

PROJECT ADMINISTRATION DATA SHEET



ORIGINAL



REVISION NO. _____

Project No. E-24-616GTRI~~OT~~DATE 2/2/83Project Director: Dr. John M. HammerSchool/Inst ISyESponsor: National Science FoundationType Agreement: Grant No. IST-8217440Award Period: From 1/15/83 To 6/30/85 * (Performance) 9/30/85 (Reports)Sponsor Amount: Total Estimated: \$74,506 6/30/86 Funded: \$ _____

Cost Sharing Amount: \$ _____ Cost Sharing No: _____

Title: Models of Human Performance Using Text Editors (Information Science)

ADMINISTRATIVE DATA

OCA Contact Frank Huff x4820

1) Sponsor Technical Contact/Program Officer

2) Sponsor Admin/Contractual Matters/Grants Official

H. E. BamfordSharon GrahamDivision of Program SectionDivision of Grants & ContractsNSFDirectorate for AdministrationWashington, DC 20550NSF(202) 357-9555Washington, DC 20550(202) 357-9843Defense Priority Rating: N/AMilitary Security Classification: N/A

(or) Company/Industrial Proprietary: _____

RESTRICTIONS

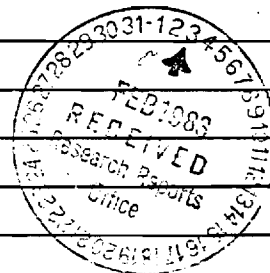
Fee Attached NSF Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval - Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of \$500 or 125% of approved proposal budget category.

Equipment: Title vests with GIT

COMMENTS:

* Includes a 6 month unfunded flexibility period.



COPIES TO:

Research Administrative Network
Research Property Management
Accounting
Procurement/EES Supply ServicesResearch Security Services
Reports Coordinator (OCA)
GTRI
LibraryResearch Communications (2)
Project File
Other Hammer
Other _____

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEETDate 3/14/88Project No. E-24-616 School/~~XXX~~ ISvEIncludes Subproject No.(s) N/AProject Director(s) John M. Hammer GTRC/GITSponsor NSFTitle Models of Human Performance Using Text Editors (Information Science)Effective Completion Date: 6/30/86 (Performance) 9/30/86 (Reports)

Grant/Contract Closeout Actions Remaining:

- ☒ None
- ☐ Final Invoice or Copy of Last Invoice Serving as Final
- ☐ Release and Assignment
- ☐ Final Report of Inventions and/or Subcontract:
Patent and Subcontract Questionnaire
sent to Project Director ☐
- ☐ Govt. Property Inventory & Related Certificate
- ☐ Classified Material Certificate
- ☐ Other _____

Continues Project No. _____ Continued by Project No. _____

COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/GTRI Supply Services
Research Security Services
Reports Coordinator (OCA)
Program Administration Division
Contract Support Division

Facilities Management - ERB
Library
GTRC
Project File
Other _____

A display editor with random access and continuous control

JOHN M. HAMMER

*Center for Man-Machine Systems Research, Georgia Institute of Technology,
Atlanta, Georgia 30332, U.S.A.*

(Received 3 June 1983, and in revised form 12 October 1983)

An analysis of human information-processing during editor positioning led to a text editor with two significant features: continuous control and random access to text. Continuous control is a feature that allows the user to control the editor while it executes a positioning command. It will be shown that such a style of interaction eliminates difficult design decisions and leads to new methods of positioning an editor which are also less sensitive to human error. Random access to the text file means that the editor can be positioned to any point in the file in a constant time. The advantage of random access is that it is noticeably faster than the sequential access used by most editors. The implementation of continuous control and random access is discussed.

Introduction

An editor that is more quickly positioned by users is described. Two features are responsible. The first is continuous control, where the user can control the editor while it executes a positioning command. The second feature is random access to text in which the editor can be positioned in constant time to any page in the file.

