2.2). This routine produces a generated number as output whose operand difficulty score is close to a requested score. The input to MAKENUM consists of a requested (operand difficulty) score and a limitation on the number of digits to place in the generated number.

Before using MAKENUM the generator, using the secondary input specifications, makes a guess on the number of digits to place in the generated number. For example, the maximum number of digits in our first operand is three. This figure will be used $2/3$ of the time. Two will be used $1/3 \times 2/3 = 2/9$ of the time, and one will be used $1/3 \times 1/3 = 1/9$ of the time. This decision process is similar to the one used in deciding the number of operands.

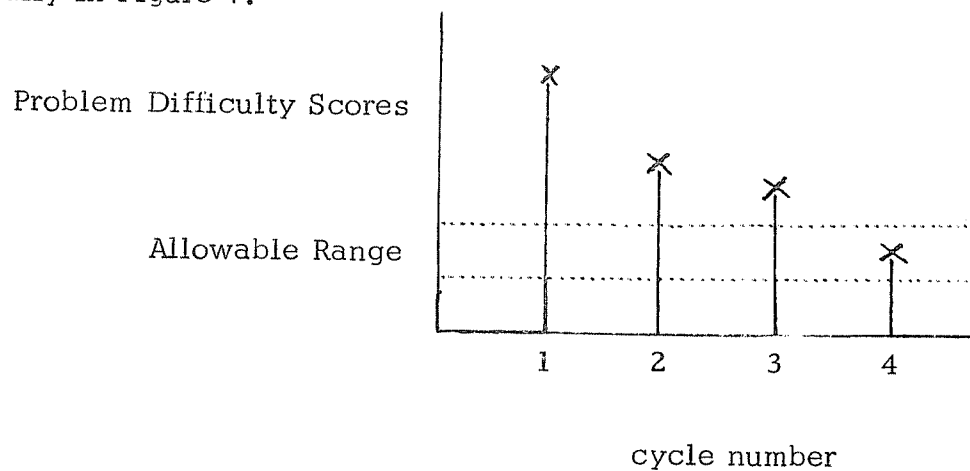
A problem is constructed using the actual operands produced by MAKENUM. This problem is checked to see if it falls in the allowable range and if not another cycle is made.

In our example, on the first cycle, the two operand scores input to MAKENUM were 16 and 18. MAKENUM generated the two numbers 67 and 78 with operand difficulty scores of 17 and 18 respectively. The problem difficulty score was 35 and it exceeded the maximum problem score. The generator increased the cycle count by one and looped back to recompute the bias factor. On the second cycle the operand input scores to MAKENUM were again 16 and 18 but this time the routine generated the numbers 66 and

78 with operand difficulty scores of 15 and 18. This made the problem difficulty score 33 and hence it was accepted by the generator.

The generator accepts the first problem that falls within the specified difficulty range. It then selects a question format to use in presenting the problem to the student, computes and stores the correct answer, lists the potentially applicable hints, and returns control to the main program.

The question generators for subtraction, multiplication, and division operate in a similar fashion. The problems generated on repeated cycles and their relation to the problem range are represented schematically in Figure 7:



Potential Problems Produced in a Question Generator

Figure 7

Because of the heuristic nature of the metrics and the threshold difficulty scores and the generating processes, it is possible to have generated problems in addition and division miss the allowable problem

range. This situation has been remedied by having the allowable range bounds vary by one percent in addition problems and by two percent in division problems.

In order to give the student some practice with negative numbers, level 7 and 8 subtraction problems may have the subtrahend greater than the minuend. Division problems in levels 1 thru 3 are constructed so the denominator always divides into the numerator an even number of times.

6.2.2 The MAKENUM Routine

The MAKENUM routine is used by all the question generators in COACH. Given a specific operand score and a limitation on the number of digits to use (referred to as N below) as input specifications the routine generates a specific number, i.e. it generates an operand.

MAKENUM begins by comparing the maximum number of digits to use with the desired operand score. If a minimum assignment of digit scores produces an operand score greater than that desired, N is decreased appropriately.

N (or its altered value) is now used to compute the maximum possible operand score for an N-digit number. If this is greater or equal to the input score N is left unchanged; otherwise N is increased until the maximum possible score equals or exceeds the input score specification.

The routine next computes an n-component (digit score) vector (n = current adjusted value of N) in which it places tentative guesses for digit scores. It selects a component randomly, inserts a digit score, computes

the resulting tentative operand score, chooses another component, and repeats the process. As each component guess is made the current operand score is compared with the input score and if the current operand score is less the process repeats; otherwise minimum digit scores are assigned to the remaining empty components.

The routine now compares the "tentative" operand score of the n-component (digit score) vector with the operand score input to the routine. If they are identical all is happiness and the digit selection phase of the routine is activated.

If the "tentative" operand score is different from the input operand score the routine will proceed to "jitter" component positions. (A maximum of five component positions will be jittered.) In "jittering" it selects a position and may increase or decrease the digit score by one. Then the routine compares the new "tentative" operand score with the input operand score, and if there has been a decrease in the difference between the two then the change in the digit score is accepted and jittering is repeated. When "jittering" is completed the digit selection phase of the routine is activated.

In the digit selection phase MAKENUM chooses specific digits for insertion in particular component positions by using a random selection process on the decimal numbers that have that particular digit score. This completes the operation of the routine.

A simple example of MAKENUM in operation now follows. We assume the operand input score was 18 and follow the process that produces the generated number of 78.

The maximum number of digits, \underline{N} , is set to 3 by the question generator, which in turn uses the secondary specifications.

Step 1: Check the minimum operand value for \underline{N} digits
 $= (5 + 3 + 2) \times 2 = 20$ which is greater than 18.

\underline{N} is decreased by one so $\underline{N} = 2$.

Step 2: Check the maximum operand score for current value of \underline{N} .

$= (3 + 2) \times 4 = 20$ which is greater than 18. Leave

\underline{N} unchanged.

Step 3: Set $n = \underline{N} = 2$.

Step 4: Choose a random position (1) and a random digit score (2).

Step 5: Choose a second random position (2) and another random digit score (4).

The n -component (digit score) vector now looks like

4	2
---	---

and its operand score is $2 \times 2 + 4 \times 3 = 16$

Step 6: Choose a component position and jitter; In this case position one is chosen, the digit score is incremented by 1 and the vector appears as

4	3
---	---

its operand score is now 18 and the difference is zero.

