INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700  800/521-0600
Hypothesis selection strategies in a debugging task

Ashby, Alan Bradley, M.A.
Rice University, 1989
RICE UNIVERSITY

HYPOTHESIS SELECTION STRATEGIES IN A DEBUGGING TASK

by

ALAN BRADLEY ASHBY

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE MASTER OF ARTS

APPROVED, THESIS COMMITTEE

[Signatures]

David Lane, Associate Professor of Psychology, Chair
William C. Howell, Professor of Psychology
John Brelsford, Professor of Psychology

Houston, Texas
July, 1988
HYPOTHESIS SELECTION STRATEGIES IN A DEBUGGING TASK

ALAN BRADLEY ASHBY

Abstract

Previous research on the human process of debugging indicates that it can be described as a process of testing hypotheses. It is noted that Hypothesis Testing Theory (HTT) as proposed by Levine (1966) can provide a guiding framework to study debugging. Experiments 1 and 2 tested the following hypothesis: Given a very simple debugging task for which there is a clearly defined optimal strategy for solution, would people discover and use that strategy. Subjects tended to discover and use the optimal strategy in experiment 1, but not for the slightly more complex task of experiment 2. Experiments 3 and 4 tested the assumptions of HTT of domain sampling on a task similar to the ones used in Experiments 1 and 2. The assumptions of HTT were supported by the results of experiment 3. All of the subjects in experiment 4 used a sequential search strategy. The results of each experiment are discussed and additional research is proposed.
Contents

Introduction_________________________1
Experiment 1_________________________16
Experiment 2_________________________25
Experiment 3_________________________29
Experiment 4_________________________34
General Discussion____________________39
References__________________________45
List of Figures

1. Typical problem display used in experiment 1. ________19
2. Typical testing display used in experiment 1. ________20
3. Number of subjects following various strategies as
   a function of problem number for experiment 1. ________22
4. Number of subjects following various strategies as
   a function of problem number for experiment 2. ________28
5. Typical problem display used in experiment 3. ________32
6. Typical problem display used in experiment 4. ________37
Introduction

After spending three billion dollars for weapons software in 1980, it has been estimated that the Department of Defense will spend around 30 billion dollars on software in the year 1990 (Dunn, pg. 5). This is just one illustration of the expected growth in software systems over the next few years. It is likely that debugging, the process of finding the cause of software failures, will account for a large portion of the increasing cost of software development.

Brooks (1972) cites several statistics on the development of several large software projects that support the thesis that debugging costs are a major part of software costs. Approximately half of all the person-hours spent during the development of a software project is spent debugging. Maintenance of a software system after development costs 40% or more of the cost of development. Fixing a bug has a 20%-50% chance of introducing another bug, while the time to find each succeeding bug grows exponentially.

However, there is not a large body of research on the process of software debugging. The few studies on the human process of debugging software that do exist share a common theme. A macro description of the debugging process is derived from a protocol analysis. Subjects' apportionment of time is observed
and various phases of debugging are identified. A common variation of this theme is to compare experts vs. novices on several dimensions as they debug several programs (Gugerty & Olsen, 1986; Kessler & Anderson, 1986). A typical conclusion is that debugging is a cognitive task that involves hypothesis testing and inferring global attributes of the program in question (Bois & Gould, 1974; Gould, 1975; Gugerty & Olsen, 1986; Kessler & Anderson, 1986).

Gould (1975) studied ten experienced programmers as they debugged 12 Fortran programs with various types of bugs. He reported that subjects generally selected a strategy to find clues based on studying the code in question, then after finding a clue or clues, develop and test a hypothesis about the bug with the information obtained. It was noted that a sequential search of the statements for the "bug" was a common strategy for three of the seven programmers.

Kessler and Anderson (1986) studied eight novice programmers as they debugged six Lisp functions. The subjects had had at most one introductory Pascal course and did not have any prior experience with Lisp. Using protocol analysis, Kessler and Anderson reported that the process of debugging could be broken down into the following four "episodes": code comprehension, bug detection, bug localization, and bug repair.
The percentage of time spent in each phase was not reported, but bug localization and bug repair were reported as the bulk of the protocol analysis data. Bug localization was also reported as the most "difficult" episode with the subjects usually locating the bug after "going through some iterations of creating and rejecting hypothesis."

Gugerty and Olsen (1986) ran two experiments comparing differences between skilled and novice programmers. The skilled programmers were advanced graduate computer science students while the novice programmers had had just one programming course. In the first experiment, 18 novices and six experts debugged three Logo programs while in the second experiment ten novices and ten experts debugged a single Pascal program.