This article contains five parts. The first briefly describes the environment in which the editor was used. The second part describes the editing task for which this editor was designed. Positioning an editor and our view of human information-processing during editing are described. The third part reviews previous literature with special emphasis on editors designed with a particular view to humans. The fourth and fifth parts describe continuous control and random access to the text.

The editor ran on a DECsystem-10 using advanced CRT displays capable of cursor positioning and insert/delete line operations. The most common transmission speeds were 2400 and 9600 baud. The DECsystem-10 is a 36-bit wide, medium size mainframe used primarily for timesharing. It is found primarily in universities and research centers.

The roughly two dozen users of the editor were primarily computer engineers and artificial intelligence researchers who made sophisticated use of the computer. Several secretaries, who edited for several hours a day, were also users. The files edited were typically 1000 or more lines. Many of the users became interested in the editor through other users. The view of editing and the human factors view of editing were not, however, a result of observing these users. Instead, the author simply attempted to design a better editor for personal use.

TERMINOLOGY

The phrase *positioning an editor* will be used (for economy of space) for moving the editor's internal screen cursor from one point to another point in the text file.

The term *module* will be used to refer to procedures and subroutines in programs and to chapters, sections, etc., in text files.

The editing task

The user was assumed to be a programmer familiar with the program or document being edited. Positioning, the only aspect of editing to be discussed here, was assumed to be either local—such as changes within the same module—or global, which could be to another module located anywhere in the file. Further, the user was assumed to desire locality in the effects of most editor commands. Thus, most commands could affect only the current module. The editor was designed for files that consisted of a number of modules. It was not designed to edit data files or databases, although it could be used for this task.

HUMAN INFORMATION-PROCESSING

The editor was designed to facilitate human information-processing during editor positioning. Many kinds of human information-processing abilities are used during text editing—visual perception, planning, memory retrieval, motor control, etc. In this section the abilities that are assumed to be used during editor positioning are presented. Although plausible, their existence is based only on informal observation, not controlled experiments. Furthermore, these abilities are assumed to be important factors in human performance while editing.

The first assumption is that the human must estimate the distance between two points in a file (i.e. the current location and the desired location) except when the distance is very small. Rather than requiring estimation, the editor displayed text continuously to the user. When the desired line was under the screen cursor, the user stopped the editor.

The second assumption is that a human may know the text around the desired location (thus, editor commands that search for text), but that the text between the current and desired location will not be considered when the search key is formulated. Thus, the search key might be longer or shorter than needed (choosing an optimal key requires examination of all text between the current and desired location). As will be seen later, the editor is designed to accommodate the tendency for too short and too long keys.

The third assumption is that humans make errors while positioning an editor. A natural and sometimes-used accommodation is a command to undo the effects of the previous command. While such a command was available in this editor, the approach taken was to minimize the negative impact of errors.

The fourth assumption deals with human memory-retrieval during global editor positioning. The human is assumed to be able to retrieve easily the name of the module to which the editor is to move. Thus, in moving long distances, modules are more convenient mnemonics than relative or absolute page and line numbers.

DESIGN PHILOSOPHY

The assumptions about human information-processing lead to two design philosophies. The first, which is novel, is that the user remains in control while the editor is executing a positioning command. This means that the user may make certain modifications to

a command during its execution or stop the command prematurely to execute another command. To remain in control, the user must receive continuous feedback on what the editor is doing. Yet, under certain situations, the user must be able to limit the amount of feedback because the intermediate editor actions are not needed. Finally, the editor must be kept close to the user. The editor cannot be controlled if it has a dozen lines in the output buffer to the terminal, for any action the user types will take effect a dozen lines ahead of what the terminal is displaying.

The second design philosophy is that the text file be viewed as a collection of modules. Local editor positioning was assumed to remain within one module, which was assumed to be stored on one page of text. Thus, most positioning commands would not move the editor away from the current page. The contents of a page, no matter how large, were stored entirely in main memory. Access to other pages was primarily through a global search and by page name. Associated with every page was a list of zero or more names that were used to retrieve that page. All pages were stored in a random access disk file for fast access that was independent of the editor's current position. While random access pages accessed by name is not novel (Samuel, 1977), it has not been described in the archival literature.