Step 7: Randomly choose a decimal number having digit score 3 (the number 8 is chosen) and randomly choose a decimal

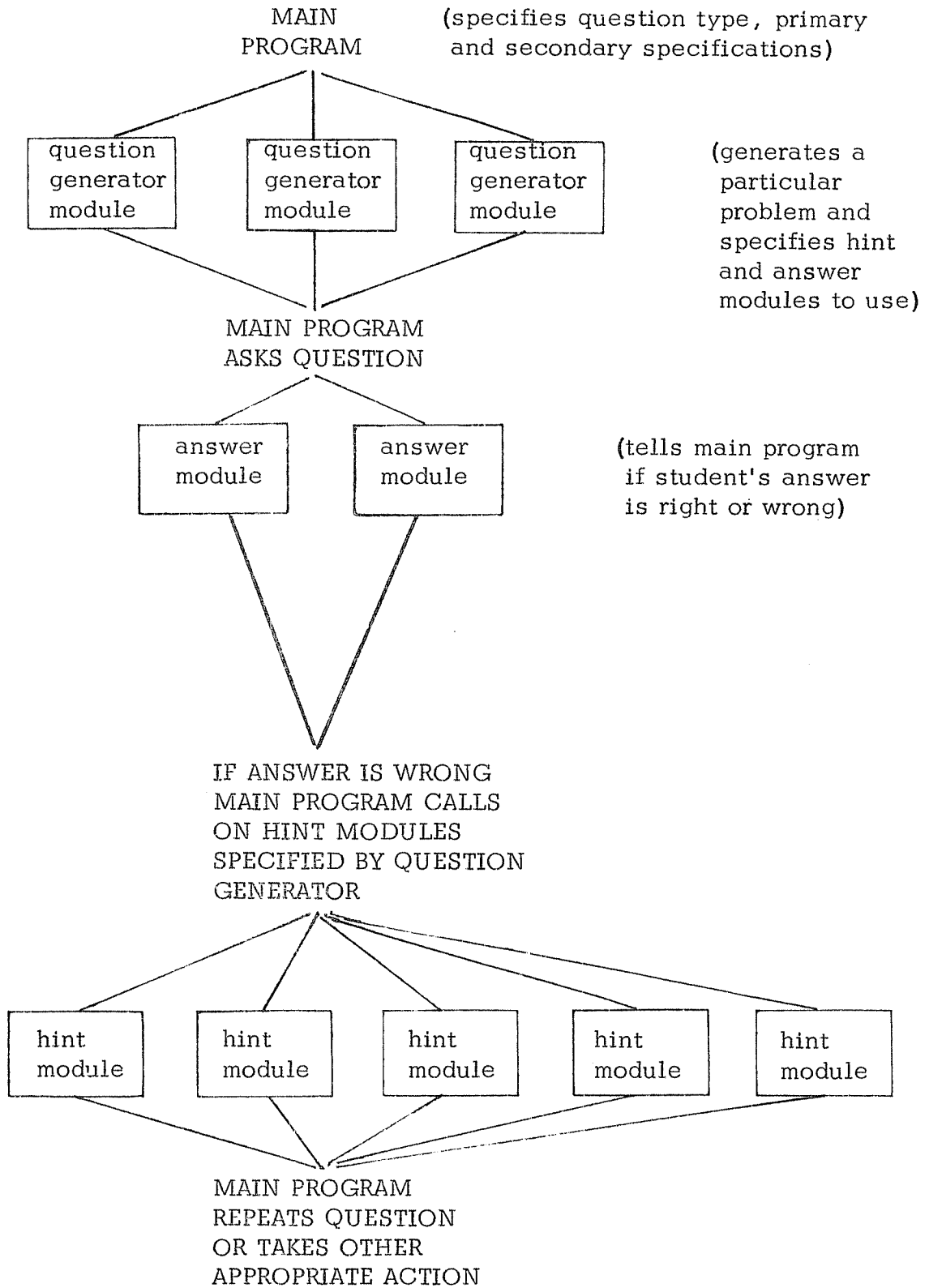
number having digit score 4 (the number 7 is chosen). Thus the number (i.e. operand) 78 has been generated.

7. Answer and Hint Modules

Answer modules are individual programs that compare the student's reply with the correct answer and determine if the two are the same. At the present time there are two such modules, one for problems in addition, subtraction, and multiplication (ASM) and one for problems in division.

The ASM answer module is quite straightforward. It merely scans the student's integer response and compares it with the correct answer. The module for division is somewhat more complicated because division problems may have remainders, for example, the problem $22/7 = 3.142857\dots$ may have its answer typed-in as 3 R 1 (three with remainder one) or the student may type-in a result that has been rounded to n decimal places ($n \leq 9$). Thus the student responses of 3 R 1, 3.1, 3.14, 3.143, 3.1429, 3.14286, 3.142857, etc. are recognized as correct answers for the problem $22/7$.

A schematic diagram depicting the relationship between the main program and the various types of modules in COACH is given in Figure 8. This organization allows special conventions to be established between particular generator, answer, and hint modules that are essentially independent of the main program. In such an environment modules can be constructed and added for other topics, additional problem types, more hints, etc. without the need for large changes in the system.



Ordering of Modules in the Flow of Control

Figure 8

The current version of COACH contains ten hint modules. Several of these compare the student's incorrect response with the correct answer and attempt to provide him with feedback on the cause of his error. Six hint modules are for addition, subtraction, and multiplication problems and perform the following functions:

- Hint Module 1: Types the correct answer.
- Hint Module 2: Types a message to the student telling him his answer was not correct. It repeats his incorrect answer, thereby providing a check for the case where hardware or data transmission failure caused the computer to receive a response different from that typed by the student.
- Hint Module 3: Informs the student his answer was too large and tells him whether this difference was small or fairly large.
- Hint Module 4: Informs the student his answer was too small and tells him whether this difference was small or fairly large.
- Hint Module 5: Checks to see if the sign of the student's answer is correct.
- Hint Module 6: Checks the digits in the student's answer with those in the correct answer and tells him how many are wrong. For example, if the correct answer was 1564 and he answered 1554 he would be told one of his digits was wrong; if he answered 564 he would

be told his answer did not have enough digits in it.

Additional hint modules can be added, such as one for addition that attempts to analyze a student's carrying mistakes, or one for subtraction to analyze mistakes in borrowing, however such programs have been postponed indefinitely as the present system operates in a reasonable manner without them.

Obviously question generator or hint modules could present additional factual or informative material before or during the time a topic was being covered. When a particular level of difficulty was reached a question generator might tell a student something about a specific sub-topic, e.g. it might explain how to do multiplication problems of the form $\frac{XX}{XX}$ if the student had been doing those of the form $\frac{XX}{X}$. The teaching strategies that decide when to present such material could be built into either the main program or specific modules. The present program does not present such informative material, although it could.

There are four hint modules for division:

Hint Module 7: Types the correct answer. If the student typed-in an incorrect answer with n decimal places, the module will type out the correct answer rounded to n decimal places.

Hint Module 8: Tells the student his answer is wrong. Its function is similar to hint module 2 although its detailed actions are slightly different.

Hint Module 9: Checks to see if the student's quotient is correct but the remainder is zero and wrong. If this occurs the student may simply have forgotten to type-in a remainder; he is reminded of this.