In the first experiment, experts found the bug 89% of the time averaging seven minutes per solution while novices found the bug 72% of the time and averaged 18.2 minutes per solution. Experts tested an average of 1.9 hypotheses per program as compared to 4.5 for novices. These results were replicated in the second experiment with nine of the ten experts finding the bug in the allotted time and only five of the ten novices finding the bug in the allotted time. For those subjects who found the bug, experts averaged 14.2 minutes to find the bug and novices averaged 33.1 minutes. Experts also averaged 1.2 hypotheses
tested before finding the bug while novices averaged 1.8 hypotheses tested. Both experts and novices usually spent 70% of the time allocated studying the program before testing their first hypothesis. Novices also had a tendency to introduce their own bugs while testing hypotheses that subsequently went uncorrected.

The results of Gugerty and Olsen indicate the same pattern of behavior as the results reported by Kessler and Anderson (1986). Subjects began the debugging session by studying the program in question, followed by iteratively testing hypotheses until the "bug" was found. Experts and novices did not differ in the pattern of behavior, but in the amount of time allocated for code comprehension, the time required to test hypotheses, and the number of hypotheses tested. Experts tended to study the program longer, test hypotheses faster, and test fewer hypothesis before the "bug" was found. A hypothesis being defined in this study as an explicit change in the program being tested. However, it cannot be determined if experts actually tested fewer hypotheses since they might have tested hypotheses by tracing program behavior in their minds. This idea is supported by the large percentage of experts correcting the program on the first hypothesis. Results from this series of experiments indicate no qualitative differences between experts and novices, but that experts are faster at the task.
Jeffries (1982) has also reported similar findings. Six experts composed of graduate students in computer science and four novices who had just finished their first introductory computer science course used similar strategies while debugging two short pascal programs. However, the experts spent more time comprehending the programs, found the "bug" quicker, and remembered more details of the program in question.

The pattern of results in the debugging research literature are remarkably consistent so far. However, the studies described so far have concentrated on a macro description of the debugging process by means of a protocol analysis. Hypothesis generation and testing are commonly identified sub-tasks. Little or no systematic investigation of the sub-tasks involved in the process of debugging has been done.

It would be expected that a larger body of research would exist on the human process of debugging. Unfortunately this is not the case. One reason for this lack of research is that people in the computer field generally believe that debugging is an art and is not subject to systematic investigation. This attitude is illustrated by the following quote from a book on software testing by Dunn (1984, pg. 220):" Unfortunately, the definitive guide to debugging has never been written, nor is it likely to be. A
debugging Baedeker is about as plausible a project as a handbook of problems of the class crimes and for the same reason."

A related area of research that appears related to software debugging is systems fault diagnostics and reliability theory. In this body of literature, several assumptions about the system are made. The individual components of the system have a known life expectancy or failure rate and the system is usually one copy of a design whose performance characteristics are well known. Fault localization is then a straightforward quantitative probability analysis to determine which component to test or replace (Bazovsky, 1961; Barlow, 1965).

Not surprisingly, empirical research on fault diagnostics assume the above model of fault diagnostics. Bond and Rigney (1966) reported that subjects were non-Bayesian when judging the probabilities of symptoms given certain failures. Navy technicians judged the probabilities of various symptoms given specific failures. Actual testing behavior was compared with testing behavior predicted by Bayes’ theorem. The agreement between actual and predicted behavior was about 50%.

Mills (1971) reported subjects were non-optimal in estimating the likelihood that various components will fail. Subjects were asked to locate the faulty component in a chain of components. Each component had a different probability of
failure. Over a sequence of trials, subjects took the failure probabilities into account, but were not optimal.

Several studies have also looked at hypothesis generation in troubleshooting tasks. Rouse (1978) has noted that people are more likely to make use of positive information than negative information in forming hypotheses while troubleshooting. Positive information is information about what has failed while negative information is information about what has not failed. When given information about components that could or could not have been part of a failure, subjects did not use information about components that could not have been part of the failure. This information could have been used to prune the number of hypothesis generated.

Mehle (1980) has also noted that people generate incomplete lists of hypotheses to test while troubleshooting. Experienced and novice mechanics were asked to list possible causes of a given problem on an automobile. The lists generated were generally incomplete. The subjects were confident in the completeness of their lists, however.

Other research topics in fault diagnostics include individual differences and instruction. Individual differences identified to have an affect on trouble-shooting skill include mental models of the components being tested and the performance characteristics
of the system tested (Gitomer, in press). More skilled electronic troubleshooters had errors that were more computational than conceptual than less skilled troubleshooters. The amount and type of information during instruction about the system to be tested has also been studied extensively. Morris and Rouse (1985) provides a review of this research.

However, most software systems are unique systems where estimations of probabilities of component failures do not apply. Myers (1976) has noted that a software fault is a design fault while faults in other systems are mainly due to manufacture errors or failure. A statement in a software program is rarely wrong because of an error in duplication when the program is distributed. Such errors are easy to diagnose and correct when they do occur. Likewise, a statement never wears out due to it being executed repeatedly. Thus, caution should be used when generalizing the results from the task domain of fault diagnostics to that of software debugging. Still, research in failure-rate models have identified that hypothesis generation and testing is an important component in human fault diagnostics.