Literature

Surveys describing editing and major editors are Meyrowitz & van Dam (1982a, b), respectively. A review of relatively recent research on the human-computer interface aspects of editing is Embley & Nagy (1981). The implementation of display editors is discussed in Finseth (1980).

The remainder of this section will discuss editors that were designed for a specific view to human information-processing. Also explored will be the implications of a view for the editor.

The Xerox Star (Smith *et al.*, 1982) was designed to have a concrete and simple interface. It is concrete because all entities—objects and actions—are represented by screen icons. Either can be selected by a mouse, a pointing device. Finally, the screen icons display all available aspects of the Star. There are no hidden mechanisms.

The simplicity of the Star interface is in its command interface. The same universal command set is consistently used to manipulate all entities—text, file, messages, icons themselves, etc. The interface is also intended to be modeless. Thus, any action can be taken in any situation.

EMACS (Stallman, 1981) was designed to be extensible and self-documenting. While no models of human information-processing are explicitly stated, there are some implicit assumptions. First, the users are assumed to be computer scientists. Second, these experts will need to customize the editor for a variety of tasks. Finally, the editor design is best done by these experts instead of by a designer who cannot anticipate all the needs of and improvements by the experts.

The consequence of these assumptions is that EMACS contains two parts: a display-text management package plus an editor programming language. Expert users can and do modify the editor program. Stallman considers distributed editor modification to be a success. Many of the features of the default EMACS editor program were developed by users.

Ed (Kernighan & Plauger, 1981) is the editor distributed with UNIX. Ed, like UNIX, was designed to be concise and powerful. It achieves these ends by a clever combination of a small set of primitives. Thus, it is held to be human-engineered (for computer scientists). Unfortunately, its terse straightforward design has recognized problems (Norman, 1981). Because Ed is intended to receive commands from another process through a one-directional pipe (as well as from a keyboard) it provides virtually no feedback. Consequently, a user has difficulty determining the editor's mode. Second, Ed's straightforward design will cause it to do exactly what the user requests, whether dangerous or not.

The UNIX developers are correct in stating that a simple interface should be a contribution to good human-engineering. However, Norman is correct in showing that the simplicity is not apparent to the casual user, and that the simple absence of feedback is a drawback even for experts.

Continuous control

Continuous control is described in two sections. The first describes the implementation of various control features. From understanding the implementation one can gain some idea of the editor's capabilities. The second section describes the advantages that arise from these and other capabilities. In particular, we will explain how the editor supports the previously described forms of human information-processing.

EXECUTION OF A COMMAND

Virtually every positioning command was executed in a central routine which is described as follows. The routine had four arguments.

1. UNITS—the size (character, word, line, or page) and the number of units to be crossed before stopping the positioning.
2. KEY—an (optional) text string to search for.
3. SCROLL—a boolean that determined if the screen cursor was maintained on the same line as the internal cursor. If true, the user saw where the editor was while it moved. If false, the editor moved—without changing the display—to the new, final location and then updated the display.
4. QUERY—a boolean that determined if the keyboard was queried during execution of a command.

The code was as follows:

```
repeat
    move 1 UNIT;
    optionally search for KEY;
    if    SCROLL
    then  SCROLL_DISPLAY;
    if    QUERY
    then  QUERY_KEYBOARD;
until   moved over requested number of UNITS
or      found KEY
or      command issued
or      moved to end of text;
```

```
if      not SCROLL
then    FIX_SCREEN;
```

In a commonly used local search command, a search KEY would be specified, the UNITS specified as an infinite number of lines contained on this current page, and SCROLL and QUERY would be true. In executing this command, the routine would move down one line and check for occurrence of the search key. If found, a flag would be set to terminate the loop. Otherwise, the screen would be scrolled (if necessary) to keep the current line displayed on the screen, and the keyboard queries. This process continued until finished (e.g. key found in text or no more text lines) or until the user typed something at the keyboard.