Hint Module 10: Checks to see whether the student's quotient and/or remainder is wrong and tells him whether his answer is low/high.

8. Miscellaneous Features

8.1 The Practice Option

In order to allow for repeated drills beginning at the same level but having different problem sequences an option was included so the student could type-in PRACTICE 1 at the time he specified level 2 problems in subtraction instead of typing-in only SUBTRACTION 2. A different but comparable sequence of problems would then be generated. At a later time he could type-in the messages SUBTRACTION 2 and PRACTICE 2 and receive yet another sequence that was different from the earlier two sequences. Similar remarks apply for the other topics and other levels of difficulty.

8.2 The Monitor Printer

As mentioned earlier, a copy of the teletype communication between the student/teacher and COACH is produced on the monitor printer. This copy includes additional information on the generated problems such as

their ranges, difficulty scores, and elapsed times.

8.3 Control Mode Options

A special control mode of operation can be activated (by typing-in the word CONTROLMODE). This allows the teacher/supervisor to make use of several special features.

8.3.1 Setting Probability Switches

The settings of the probability switches are initialized by the main program. In control mode it is possible to alter these values dynamically by typing a message like:

```
SET PROBPI = 20
```

This would set the threshold value for probability switch #1 (PSW #1) equal to a value of 20. A summary of the current initializations for probability switches is given in Table 2.

#1	(take question from OPMISLIST)	= 20
#2	(take question from REDOLIST)	= 10
#3	(probability of having COACH generate an incorrect answer while operating in the automatic answer mode - discussed below)	= 10
#4	(has no effect at the present time)	
#5	(add a question answered correctly the first time to REDOLIST)	= 40
#6	(no change in range, should another question of same type be asked)	= 10
#7	(range is to be dropped, should another question of same type be asked)	= 50
#8	(range is to be raised, should another question of same type be asked)	= 20

Initial Values of Probability Switches

Table 2

8.3.2 Setting INCREMENTSPERLEVEL

The number of ranges within a level of difficulty is determined by the variable INCREMENTSPERLEVEL mentioned in section 3. This variable is currently set to a value of 10. In control mode this value can be altered to 20 (or any other integer) by typing:

```
SET INCREMENTSPERLEVEL = 20
```

Making the variable larger would tend to cause the system to produce more problems in each difficulty level. A smaller value would cause it to generate fewer problems per level and the problems thus generated would be further apart from each other in difficulty scores.

8.3.3 Recovery Procedure

At periodic times during a student's run the system will save on disk all the restart information needed to restart the student at that point. This provides a partial recovery feature for a student in case a mechanical failure causes the computer/system to die. Up to ten students may have this information saved for them. A student will receive a warning at the start of his run if his restart information cannot be saved on disk but he will be able to continue the run. In control mode the teacher may type-in

```
DUMP NAMES
```

and receive a listing of the students having current restart information available. The teacher may also remove a student's file from disk by typing:

```
ERASE JOHN JONES
```

8.3.4 Debugging Aids

In control mode an automatic answering feature may be turned on by typing the message:

```
AUTOANS ON
```

This causes the program to activate another set of modules which use the correct answer to construct a pseudo-student response. Depending on the value of probability switch #3, the correct answer may or may not be permuted into an incorrect one. It is possible to view the role of the student as simply a supplier of answers which are either correct or incorrect. Then AUTOANS in combination with PSW #3 effectively serves as a simple model for a student. More importantly, AUTOANS is a valuable aid since it provides a way to study the action of particular metrics and problem generators in long problem sequences.

The other useful debugging aid in the program is a bit controlled dynamic trace option. Each routine in BABYG has a trace bit associated with it. In the event of a program malfunction it is possible to follow the problem sequence from the last restart point up to the point of danger, enter control mode, type-in:

```
TRACE 1 THRU 47 ON
```

and then continue running the program. A large body of additional information on the detailed functioning of the program is then generated on the monitor printer.

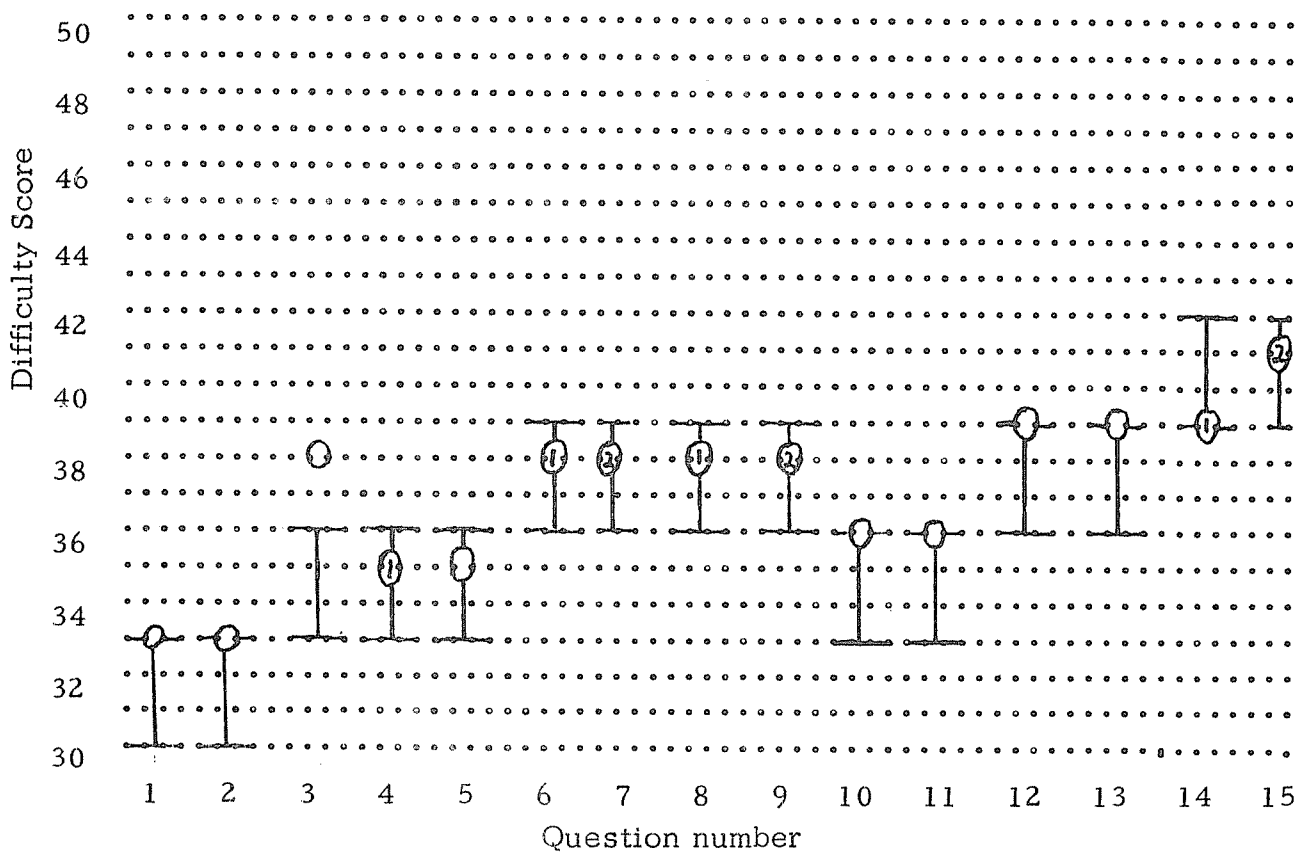
9. Current Results

Although the program is operating, few students have used it. Experimental runs have been made with some first, second, and fourth grade students. A profile for a second grader is given in Figure 9 for a run on addition 2 problems that lasted about half an hour. The following symbols are used in Figures 9, 10, 11:

I indicates the range within which problems should be generated.