Software testing and reliability research has concentrated on discovering if program faults exist, not on the process of finding the faults. This distinction between bug detection and bug localization has emerged gradually over time. Initially, software
testing and software debugging were considered the same activity and the terms were used interchangeably (Gelperin & Hetzel, 1988). Software testing and debugging was considered a necessary but informal part of program development. This period of software testing has been identified as the "Debugging Period" of software testing and lasted until 1956.

Later periods of software testing identified by Gelperin and Hetzel share common characteristics. The process of testing software is considered a separate process from that of software debugging. The importance of software testing was recognized and greater emphasis was placed on software testing as a formal part program development.

The test period that followed the "Debugging Period" was identified as the "Demonstration-Oriented Period". During this period, testing was viewed as a process of demonstrating that the program meets it specifications. Debugging was considered the process of getting the program to run, while testing was considered the process of making sure the program solves the problem. In this period, both debugging and testing included bug detection, bug localization, and bug correction.

Myers (1979) ushered in the "Destruction-Oriented" period of software testing. Testing was viewed as the process of running a program with suitable test cases for the purpose of discovering
bugs. The distinction between testing and debugging was refined further. The process of software testing is the process of bug detection. The process of software debugging is the process of bug localization. Later periods of software testing share this distinction of bug detection and bug localization. Software testing and reliability research today concentrate mainly on estimating the number of bugs remaining and and determining if a set of inputs test a program adequately (Hamlet, 1988; Gelperin & Hetzel, 1988).

As noted by Kessler and Anderson (1986), the "bug" localization phase is the most difficult. It is also the most prevalent behavior noted in the studies of software debugging and appears to show the greatest differences between novices and experts. This phase can be characterized as a sequence of generating and testing hypotheses. Hypothesis generation has also been identified as an important part of human troubleshooting behavior. Hence, a more systematic investigation of the process of generating and testing of hypothesis is in order.

There has been considerable research on problem solving in cognitive psychology. Several aspects of problem solving has been identified. Among these are perception of the problem domain (Chase & Simon, 1973), internal representations of the problem (McCloskey, 1983; Miller, 1956), individual differences
in capabilities (Sternberg & Weil, 1980), and mental procedures (Newell & Simon, 1972). Central to major theories of problem-solving is hypothesis generation and testing behavior (Levine, 1975; Newell & Simon, 1972).

Newell and Simon (1972) characterize problem solving as a search for the solution of a problem over a problem space. A person's perception of the problem, internal representation of the problem, and repertoire of mental procedures all influence the definition of the problem space and how it is searched for the solution. The problem space contains not only the solution of the problem, but the set of solutions a person might consider as a candidate solution. Hence problem solving might be described simply as a repeated sampling of possible solutions to the problem from some set of possible solutions. This can also be stated as hypothesis generation and testing.

A particularly relevant line of research is the hypothesis testing theory (HTT) of learning as proposed by Levine, 1975. This theory is based on research of human discrimination learning (Levine, 1966) and has roots in the previous research of Krechevsky (1932a) and Harlow (1950).

Research on hypothesis testing began with Krechevsky (1932b) studying stimulus discrimination in rats. In this paradigm, rats decide which alley to follow to reach food on the
basis of some sensory discrimination. A typical "run" actually consisted of a series of choices between a series of two alternative pathways. Krechevsky reported systematic variations in the errors the rats made. Krechevsky rejected the current accepted theory that learning consisted of a gradual build-up of stimulus-response connections. Krechevsky proposed that rats systematically attempt various solutions and reject a candidate solution when they fail.

Harlow (1950) studied the errors made by monkeys in a sensory discrimination task. Harlow concluded that the errors made were not due to chance or random behavior, but due to the presence of underlying "error" factors. Among these was a tendency to "try out" or "explore" stimuli in a discrimination problem.

Levine integrated the work on hypotheses in discrimination learning into a coherent theory. Hypothesis testing theory makes certain assumptions on the generation and testing of hypotheses based on domains. According to hypothesis testing theory, people sample from a "domain" of solutions when attempting to solve a problem. A "domain" being loosely defined as a related set of hypothesis and may overlap.

Two of Levine's (1974) assumptions about hypothesis testing are particularly relevant to the study of debugging: (a) People
infer the solution domain of the n+1st problem from the solution domains of the previous n problems (the transfer assumption) and (b) People only switch solution domains when the current domain is exhausted (the empty-set assumption.) These assumptions elegantly explain "Einstellung" or psychic blindness (Luchins, 1942; Levine, 1971; Fingerman & Levine, 1974), the tendency of people to overlook simple solutions to problems after solving problems that have complex solutions. For example, people given a series of mazes requiring a circuitous solution path will attempt a circuitous solution for the next maze given and completely overlook a straight line solution path (Luchins, 1942).