The actions taken by SCROLL_SCREEN, QUERY_KEYBOARD, and FIX_SCREEN will be discussed next. The other code sections are fairly typical of an editor and will not be discussed.

SCROLL_SCREEN kept the user informed of the editor's current position. If the editor was positioned to a line that preceded or followed the top line or bottom line on the screen, respectively, then the screen was scrolled. The routine then always positioned the screen cursor to the editor's current internal cursor.

QUERY_KEYBOARD could take several actions. Already mentioned have been stopping execution of a command and executing another command. A space character simply stopped execution and was itself discarded. The space was chosen because it should be fast; the terminal space bar is large and directly under the thumbs at almost all times. A control character (all commands began with control characters) also stopped execution; when control returned from this routine to the top level, this control character was read as a command, which would most likely result in another call to the central routine.

QUERY_KEYBOARD also allowed two modifications during the execution of the current command. The first modification was to change SCROLL to false, which caused the display to cease being scrolled. This modification is useful when the user decides not to watch what text the editor passes over. For example, the local search command (described above) displays all the text it passes over. The user can, however, shut off this feedback for the duration of one command. Control-O was chosen to be this command since this key serves the same purpose outside the editor.

The second modification (under QUERY_KEYBOARD) was to control the length of the output buffer. Its length controlled how far ahead the editor was of the screen. If it was too large, the editor could easily be 10 or 20 lines ahead of the display, with these same lines being in the output buffer waiting to be transmitted to the terminal. If the length was too small, the terminal could not be driven at its rated speed due to timesharing. The buffer length could be varied dynamically by the user to respond to differences in the system load and the baud rate. The digits 0-9 were used for this function.

FIX_SCREEN's purpose was to insure that the line at which the editor was internally positioned was also displayed on the screen. It would be called if the screen were known to need adjustment to display the current line. It recognized three cases. First, if the current line was on the screen, it did nothing. Second, if the current line was just off the top or the bottom of the screen, it was scrolled. Finally, if the current line was far off the screen, the best that could be done was to erase the screen and display the current line and its neighbors.

This redisplay began by first displaying the current line (presumably, the most important) in the center of the screen. Following that, lines above and below the current line were alternatively added until the screen was full. In cases where the current line was the first or last line of a page, the redisplay started at the top or bottom of the screen, respectively, and filled toward the other boundary (thus, providing immediate feedback about being at either end of the page). This redisplay process also included calls to QUERY_KEYBOARD thus allowing: (1) the redisplay output to be stopped to execute another command; (2) the output to be stopped; and (3) the buffer length to be controlled.

Although the lines are displayed in an unusual and what may seem to be a distracting order, it is consistent, even desirable for continuous control. Showing the current editor position first is showing the most important line first. Thus, the order lines are displayed is better for continuous control.

ADVANTAGES OF CONTINUOUS CONTROL

The advantages of continuous control are the elimination of certain difficult design decisions, better support for some methods that humans use to position editors, tolerance for human error, and synergism of editing features.

The first advantage is to eliminate certain static design decisions that are better made dynamically by the human while editing. An example of this is scanning for a particular line of text by scrolling the display. Most editors have commands to move the editor position forward or backward N lines. This command is often used when scanning for text. N often can be omitted, in which case it defaults to some value, say 16. Of course, $N = 16$ is practically always suboptimal (unless exactly 16 lines were required). Further, typing a value for N is extra effort.

With continuous control, the user need only indicate the direction the editor is to move. The editor then continuously scrolls the display in that direction. When the desired line appears at the cursor, it can be stopped by command. This mode of interaction better supports the user's scanning than do traditional editors. Of course, the user is unlikely to be able to stop the editor exactly on the desired location. **Commands to move one line can be used to achieve final positioning.**

The second advantage is the feedback provided by commands, especially search commands, as they execute. For example, the local search command (stays on the current page) displays every line it crosses over as it looks for the key. The search can be stopped if something of interest is noticed or the command is found to be in error. Displaying the intervening text could be a disadvantage if the feedback level itself were not controlled. Fortunately, it is controlled. Finally, the control of command and feedback is dynamic—it is not chosen when the command is initially entered but rather as results of the command are seen. Certainly, it is better to take action after partial results are displayed rather than try to predict those results before the command is issued.