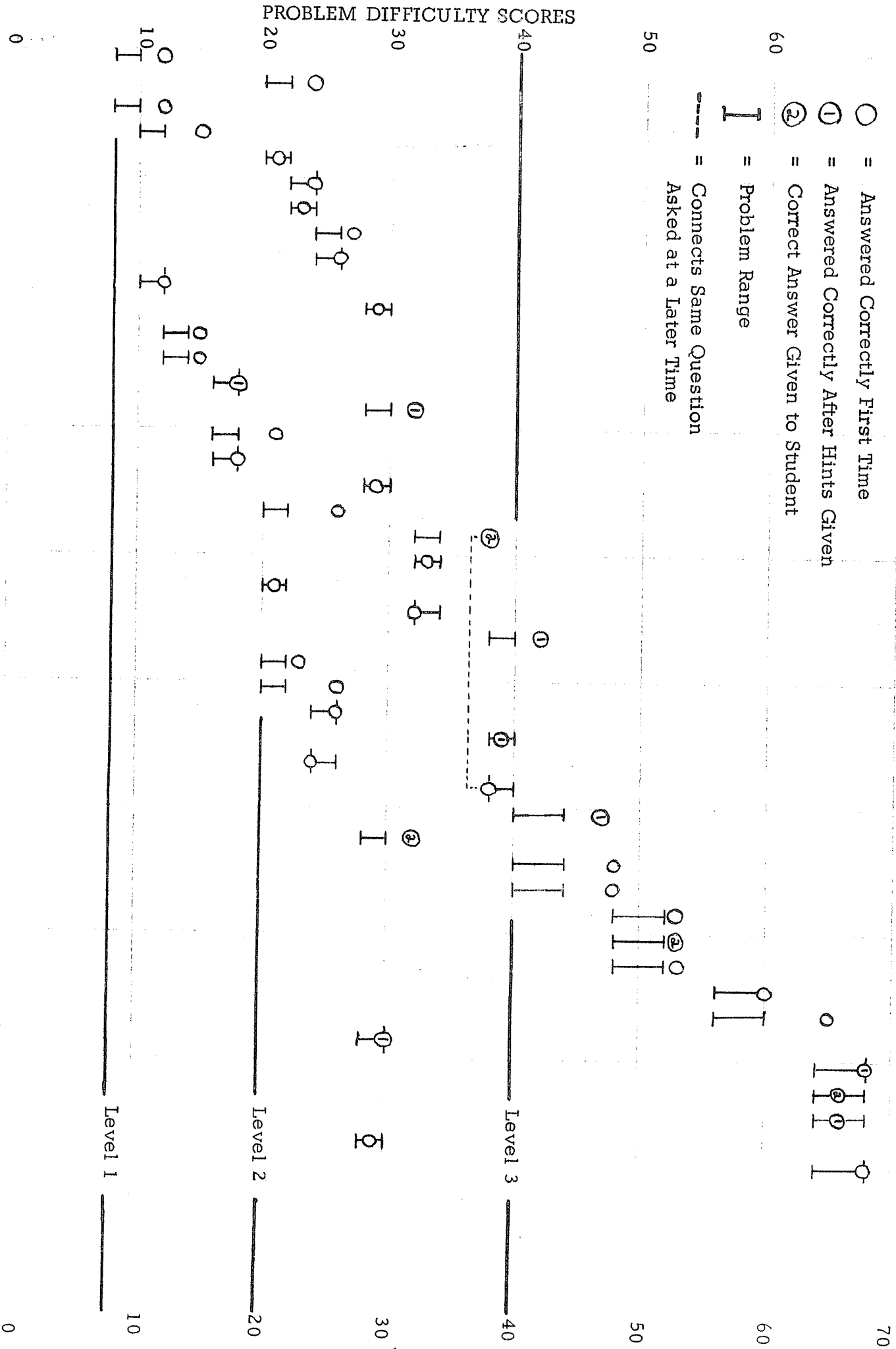
○, ⊙, ⊚ indicate the difficulty score of the generated problem and whether it was answered correctly the first time ○, or after hints were given ⊙, or after the correct answer was given to the student ⊚.

----- connects two instances of the same question which are asked at different times (from OPMISLIST or REDOLIST).