According to infinite-set assumption of HTT, the solution may never be reached if a person initially samples from a large or infinite domain that does not contain the required solution. Since a switch to a different domain occurs upon exhaustion of the current domain, the person will continue to sample from this wrong initial domain and never arrive at a correct solution. Exhaustion of the domain would never occur in an infinite domain, while resampling used hypotheses might occur in a large domain (Levine 1971; Fingerman & Levine, 1974).

A study by Levine (1971) illustrates these assumptions and the paradigm used. A deck of cards was prepared with half of the cards containing an "A" on one side and a "B" on the other side
and the other half of cards, vice versa. The order of the cards was random. On a problem, a series of cards was presented to a subject, and the subject was to respond with one of the two letters on the card. For each card, presented the subject was informed if the response was correct. Subjects were presented a series of six training problems in which the correct solution was from a set of position sequences, such left-right-left-right-... A switch problem was then given where the solution was from the set "always say A" or "always say B". Levine found that 80% of the subjects failed to find the solution within 100 problems. The switch problem, if given initially, would usually be solved immediately making their final error on the second problem.

This type of behavior has been observed casually by the author while working at a user clinic in an academic computing center. The job entailed helping computer users having various levels of experience with problems they could not overcome by themselves. A common occurrence was a user asking for help with a problem which turned out to be something simple, much to the embarrassment of the user. The last part of the previous statement was added only to point out the fact that the problem was usually of the type that the user felt that it should have been found without help.
Myers (1979) has also noted the same behavior in software debugging and has offered hints on overcoming this tendency. If an impasse in debugging a program occurs, often describing the problem to someone else helps discover the solution. "Forgetting the problem" by leaving it alone for awhile, such as overnight, often helps to discover the solution as well. Myers (1979) also recommends two methods for debugging software. Inductive debugging is the method of organizing relevant data and repeatedly testing hypotheses generated by the data. Deductive debugging is the method of listing all possible hypotheses first and using data generated to prune the hypothesis set. The remaining hypothesis are then tested and pruned further. In both cases, hypothesis testing plays an integral role.

It appears then that Hypothesis Testing Theory would prove useful as a guiding framework to explore software debugging. The following four experiments reported here used HTT to investigate hypothesis selection strategies used in solving a debugging task. Using the terminology of HTT, "strategy" is being defined here as the sequence of domains selected as well as the sequence of hypotheses selected within a domain. These experiments are reported not only to provide a clearer understanding of debugging, but to extend Hypothesis Testing Theory of problem solving. Software debugging is a human
problem-solving process that involve generation and testing of hypotheses. It would be expected that results from previous research on HTT in other tasks would be replicated for the task of software debugging.

Experiment 1.

During the process of debugging, the case often arises where multiple hypotheses could be generated regarding a specific "bug." If each hypothesis were equally likely to provide the solution, the question of which hypothesis to test first naturally arises. The optimal answer would be to start with the easiest hypothesis to test first. This would be the hypothesis associated with the least cost according to some criterion, such as time or effort. The optimal sequence of testing the hypotheses would be to progress from the hypothesis with the least cost to test to the hypothesis with the highest cost to test. Experiment 1 was designed to see how well people are at discovering and using this optimal strategy. The task used in Experiment 1 was specifically designed so the costs involved in testing each hypothesis were clearly defined and the a priori probability of any hypothesis being correct was the same as any other.

Method.
Subjects and Equipment.

The subjects consisted of 16 college students. A Macintosh computer was used for display purposes with the mouse being used as a selection device.

Procedure.

The independent variable was problem number and the dependent variable was solution strategy. Each of the 16 subjects was given a series of 20 problems in which the subject was to determine which one out of five equations had a "bug" in it. The solution strategy a subject used for each problem was classified as one of the following three: an optimal strategy, a sequential strategy, or no discernable strategy.

The subjects were instructed as follows: The task was to find the single "incorrect" equation in each problem as fast as possible without sacrificing accuracy. The equations could be tested in any order subjects chose. An equation could be retested if desired. Subjects were also instructed that each equation was as likely as any other to be wrong. This was done so subjects would not automatically infer that the longer, more complex equation were more likely to be incorrect. An experimental session lasted approximately 35 minutes.

Figure 1 illustrates a typical screen for Experiment 1. As can be seen, each problem had a display with five equations with
each equation having a different number of terms. The order of the equations was random on any given problem. Next to each equation was a "test" button the subject could click (by moving the cursor on the screen with the mouse to within the outline of the "button" on the screen and pressing the mouse button) to bring up a "testing display as illustrated in Figure 2. This display had explicit values for each variable in the equation as well as the original equation. The subject would then either click in the "correct" button if the equation was correct with the values stated, otherwise in the "incorrect" button. If the "incorrect" button was clicked and the equation was in fact incorrect, the next problem was presented. Otherwise, the original display with the five equations was displayed so a subject could test another equation. A check-mark was placed next to previously tested equations.
Figure 1. Typical problem display used in experiment 1.

\begin{align*}
\text{test} & \quad Y = A + 6B + 6C \\
\text{test} & \quad Y = A + 9B \\
\text{test} & \quad Y = A + 5B + 6C + 4D + 4E + 8F \\
\text{test} & \quad Y = A + 8B + 2C + 5D \\
\text{test} & \quad Y = A + 5B + 7C + 2D + 9E
\end{align*}
Figure 2. Typical testing display used in experiment 1.
Results and Discussion.