A third advantage lies in a pair of commands that repeat the previous search and return the editor to its previous position. The command to repeat the previous search is most useful when the original search key was either intentionally or unintentionally made too short. This one keystroke command (which for speed was keyed by striking the control key and space bar) is often a fast way to reach the desired location with a short string. In fact, this command is able to keep the editor continuously

moving towards the desired location, and thus is used to control the editor continuously.

In fact, the repeat search key is often used too quickly, causing the editor to move past the desired position. To its aid comes the one keystroke command which returns the editor to the previous position. Since reverting is so easy, the user is allowed to be sloppy in using the repeat search command. Thus, these two commands demonstrate a tolerance for human error.

The design has other aspects that make it tolerant of human error. If a search string is chosen erroneously (nothing in the file matches), the user will receive feedback about the large amount of text the editor is crossing over. The volume of this feedback may tip off the user that the search is not working, and it may be stopped. If the user visually scans text in the wrong direction, the editor can be turned around very quickly.

Editor feature synergy is demonstrated by `FIX_SCREEN`, the repeat search, and the revert position commands. Suppose that a global search is issued for each instance of a routine call and that repeat search will be used to find successive instances. Each instance will be displayed by erasing the screen and filling from the center outwards. This output can be terminated by a new command as soon as the user determines that this instance is not the one required. Repeat search and revert position will complement each other as previously described. Each of these features is individually powerful, but they become more powerful when used in conjunction. To see this, consider the absence of each feature one at a time. If the display output could not be stopped early by command, a single keystroke repeat search command is much less effective, for the user must still wait for the output to finish before the next instance is displayed. If a search can be repeated only by issuing a new command of two or more keystrokes, the user is unlikely to be able to get the command off before the screen is filled anyway. Finally, if the revert position command was unavailable, the user would (as explained earlier) have to be more careful when issuing repeat search commands.

INCREMENTAL SEARCH

Incremental search is a form of search in which entering one additional character of the search key causes the editor to position immediately to the next instance of the just-lengthened key. Ordinary searches, in contrast, wait until the entire key is specified before any positioning is done. Incremental searching was, to the best of the author's knowledge, first implemented in EMACS (Stallman, 1981). Its aims are consistent with allowing the user to control the editor during positioning by giving feedback.

While other editor features described in this article were successful, incremental search did not operate as expected. It appears that updating the screen after each character had a negative effect on human attention, for the eye was drawn to the screen. Perhaps other methods of screen updating that limit the amount of feedback (a window of three lines or a bit-mapped display) might make this positioning method more usable.

RELATED FEATURES IN OTHER EDITORS

Continuous control may appear to be similar to command canceling and incremental redisplay, both of which have been implemented in other editors. This section shows the difference between them.

Meyrowitz & van Dam (1982a, Section 3.4.1.3) discuss command canceling, or stopping a command that has gone awry. Canceling is often implemented with a

software interrupt that cleans up and jumps to the command level. In contrast, a continuous control editor interacts with the user during command execution. This interaction is motivated by our own view of human information-processing. For example, feedback is necessary for control; our editor provides it and even allows control of feedback itself. Some of the commands of a continuous control editor can be designed so that they must be stopped.

Incremental redisplay delays updating the screen while the user continues to enter commands. By delaying feedback, it reduces transmission bandwidth if a number of changes are made to some text. It is an excellent idea if the terminal is not served by a high transmission speed. Because it is based on delayed feedback, it takes a somewhat different approach from continuous control, which provides full feedback.

Random access text

The editor, as stated earlier, imposed a structure on the file. It was to be a sequence of pages, where each page was assumed to contain one procedure or several related procedures. Pages were stored in a random access file. Although this practice is not new (Samuel, 1977), it has not been described in the archival literature. It is described here.