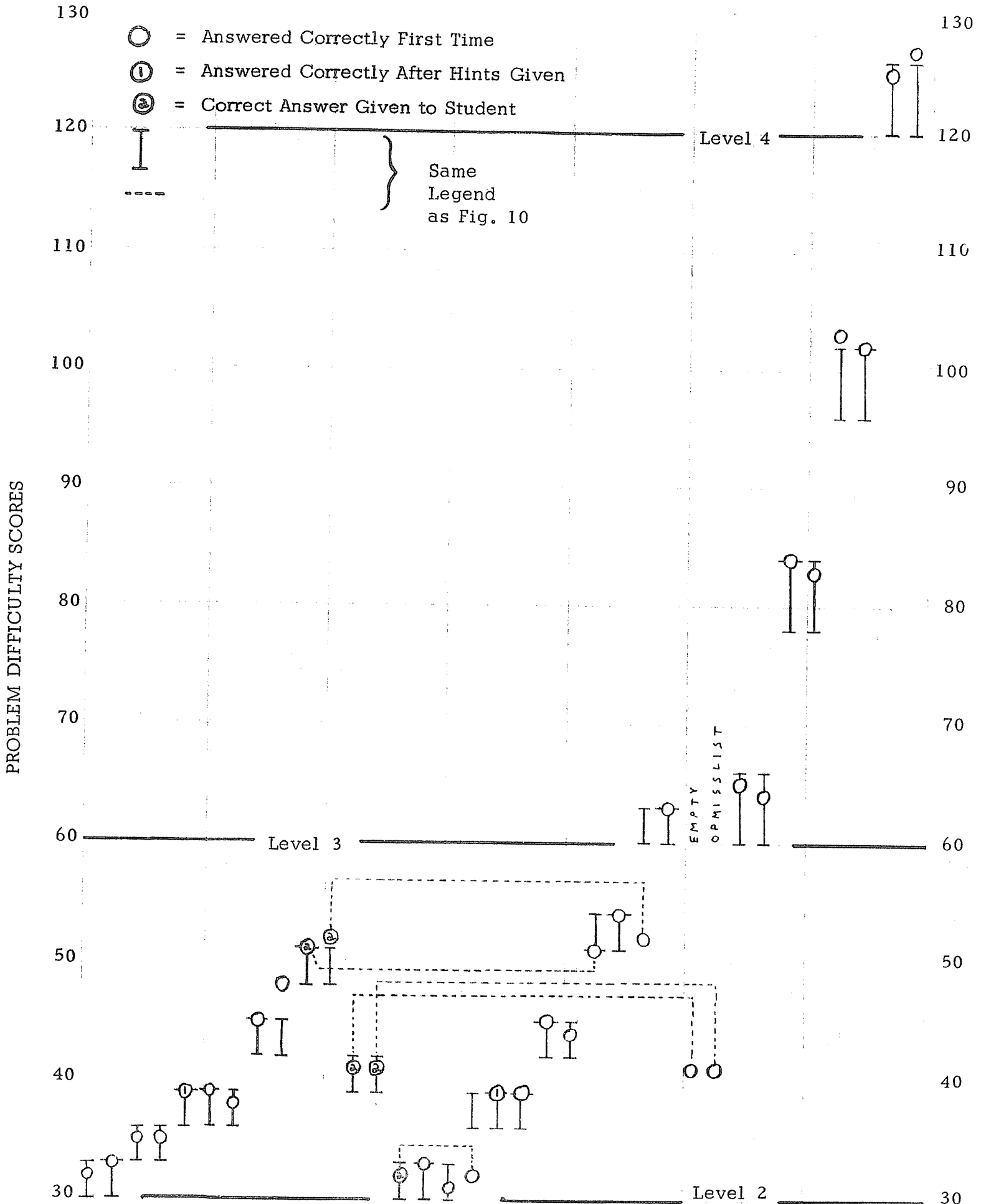
Half Hour Run on Addition 2

Figure 9



Intermixed Level 1 and 2 Multiplication Problems Done By A Fourth Grader

Figure 10



The results of a forty minute mixed run by a fourth grader are summarized in Figure 10. Figure 11 contains a longer run on addition starting with level 2. Initial reactions from most student users have been favorable. Many more runs have been made with COACH operating in the automatic answering mode.

It is difficult to make an accurate estimate of the cost per student per hour at the teletype, particularly since user demands on the local time sharing system may cause charged times to vary by a factor of ten or more, depending on what other programs are running at the same time. Local accounting practices are also a significant factor. Based on current operating experience the cost seems to be between \$5 and \$15 for each hour the student is at the teletype.

10. Conclusions

COACH has achieved its goals of automatically generating simple arithmetic problems using topic metrics and serving as a convenient vehicle for the study of some rudimentary problems in CAI. The most important questions revolve around the selection of topic metrics and the inverse generation procedures needed in order to produce questions. The procedures adopted in COACH are fairly specialized, reflecting the nature of the problem domain for study and other ad hoc considerations.

During the development of the program small changes in the metrics sometimes produced very large changes in the nature of generated problems. This sensitivity is undesirable but it is not immediately clear what can be done to circumvent the problem.

It would be nice if there was an easy way to construct metrics for more general problem domains. At the same time it would be nice to handle automatically the inverse construction problem of question generators using metrics. It is not at all clear at this time whether this can be done in any type of general fashion.

From the specialized metrics considered above the following comments may be made: numeric scores obtained from a metric should be useful; the components of a metric should be such that topical subgoal problems can be generated. Notice these components effectively constitute a hypothesis on the inherent structure of the problem domain.

The modular design of COACH helps to isolate certain areas in which little work has been done. For example, the construction of appropriate hint modules for problems generated automatically has not been investigated in any great detail yet this problem, among others, will need to be studied if truly interactive programs sensitive to student needs and requests are to be developed.

One of the largest drawbacks in the present program involves the addition of other topics. Making such additions presently requires a knowledge of Algol programming for the Burroughs B5500 as well as a knowledge of the conventions adopted in COACH, some of which are indicated in Appendix A. For this reason teachers are not in a position to add new topics to the system.

Present plans for future work are aimed at studying a specialized instructional environment in which skeleton patterns and other information provided by a teacher are used to construct questions and analyze incorrect answers .

11. Acknowledgements

I would like to thank Professor Leonard Uhr for his encouragement during the course of this project. Special thanks are due Tom Moran, Laird Beaver and other staff members at the University of Wisconsin Computing Center whose help and suggestions contributed significantly to program debugging.

This research has been partially supported by grants from the National Institutes of Health (MH 05254, MH 12266), the National Science Foundation (GP-7069 and the Wisconsin Alumni Research Foundation.

12. Bibliography

- (1) Feingold, S. L. PLANIT - A Flexible Language Designed for Computer-Human Interaction, in AFIPS Conference Proceedings Volume 31, Fall Joint Computer Conference 1967, Thompson Books, Washington, D. C.
- (2) Rath, G. J. and Anderson, N. S. and Brainerd, R. G. The IBM Research Center Teaching Machine Project in Automatic Teaching: The State of the Art, E. Galanter (ed), John Wiley & Sons, N. Y. 1959.
- (3) Smallwood, R. D. A Decision Structure for Teaching Machines, MIT Press, 1962.
- (4) Uhr, L. The Automatic Generation of Teaching Machine Programs, unpublished manuscript 1965.

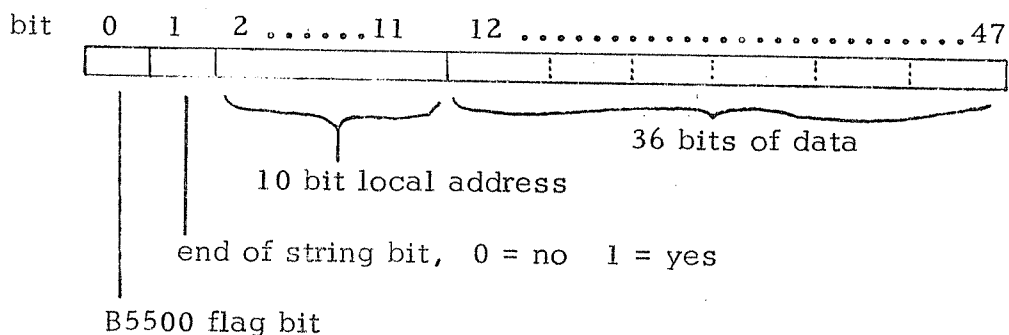
Appendix A Special Conventions in COACH

The following discussion of some of the detailed conventions adopted in programming COACH assumes the reader is familiar with Algol programming on the Burroughs B5500 computer, among other things.