The order in which a subject tested the equations on a problem was used to infer the strategy for selecting equations. The strategy was classified as either optimal, sequential, or not discernable. The optimal strategy for this task would be to start with the equation that contained the smallest number of terms and then progress to equations with a higher number of terms. This strategy minimizes the amount of time required on the average to find the "incorrect" equation, given the assumption that longer equations take longer time to solve. A sequential strategy was defined as a progression from the top equation to the bottom equation or vice versa. All other sequences of testing the equations were classified as no discernable strategy. Display as illustrated in Figure 2. This display had explicit values for each variable in the equation as well as the original equation. The subject would then either click in the "correct" button if the equation was correct with the values stated, otherwise in the "incorrect" button. If the "incorrect" button was clicked and the equation was in fact incorrect, the next problem was presented. Otherwise, the original display with the five equations was displayed so a subject could test another equation. A check-mark was placed next to previously tested equations.
Figure 3. Number of subjects following various strategies as a function of problem number for experiment 1.
Figure 3 shows the number of subjects following the optimal strategy, a sequential strategy, or no discernable strategy as a function of problem number. The scoring of a strategy as optimal was lenient. In two cases where a strategy could be classified as either sequential or optimal, it was scored as optimal. As can be seen, the number of subjects following the optimal strategy increased from 1/16 (6.25%) to 11/16 (68.75%), \( t (15) = 3.87, p < .01 \). The 95% confidence interval around the percentage using the optimal strategy on the last problem for the population ranges from 46% to 92%.

Surprisingly, five out of 16 subjects were not following the optimal strategy by problem 20. It was expected that close to 100% of the subjects would find the optimal strategy, given the very simplistic nature of the task. The number of terms for each equation was clearly visible. It would seem that the easier equations to test would be readily apparent.

One explanation is that prior experiences of the subjects might have lead them to believe that the longer equations were more likely to contain the bug. However, instructions to the subjects stressed the fact that each of the five equations had the same probability of containing the bug. It would also be expected that subjects who believed that the longer equations were more
likely to contain the bug would test the equations in order from the longest to the shortest. This pattern of behavior was not seen in the subjects' data. A subject would quickly learn that assignment of the bug to an equation was random during the course of the experiment as well.

Another explanation is that subjects tested the equations in sequential order for bookkeeping purposes. Subjects might have tested the equations from top to bottom or vice versa in order to simplify keeping track of which equations had been previously tested. However, as noted earlier, this bookkeeping was done for the subjects by placing a check mark next to equations that had been tested. It seems unlikely that a sequential testing order would provide any additional benefit over the check mark.

It should be pointed out that most of the subjects who eventually learned the optimal strategy started without a discernable strategy. Although the differences in difficulty among the various equations were readily apparent, it seems as though the subjects must actually experience the difficulty of an equation before difficulty becomes a salient criterion to select equations. Those subjects using a sequential strategy to find the bug on problem three were still using this strategy on problem 20. Such subjects seem to stay oblivious of the optimal strategy, and thus the difficulty of the equation never became a salient
criterion for selection. The overall trend is that those subjects with no discernable strategy in the first few problems end up finding the optimal strategy, while those that use a sequential strategy stay with it.

These results suggest that if the difficulty of the equations were made less salient by making the task demands less clearly defined, the optimal strategy would not be as obvious. In Experiment 2, the task was made slightly more complex by having the possibility of having more than one "bug" present. This also made the task more realistic because programmers seldom, if ever, know the number of remaining bugs.

Experiment 2.

Method.

Subjects and Equipment.

The subjects were 16 college age undergraduates. None of the subjects participated in the first experiment. The same equipment was used as in the first experiment.

Procedure.

The second experiment was like the first except there was the possibility that more than one equation was "incorrect" (the subjects in this experiment received the same instructions as in Experiment 1 with this exception). As soon as the last "bug" was
found, the next problem was presented automatically. The definition of the optimal strategy remained the same, and for the same reasons given earlier. The experiment took approximately 45 minutes to perform.

**Results and Discussion.**

The method of classifying a strategy for a problem was the same as in experiment 1. Figure 4 show the number of subjects following the optimal strategy, a sequential strategy, or no discernable strategy as a function of problem number. The number of subjects following the optimal strategy on the first problem as well as the last problem remained constant at 1/9 (11.1%) and was in fact the same subject.