Because the user was assumed to want most commands to have only local effect, most positioning commands would not leave the current page. The farthest a scroll or local search would move was the first or last line on a page. All local commands scrolled the text so that the user could see what was happening (of course, the user could discontinue this output). If the file was split into pages as assumed, the amount of scrolling output (feedback) would be reasonable. Of course, some commands did cross page boundaries, and they did not scroll the screen. The global search did, however, display the first line of every page it entered, thus indicating progress (the computer is actually serving the user) and feedback on the distance being covered. To the same goal, the routine that read in pages also simultaneously loaded the screen. Thus, the first lines of a page were displayed before the last lines were entirely read in. This practice violates modular program design, which would have separated page access from display, and would have the page entirely read in before the first line was displayed. The advantages to the user of an immediate display outweighed the problems of increased program complexity.

Random access to text pages offers two advantages. First, random access gives noticeably faster response to retrieval of text that is far from the current location. For example, the SOS editor on the VAX 11/780 requires roughly 0.07 s/disk block (real time) to move to another point in a file. Of this time, 0.03 s/block is due to the seek and transfer rate of the disk alone. For files of 100 blocks, these times become quite noticeable.[†]

The second advantage of random access is having labels or tags which point to certain locations in the file. The tag allows positioning by name rather than by page and line number. It is also superior to global search because of speed and the uniqueness of tags. Tags can be implemented at very little additional cost since the variable-sized pages must be indexed.

[†] Real time response was measured on a lightly loaded system. The disk transfer and seek rates are 2 μ s/byte and 38 ms average, respectively. The SOS buffer size is 10,000 bytes; a block is 512 bytes.

The only disadvantages to random access text are (1) a rare delay for expanding the file to accommodate a just-enlarged page and (2) the wasted space due to blocking, which leaves empty space for expansion at the end of each record. These two factors trade off against each other. Allocation in smaller units reduces waste but increases the frequency of expansion. For reasonable allocation sizes, random access text will on the average far outperform sequential access.

Conclusion

An editor with random access to text and user continuous control over positioning has been described. Its design was based on assumptions about human information-processing during text editing. The editor's special features make it faster, less error sensitive, and more natural for editor tasks.

This preparation of this article was supported by the National Science Foundation under Grant No. IST-79-1647 and Grant No. IST-82-17440.

References

- EMBLEY, D. W. & NAGY, G. (1981). Behavioral aspects of text editors. *ACM Computing Surveys*, **13**(1), 33-70.
- FINSETH, C. A. (1980). A theory and practice of text editors. *Technical Memo 165*, Laboratory for Computer Science, M.I.T., Cambridge, Massachusetts.
- KERNIGHAN, B. W. & PLAUGER, P. J. (1981). *Software Tools in Pascal*. Reading, Massachusetts: Addison-Wesley.
- MEYROWITZ, N. & VAN DAM, A. (1982a). Interactive editing systems: part 1. *ACM Computing Surveys*, **14**(3), 321-352.
- MEYROWITZ, N. & VAN DAM, A. (1982b). Interactive editing systems: part 2. *ACM Computing Surveys*, **14**(3), 353-415.
- NORMAN, D. A. (1981). The trouble with Unix. *Datamation*, **27**(12), 139-150.
- SAMUEL, A. (1977). E. *Stanford Artificial Intelligence Laboratory Memo*.
- SMITH, D. C., IRBY, C., KIMBALL, R., VERPLANK, B. & HARSLEM, E. (1982). Designing the Star Interface. *BYTE*, **7**(4), 242-282.
- STALLMAN, R. M. (1981). EMACS: the extensible customizable, self-documenting display editor. *Proceedings. ACM SIGPLAN/SIGOA Conference on Text Manipulation*, Portland, Oregon, pp. 147-156.

Appendix: Implementation of the index†

This index implemented for random access was required, first of all, to describe pages so that they might be randomly accessed. Second, the index was required to map names—text strings—onto pages. Third, the entire file was to be readable by compilers without a conversion step to and from a special editor format. Thus, the index, stored at the front of the file it described, was to be interpreted as a comment by compilers. Also, pages were required to be physically stored in the logical order they appeared to the user.