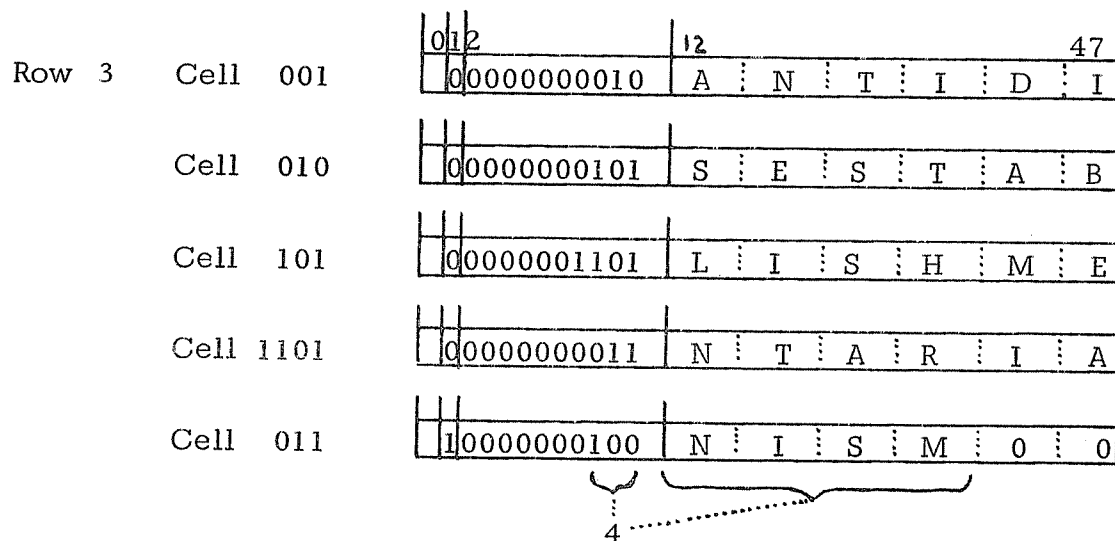
A.1 String/List Storage

In order to store strings of characters such as those in question formats, student responses, etc. a two dimensional array is defined, the X array with maximum dimensions of 256 rows and 1022 columns. Each cell in the array can hold six 6-bit characters of information or 36 bits of information. Cells may be linked together in strings and a ten-bit forward pointer is stored in the cell which points to the next cell in the string in the same row. Thus these local pointers have an implied 8-bit row specification. The end of a character string is indicated by a bit in the cell being set to one. In the last cell of a string the local address field holds a character count instead of a row address. The character count is zero (for 6 characters, i.e. full words) or one thru five.

Typical X array cell



For example the word ANTIDISESTABLISHMENTARIANISM might be stored as follows:



The eighteen bit binary string address of this word (henceforth simply referred to as its string address) would be

00000101	000000001
⏟	⏟
row	first cell in string

Cell 0 in each row keeps track of the available cells within that row. Unused cells are linked together on a list. Word 0 also contains the number of unused cells currently available in the row. If a string or list requires more cells in a row than currently available the next higher row will be tried or a new row will be opened. This organization provides the program with a large amount of working storage.

A.2 Internal Representation of Questions

A question occupies at least seven cells and has the following format in the Q array:

Q [0]	12	29: 30	47
	List left link	: List right link	
Q [1]	12	23: 24	47
	Topic code	: Problem difficulty score	
Q [2]	12	29: 30	
	Answer module to use	: Question format to use	
Q [3]	12	29: 30	47
	Hint List	: Parameter for hint list	
Q [4]	12	29: 30	47
	1-back student ans	: **2-back student ans**	
Q [5]	12	29: 30	47
	Correct answer	: Number of operands	
Q [6]	12	29: 30	47
	**Operand 1 **	: **Operand 2 **	
	----- and possibly -----		
Q [7]	12	29: 30	47
	**Operand 3 **	:	

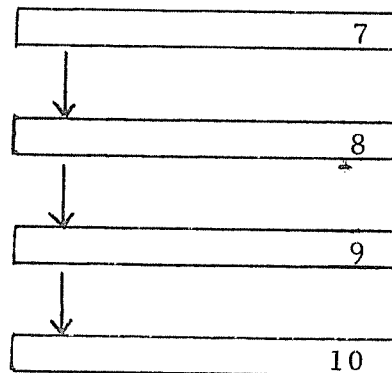
etc.

..... indicates this is an 18 bit string address

If a question is placed on OPMISSLIST or REDOLIST Q [0] holds the necessary linking information.

The topic code and problem difficulty scores are simple integers within their respective fields in Q [1], as are the answer module number and question format numbers in Q [2]. These question formats are discussed below in more detail.

The hints that may be applicable to the student's reply to the question are stored in a list. For example a division problem would have the hint list:



These four words would be linked in the X array and the eighteen bit address of the first one would be placed in Q [3]. By convention, the number placed in the first position of a hint list always tells the system which hint module can be called to provide the student with the correct answer. The hint list parameter in Q [3] is not used at the present time.

When a student misses a question his reply may be stored in the X array and the string address of it placed in Q [4]. Another incorrect reply may also be noted and stored for future use. At the present time little use is made of this historical information although some of the hint modules may compare his current incorrect reply with an earlier wrong answer and if they are the same then a comment on his inattentiveness may be generated.

The generator, answer, and hint modules may adopt private conventions on the way answers to a problem are to be transmitted from one module to another. For instance, with addition, subtraction, and multiplication the answer is a BCL string of digits and its string address

is stored in Q [5]. The division generator constructs a special two word list in which the quotient is specified in the first word (i.e. the first 6 characters) as an integer and the remainder is placed as an integer in the second word (i.e. the next 6 characters). The string address of this list is placed in Q [5].

The number of operands is specified as a simple integer in Q [5] and the string address of each operand is given using Q [6] and additional words if needed.

The seven or more cells in the Q array that define a question may be placed on a list which is stored in the X array. The question as a whole may then be manipulated as a compact entity (e.g. the string address of this question may be added to OPMISSLIST or REDOLIST).

A.3 Format Specifications for Teletype Questions

The design of the actual formats to use in order to pose questions on the teletype was separated from the main program in the following manner:

Questions often amount to simply inserting specific operands into an otherwise fixed format although the operands may vary in size, i.e. number of characters. A convention has been adopted so the BCL string for operand 1 has its string address placed in the first word of array P (the parameter array), the second operand has its string address placed in the second word of array P, and so forth.