The 95% confidence interval around the percentage using the optimal strategy on the first and last problem ranges from 0% to 31%. The number of people following the optimal strategy on the last problem for this experiment was significantly lower than for Experiment 1, \( \chi^2 = 7.65, \ p < .01 \).
Although the task of Experiment 1 was made only slightly more complex, a dramatically smaller percentage of people were following the optimal strategy on the last problem. A closer look at Figure 4 show that subjects tended to stay with the same strategy throughout the experiment. As in experiment 1, those subjects
Figure 4. Number of subjects following various strategies as a function of problem number for experiment 2.
using a sequential strategy on problem three used the strategy for the rest of the problems. The plots of those subjects using a sequential strategy also showed the least amount of perturbation from both Experiment 1 and Experiment 2.

The main reason for the difference in results from Experiment 1 to Experiment 2 is that those subjects in Experiment 1 following no discernable strategy on the first few problems eventually learned the optimal strategy. Those subjects in Experiment 2 that were following no discernable strategy on the first few problems never learned the optimal strategy.

This lack of learning by having only the possibility of more than one "bug" present is puzzling. The only explanation that can be offered at this time is that the probabilistic reasoning required to ascertain the optimal strategy is slightly simpler. These results imply that if there were an optimal strategy on a "real world" debugging task, it would not be discovered.

Experiment 3.

The debugging task for experiment 3 is somewhat more realistic than the tasks used in the first two experiments. The equations were arranged to resemble an "electronic spreadsheet". The equations used could be split into two domains based on complexity. The order that solutions appeared in these two
domains was manipulated to test several assumptions of Hypothesis Testing Theory. Subjects were given a series of training problems in which the solution was in a consistent domain. A switch problem was then given in which the solution was in the other domain.

The transfer assumption would predict that subjects would initially sample from the domain that the solution was found in for previous problems. The empty-set assumption would predict that subjects would exhaust this domain before switching to the other domain to test.

Results from a study by Lane, McDaniel, Bleichfeld, and Rabinowitz (1976) also indicate that switching domains is easier when going from an simple domain to a complex domain than vice versa. In this study, subjects were given six training problems before switching solution domains for the last two problems. Subjects in the "simple-complex" switch were more likely to switch domains on the first switch problem and continue sampling from this new domain on the next problem. Those subjects in the "complex-simple" were less likely to switch domains on the first switch problem and sampled from both domains equally often on the next problem.

Method.
Subjects and Materials.

The subjects were five college-age undergraduates. None of the subjects participated in the first two experiments. The same equipment was used as in the first two experiments.

Procedure.

The independent variable was solution-domain order having the two levels easy-hard, or hard-easy. The dependent variable was domain testing sequence.

Figure 5 illustrates a typical screen from Experiment 3. In the bottom display, nine equations were displayed in a 3x3 grid. Five of the nine equations were "easy", while the remaining four were "hard." An example of an easy cell is "The value of this cell is 4" and an example of a hard cell is "The value of this cell is the mean of 4,6 plus the sum of 2,3,4." A cell could be "tested" by clicking inside the cell's boundaries. The actual value of the cell would then appear in the top display of Figure 3. The subject would then either click on the "correct" button if the value matched the equation, otherwise in the "incorrect" button. If a subject clicked the "incorrect" button and the value was in fact incorrect, the next problem was presented.
<table>
<thead>
<tr>
<th>Cell 1</th>
<th>Cell 2</th>
<th>Cell 3</th>
<th>Cell 4</th>
<th>Cell 5</th>
<th>Cell 6</th>
<th>Cell 7</th>
<th>Cell 8</th>
<th>Cell 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>The value of this cell is the mean of 4, 1, 6, 5, plus the sum of 8, 3, 6, 8.</td>
<td>The value of this cell is the mean of 2, 4, 1, plus the sum of 3, 5, 8, 3.</td>
<td>The value of this cell is 7.</td>
<td>The value of this cell is 4.</td>
<td>The value of this cell is 4.</td>
<td>The value of this cell is 6.</td>
<td>The value of this cell is the mean of 6, 5, 4, 1, 7, plus the sum of 5, 5, 6, 6.</td>
<td>The value of this cell is the mean of 6, 5, 4, 1, 7, plus the sum of 6, 4, 3, 2.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5. Typical problem display used in experiment 3.
Again the subjects' tasks was to find the single "incorrect" cell and were given appropriate instructions. However, no mention of the relative probabilities of a cell having the bug was made.

Two subjects had seven problems where the bug was in an "easy" cell and then a problem where the bug was in a "hard" cell, while three subjects had seven "hard" problems and then an "easy" problem. Each problem took approximately two minutes to find the "bug".

Results and Discussion.

All subjects initially sampled the cell type congruent with the solution of the previous seven problems on the last problem. These cells were sampled until they were exhausted. Both subjects in the "easy-hard" condition found the bug in the last problem without retesting any "easy" cell, while two of the three subjects in the "hard-easy" condition retested "hard" cells before finding the incorrect "easy" cell. Of these two subjects exhibiting resampling, one subject retested a "hard" cell once, while the other retested "hard" cells seven times before testing an easy cell.

This pattern of results was as predicted by HTT. Specifically the transfer assumption as well as the empty-set assumption was
supported. However, what is surprising is that resampling from the initial domain occurred in the "hard-easy" condition despite the fact that the "hard" domain consisted of only four items. This finding of the "easy-hard" switch being easier for the subject than the "hard-easy" switch agrees with results of the study by Lane et al. (1976).

One explanation of the asymmetry in the tendency to switch domains for the problems used in Experiment 3 is that subjects are more confident of correct verification for the easy problems. Subject might be less confident of correctly verifying a "hard" problem and resample it. However, even the "hard" problems used were relatively easy. In fact, outside the context of the switch problem, only three of 21 problems presented where the solution was in the "hard" domain contained resampling. Two of the three problems were the initial problem given. Even an "easy" problem had resampling on the initial problem. So performance on the "hard" problem was actually quite good and should not have affected the subjects' confidence.

Experiment 4.

Experiment 4 was designed to test if people would exhibit behavior predicted by the assumptions of HTT on a more realistic debugging task than in Experiment 3. More resemblance to an
"electronic spreadsheet" was incorporated into the task. Additionally, the effect of programmer experience on hypothesis selection strategies was also examined.

**Method.**

**Subjects and Materials.**

The subjects were 15 college age undergraduates. None of the subjects participated in the first three experiments. Nine of the subjects had no programming experience and were considered "novices", while the remaining six had extensive programming experience (graduate computer science students, full-time programming jobs, etc.) and were considered "experienced". The same equipment was used as in the first three experiments.

**Procedure.**

The two independent variables were solution domain order and programmer experience. In this experiment, solution domain order had three levels: easy-hard, hard-easy, and alternating. The two levels of programmer experience were "novice" and "experienced". As in experiment 3, the dependent variable was domain testing sequence.

Figure 6 illustrates a typical screen from Experiment 3. In the bottom display, 25 equations described in English were displayed in a 5x5 grid. Six of the 25 cells were "easy", while
another six were "hard". The English description consisted of a label for that cell optionally followed by a description of the formula for that cell (for example "Widget Selling Price" or "Jan sales were units shipped * Widget Selling Price"). The cells were designed such that a "hard" cell took approximately four to six times as long to verify as an "easy" cell. The remaining cells were of average difficulty but never contained the "bug". A cell could be "tested" by clicking inside the cell's boundaries. The actual formula of the cell would then appear in the top display of Figure 6. The actual formulas refer to other cell by cell coordinates (for example "A1"). The incorrect cell did not have the formula match the English description. The only reason they would not match would be if a cell reference used in the formula did not match a label used in the description. Assignment of the formulas to a spatial location was random.

Again the subjects' task was to find the single "incorrect" cell after they were given appropriate instructions. Subjects saw six training problems before the switch problem. Each problem took approximately four minutes.
<table>
<thead>
<tr>
<th></th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Feb. profits * Feb. sales - Feb. costs</td>
</tr>
<tr>
<td>2</td>
<td>March costs were March sales - misc costs</td>
</tr>
<tr>
<td>3</td>
<td>April profits were April sales - April costs</td>
</tr>
<tr>
<td>4</td>
<td>May profits were May sales - widget selling price</td>
</tr>
<tr>
<td>5</td>
<td>June profits were June sales - widget selling price</td>
</tr>
</tbody>
</table>

**Figure 6. Typical problem display for experiment 4.**
Results and Discussion.

The solution domain-order manipulation did not have an effect. All subjects initially sampled the cells in sequential order until the "bug" was found on the switch problem. It appears that subjects did not perceive the domains in which the equations were grouped. There was no difference between the novices and the experienced programmers.

Levine used training problems in order to be sure that subjects were sampling from the appropriate domain when the switch problem was presented. This was insured by having a solution that could be found by only sampling from the correct domain. Since the training problems did not induce the subjects to sample from a specific domain, the assumptions of hypothesis testing theory were not adequately tested.

One explanation is that although the "hard" cells took longer to verify than the "easy" cells, this difference in time was not as readily apparent upon brief examination as in experiment three. Subjects must learn what English descriptions correspond to the "easy" and the "hard" cells. Thus on the first problem, it would not be expected for the subjects to know the difference between the two. By the time they learn the difference, the subjects were already using a sequential strategy and were reluctant to switch.
The use of a sequential strategy was also noted in experiments 1 and 2 and appears to have a strong intuitive appeal for subjects. This is despite the fact that sampling "easy" problems first would minimize time in the "easy-hard" and the "alternating" conditions. It must be noted that subjects would have little trouble rating which cells were harder, but they simply failed to use this information.

One advantage that the sequential strategy does have is that bookkeeping of already tested cells is minimized. The larger grid size of experiment 4 as compared to experiment 3 might account for the subjects universally using a sequential search strategy.