The question generator specifies a particular format to use, say F [3] for an addition problem having three operands. These formats are defined on BCD cards and each card corresponds to one teletype line. It will become apparent that formats defined in such a manner can be changed with a minimum of effort.

```
FO3 ADD THESE NUMBERS
FO3   *P1* + *P2* + *P3* =
```

the operands 2,45,19 would cause the teletype to print

```
ADD THESE NUMBERS
  2 + 45 + 19 =
```

Format F [4] could be defined as

```
FO4   DO THE FOLLOWING (ADDITION) PROBLEM
FO4
FO4           *P1,5R*
FO4           *P2,5R*
FO4           + *P3,5R*
FO4           -----
```

and the three operands above would cause the following problem to appear on the teletype:

```
DO THE FOLLOWING (ADDITION) PROBLEM
```

```
      2
      45
+     19
-----
```

P1 says to insert the string for the first operand at that point.

P1,5 says the field for the first operand will contain at most 5 characters.

P1,5R says the operand within the five character field should be right-adjusted (L for left adjusted).

Normally the location of the left * of an operand placement specification in the BCD card also determines its location on the teletype output line, but this may be altered, for example *60P1* specifies the operand is to be inserted starting at column 60, regardless of the location of *60P1* on the BCD card.

Multiline comments for feedback, general information, and other purposes may be defined using C [1], C [2], ... in a way analogous to F [1], F [2], ... For example, when the student answers a question correctly on the first attempt the reinforcement comment that tells him he is correct is randomly selected from a list. This random selection factor helps minimize student boredom in such an environment.

Appendix B Student/Teacher Operation of COACH

1. All messages to the computer that are typed on the teletype must end with a left arrow (←).
2. Assuming the log-in process has been completed, i.e.

```
?BO ←
?LI #1704U0823 <password> ←
```

and the program has been called

```
??EXECUTE WEXLER/COACH ←
```

when it is ready to begin it will ask you for your name.

3. The program will present a list of possible course drills.
Type in a separate message for each one you want.
Type in a message PRACTICE <n> if desired, e.g. PRACTICE 2.
The computer will respond OK as each message is accepted.
4. When finished, type-in the message FIN and the computer will then present problems.
5. In answering a problem, the entire answer must be typed-in as one message, for example

$$\begin{array}{r} 5 \\ + 9 \\ \hline 14 \end{array}$$

The answer of fourteen must be typed-in as the digit 1 followed by the digit 4 followed by a left arrow (←).
Thus the answer must be given in a left to right fashion.

6. If you make an error in typing and have not typed the end-of-message symbol ← , then type a space and the four letter word TYPO and then ← . The teletype will position itself on a new line and you may then type-in the correct message.
7. When you wish to quit, type in the message QUIT instead of an answer.
8. When the computer types out

WEXLER/COACH ...EOJ.,. (end of job)

 you should type-in the message

?LO←

 in order to log-out.
9. If you wish to revise your course specifications and begin again, type in the message RESTART instead of an answer.
10. In order to enter control mode, type in the message CONTROLMODE instead of any expected response; to exit from control mode and return to the former place in the program, type the message RESUME, then give the response that was expected by the program before you entered control mode.

Unexpected Messages

1. HALT/LOAD ...
The system died.

Repeat the log-in and execute procedure, i.e.

```
?BO ←
?LI #1704U0823 <password> ←
??EXECUTE WEXLER/COACH ←
```

2. RESTART INFORMATION UPDATED

The program has arranged to save certain information. If a HALT/LOAD occurs it is possible to resume from this point.

3. WEXLER/SLOT... REMOVED

Ignore this message. It has no significance for you.

4. #FROM SPO: ... or #FROM 1/2: ...

An information message of some kind has been sent to you.

You may make a reply by typing

```
?SS SPO: <any message> ← or ?SS 1/2: <any message> ←
```