General Discussion

The results of this series of experiments show that people are poor performers at selecting an appropriate hypothesis to test. Although subjects tended to discover and use the optimal search strategy on the very simple task used in Experiment 1, they did not discover the optimal search strategy on the only slightly more difficult task used in Experiment 2. Subjects that started off using sequential strategies in Experiments 1 and 2 were likely to continue using this strategy for the rest of the experiment. Those subjects that used no discernable strategy in Experiment 1 tended to learn the optimal strategy by the last problem. While those
subjects in Experiment 2 that used no discernable strategy tended not to discover and use the optimal strategy.

The pattern of results from Experiment 3 was as predicted by HTT. The transfer and empty-set assumptions were supported. Subjects continued to sample from the same domain on the switch problem as the domain that contained the previous solution. They sample from this domain until the domain was exhausted. The switch from the "easy" domain to the "hard" was easier than the switch from "hard" to "easy". Two of the three subjects in the "hard-easy" switch resampled already tested hypothesis while none of the subjects in the "easy-hard" switch did.

As noted earlier, if the probabilities between the hypothesis are the same, checking the easy hypothesis first will minimize the total cost to find the "bug." Even if the probabilities are skewed in favor of the complex hypotheses, the costs of testing the simple hypothesis are usually trivial when compared to the costs of falling into the einstellung trap and continuing to sample and/or resample from the wrong domain. This lends further support to the argument that simple hypotheses should be tested first.

In experiment 4, the training problems failed to induce the subjects to sample from a specific domain. The subjects did not perceive the solutions as belonging to one "domain" of solution or
another. This might have been due to the fact that the relative
difference in difficulty between the two groups of cells was not
as striking as in Experiment 3. Subjects treated the 5x5 grid of
cells as a single domain and sampled sequentially from that
domain.

The use of a sequential strategy by the subjects as a search
strategy seems particularly strong. Those subjects that started
using a sequential search strategy in Experiments 1 and 2 showed
the least tendency to learn the optimal strategy. In Experiment 4
subjects continued to use a sequential strategy despite
experimental manipulation to induce the use of other strategies,
that of sampling from specific domains.

The lack of a difference between novices and experts in
Experiment 4 is consistent with the previous research. Evidently,
experts are just as likely to use the sequential strategy to the
exclusion of others as novices. Three of the ten subjects in
Gould's study used a sequential strategy to find the bug. In fact,
the code comprehension phase mentioned in all of the studies cited
earlier were described as a sequential scanning of the program
text. Since only explicit changes in a program were considered as
hypothesis tests, it is not known if subjects were actually
implicitly testing hypothesis sequentially in this phase.
It could be argued that this is understandable behavior when debugging conventional programming languages, such as Pascal, that generally have linear flow of control. This was not true of the experimental tasks used in the experiments described in this paper. New programming languages proposed for parallel processing generally do not have this convenient linear flow of control. Some conventional computer programming tasks such as electronic spreadsheet programming, multitasking, and real-time programming also do not have linear flow of control. For such tasks, the tendency to use sequential search strategies might be non-optimal.

The tasks used in the experiments described in this paper have a common attribute. The sequential strategy, while non-optimal, will produce the correct solution. This might not be so on the programming examples given above. The tendency to use a sequential strategy and the reluctance to switch from it would then produce a phenomenon similar to einstellung. This is an area that needs further exploration.

The pattern of behavior that emerges from the results of these four experiments is as follows. If people perceive the set of possible solutions as one "domain" they sequentially sample from this domain until the solution is found. Only if the relative difficulties of each sample is striking will they use an optimal
strategy by selecting the easy samples first. If the set of possible solutions is perceived as having multiple domains, then the assumptions of hypothesis testing theory are supported.

The costs associated with debugging during program development and maintenance were stated earlier. Despite these costs, debugging is generally not taught as a topic in computer science courses. It is assumed that programmers learn how to debug efficiently on their own. The results of this series of experiments suggest that that this does not happen. The tasks used in this series of experiments were intentionally made as simple as possible. Subjects still had difficulty learning an optimal selection strategy. A brute-force sequential search was a common strategy.

The effects of einstellung can be devastating. A tremendous waste of time and effort can happen due to resampling hypotheses or sampling from the wrong domain. Recognition of this phenomenon, as evidenced by Myers (1979), can help lessen the negative impact of it. The simple rule of sampling simple domains before more complex domains can help prevent einstellung as well. This rule should be followed even if the error probabilities are skewed to the more complex domain. As noted earlier, the costs of testing a simple domain is usually trivial when compared the the cost of einstellung. The topic of
debugging should probably receive more attention in computer science courses.

It is hoped that this line of research shows that debugging is not the "art" that many people believe it is. Given a suitable framework in which to study debugging, such as hypothesis testing theory, research such as this can provide valuable results to aid in the debugging process.
References