Appendix C Sample Teletype Output

FORM THE SUM
 270
 814

1084-

EXCELLENT - YOU ARE RIGHT

FORM THE SUM
 900
 807

1707-

CORRECT

FORM THE SUM
 752
 921

673-

YOUR ANSWER OF 673 IS INCORRECT

YOU ARE A LITTLE LOW

YOUR ANSWER DOES NOT HAVE ENOUGH NUMBERS

FORM THE SUM
 752
 921

1673-

CORRECT

FORM THE SUM
 277
 670

947-

GOOD - THAT IS THE RIGHT ANSWER

FORM THE SUM

644
486

1130-

EXCELLENT - YOU ARE RIGHT

YOU HAVE MOVED UP TO LEVEL 3 PROBLEMS IN ADDITION

DO THE FOLLOWING PROBLEM

$$555 + 227 + 156 =$$

938-

VERY GOOD - THAT IS THE CORRECT ANSWER

FORM THE SUM

297
677

974-

CORRECT

ADD THE FOLLOWING NUMBERS

529
661
186

1376-

CORRECT

ADD THE FOLLOWING NUMBERS

529
661
186

1376-

VERY GOOD - THAT IS THE CORRECT ANSWER

ADD THE FOLLOWING NUMBERS

682
716
577

1975-

EXCELLENT - YOU ARE RIGHT

FORM THE SUM
2442
4702

7144-

CORRECT

FORM THE SUM
6671
7704

13375-

YOUR ANSWER OF 13375 IS INCORRECT

YOUR ANSWER IS TOO LOW

ONE OF YOUR DIGITS IS WRONG

FORM THE SUM
6671
7704

14375-

THAT IS RIGHT

FORM THE SUM
6780
7576

14356-

VERY GOOD - THAT IS RIGHT

DO THE FOLLOWING (MULTIPLICATION) PROBLEM

$$\begin{array}{r} 3447 \\ \times 462 \\ \hline \end{array}$$

1592514~

CORRECT

DO THE FOLLOWING PROBLEM

$$\begin{array}{r} 4070200 \\ - 461216 \\ \hline \end{array}$$

3608984~

GOOD - THAT IS THE RIGHT ANSWER

DO THE FOLLOWING PROBLEM

$$567 \text{ DIVIDED INTO } 2261 =$$

3.97~

YOUR ANSWER OF 3.97 IS NOT CORRECT

YOU ARE JUST A LITTLE BIT LOW

DO THE FOLLOWING PROBLEM

$$567 \text{ DIVIDED INTO } 2261 =$$

3.984~

THE CORRECT ANSWER IS 3.988

YOUR ANSWER OF 3.984 IS NOT CORRECT

YOU ARE JUST A LITTLE BIT LOW

DO THE FOLLOWING PROBLEM

$$567 \text{ DIVIDED INTO } 2261 =$$

3.99~

THAT IS OBVIOUSLY CORRECT

ADD THESE NUMBERS

671
894
878
491
695

3629*

CORRECT

DO THE FOLLOWING PROBLEM

452305
- 75729

4476576*

CORRECT

ADD THESE NUMBERS

761
981
696
678
940

4056*

CORRECT

DO THE FOLLOWING PROBLEM

557 DIVIDED INTO 1532 =

2.75045*

CORRECT

DO THE FOLLOWING (MULTIPLICATION) PROBLEM

824
X 773

636952*

VERY GOOD - THAT IS THE CORRECT ANSWER

Appendix D Sample Monitor Printer Output

ELAPSED PROCESSOR TIME = 61.78 SECONDS I/O TIME = 145.28 SECONDS
CORRECTS#<

MAXIMUM = 60 MINIMUM = 57 HISCORE = 60 LOWSCORE = 30 PROBLEM SCORE = 59

ELAPSED PROCESSOR TIME = 63.47 SECONDS I/O TIME = 149.96 SECONDS
FORM THE SUMS#<
277<#<
670<#<
#####<

947<

ELAPSED PROCESSOR TIME = 66.12 SECONDS I/O TIME = 157.00 SECONDS

GOOD - THAT IS THE RIGHT ANSWERS#<

MAXIMUM = 60 MINIMUM = 57 HISCORE = 60 LOWSCORE = 30 PROBLEM SCORE = 60

ELAPSED PROCESSOR TIME = 68.53 SECONDS I/O TIME = 163.15 SECONDS
FORM THE SUMS#<
644<#<
486<#<
#####<

1130<

ELAPSED PROCESSOR TIME = 71.00 SECONDS I/O TIME = 168.93 SECONDS

EXCELLENT - YOU ARE RIGHTS#<

YOU HAVE MOVED UP TO LEVEL 3 PROBLEMS IN ADDITIONS#<

MAXIMUM = 66 MINIMUM = 60 HISCORE = 120 LOWSCORE = 60 PROBLEM SCORE = 66

ELAPSED PROCESSOR TIME = 73.75 SECONDS I/O TIME = 175.37 SECONDS
DO THE FOLLOWING PROBLEMS#<
555 + 227 + 156 =S#<
#####<

938<

ELAPSED PROCESSOR TIME = 75.77 SECONDS I/O TIME = 179.97 SECONDS

VERY GOOD - THAT IS THE CORRECT ANSWERS#<

MAXIMUM = 66 MINIMUM = 60 HISCORE = 120 LOWSCORE = 60 PROBLEM SCORE = 65
ELAPSED PROCESSOR TIME = 77.13 SECONDS T/O TIME = 183.00 SECONDS

FORM THE SUMS#<
297<#<
677<#<
#####

974<

ELAPSED PROCESSOR TIME = 79.72 SECONDS T/O TIME = 189.08 SECONDS

CORRECTS#<

MAXIMUM = 78 MINIMUM = 72 HISCORE = 120 LOWSCORE = 60 PROBLEM SCORE = 77

ELAPSED PROCESSOR TIME = 81.65 SECONDS T/O TIME = 194.70 SECONDS

ADD THE FOLLOWING NUMBERS<#<
529<#<
651<#<
186<#<
#####

1376<

ELAPSED PROCESSOR TIME = 85.08 SECONDS T/O TIME = 201.72 SECONDS

CORRECTS#<

MAXIMUM = 78 MINIMUM = 72 HISCORE = 120 LOWSCORE = 60 PROBLEM SCORE = 77

ELAPSED PROCESSOR TIME = 86.43 SECONDS T/O TIME = 205.12 SECONDS

ADD THE FOLLOWING NUMBERS<#<
529<#<
661<#<
186<#<
#####

1376<

ELAPSED PROCESSOR TIME = 87.67 SECONDS T/O TIME = 207.63 SECONDS

VERY GOOD - THAT IS THE CORRECT ANSWERS#<

MAXIMUM = 126 MINIMUM = 123 HISCORE = 150 LOWSCORE = 120 PROBLEM SCORE = 132

ELAPSED PROCESSOR TIME = 162.95 SECONDS I/O TIME = 339.63 SECONDS

DO THE FOLLOWING (MULTIPLICATION) PROBLEMS#

3475#
X 4625#

00005#

1592514

ELAPSED PROCESSOR TIME = 163.98 SECONDS I/O TIME = 340.97 SECONDS

CORRECTS#

MAXIMUM = 185 MINIMUM = 180 HISCORE = 225 LOWSCORE = 175 PROBLEM SCORE = 184

ELAPSED PROCESSOR TIME = 164.48 SECONDS I/O TIME = 341.63 SECONDS

DO THE FOLLOWING PROBLEMS#

40702005#
0 4612165#

000005#

3608984

ELAPSED PROCESSOR TIME = 165.13 SECONDS I/O TIME = 341.92 SECONDS

GOOD - THAT IS THE RIGHT ANSWERS#

MAXIMUM = 206 MINIMUM = 200 HISCORE = 260 LOWSCORE = 200 PROBLEM SCORE = 202

ELAPSED PROCESSOR TIME = 165.67 SECONDS I/O TIME = 342.28 SECONDS

DO THE FOLLOWING PROBLEMS#
567 DIVIDED INTO 2261 =S#

3.97

ELAPSED PROCESSOR TIME = 166.52 SECONDS I/O TIME = 342.45 SECONDS

YOUR ANSWER OF 3.97 IS NOT CORRECTS#

YOU ARE JUST A LITTLE BIT LOWS#

MAXIMUM = 206 MINIMUM = 200 HISCORE = 260 LOWSCORE = 200 PROBLEM SCORE = 202

ELAPSED PROCESSOR TIME = 167.20 SECONDS I/O TIME = 343.63 SECONDS

DO THE FOLLOWING PROBLEMS#<
567 DIVIDED INTO 2261 =5#<

3.984<

ELAPSED PROCESSOR TIME = 167.85 SECONDS T/O TIME = 343.88 SECONDS

THE CORRECT ANSWER IS 3.984#<

YOUR ANSWER OF 3.984 IS NOT CORRECTS#<

YOU ARE JUST A LITTLE BIT LOWS#<

MAXIMUM = 206 MINIMUM = 200 HISCORE = 260 LOWSCORE = 200 PROBLEM SCORE = 202

ELAPSED PROCESSOR TIME = 168.48 SECONDS T/O TIME = 344.25 SECONDS

DO THE FOLLOWING PROBLEMS#<
567 DIVIDED INTO 2261 =5#<

3.99<

ELAPSED PROCESSOR TIME = 169.75 SECONDS T/O TIME = 346.63 SECONDS

THAT IS OBVIOUSLY CORRECTS#<

MAXIMUM = 160 MINIMUM = 155 HISCORE = 200 LOWSCORE = 150 PROBLEM SCORE = 159

ELAPSED PROCESSOR TIME = 170.72 SECONDS T/O TIME = 348.60 SECONDS

ADD THESE NUMBERS#<

6715#<
8945#<
8785#<
4915#<
6955#<
S#<
#####S#<

3629<

ELAPSED PROCESSOR TIME = 171.78 SECONDS T/O TIME = 349.53 SECONDS

CORRECTS#<