A BNF description of the index page is (for an Algol program):

$\langle \text{index page} \rangle$	$= \langle \text{first line} \rangle \langle \text{page} \rangle + \langle \text{last line} \rangle$
$\langle \text{first line} \rangle$	$= \langle \text{comment char} \rangle \text{COMMENT} \langle \text{CRLF} \rangle$

† Based on Samuel (1977).

```

<page>           = <comment char><start><end><name>*(CRLF)
<comment char> =
<name>           = <token>|<token>(<number>)
<start>         = disk block number
<end>           = disk block number
<last line>     = <comment char>; (CRLF)

```

The index was maintained as a comment by <first line>, <last line>, and <comment char>. The editor would examine the file extension to determine what type of file it was (FORTRAN, LISP, Pascal, etc.) and determine what values should be assigned to these meta-symbols. Each page in the file was described by a <page> in the index. The first and last disk blocks of the page were given by <start> and <end>, respectively. The <name>, if present, allowed that page to be accessed by a character string name rather than by page number.† If the <name> contained a parenthesized number, the editor would search down the page for that name. The <name> could be edited by a special mode in the editor.

Retrieving a page was a matter of reading the disk blocks belonging to the page into primary memory. Writing a page simply put the buffer contents back into the page's disk blocks, while zero filling the unused disk block(s). If editing expanded the page so that it would not fit in its allocated disk space, the file was first expanded by moving pages in a file and then adjusting the index to make it agree with the file.

Other processors—compilers and text editors—can read the specially formatted file without difficulty. As mentioned earlier, the index page is a comment. The unused portions of disk blocks contained zeroes, which in ASCII are NUL characters. By convention, NUL is ignored on input by all programs. Files produced by other programs are converted to the special format by the editor. The special format is indicated by a NUL as the first character in the file. No other program would produce a NUL as a first character.

† Any unambiguous abbreviation of a <name> would work.

To appear in Human-Computer Interaction, G. Salvendy and M. Oshima, eds., Elsevier Publishing Co., 1984, which contains papers from the First USA-Japan Conference on Human-Computer Interaction, Honolulu, August 1984.

STATISTICAL METHODOLOGY IN THE LITERATURE ON HUMAN FACTORS IN COMPUTER PROGRAMMING

JOHN M. HAMMER

Center for Man-Machine Systems Research, Georgia Institute of Technology, Atlanta, Georgia 30332 (USA)

INTRODUCTION

This article examines some actual and recommended practices for design of experiments in human factors of computer programming. The first practice examined is the actual level of power in the statistical tests conducted on controlled experiments. Power is defined as the probability of accepting the alternative hypothesis (that a difference exists) when it is true. Power depends on the number of subjects, the squared difference in means relative to subject variance (termed effect size), and the commonly reported significance level, usually $p=.05$. Because programmer variance is usually considered to be relatively high, statistical power was hypothesized to be relatively low in this literature. This hypothesis was tested by calculating the power of tests in the published literature and comparing the average power to recommended levels and to other similar studies.

The second experimental practice examined was methods for controlling programmer variance. Basically, this was an examination of the literature for tests (e.g., grade point average, months of experience) that correlated with programmer performance. If good tests can be found, they can be used to make experiments more sensitive by accounting for the predicted performance in the experimental design.

POWER OF STATISTICAL TESTS IN THE LITERATURE

Power has been defined as the probability of accepting the alternative hypothesis of a difference due to treatments, given that this hypothesis is true. In general, statistical testing involves establishing two mutually exclusive hypotheses. The first is the null hypothesis (H_0) of no difference due to changes in the independent variable. The second is the alternative hypothesis (H_1) that this difference does exist. There are Type I and Type II errors which correspond to H_0 and H_1 , respectively. The probability of a Type I error (Type II error) is that of accepting H_0 (H_1) when it is false. Reported for virtually every statistical test is the probability of Type I error, or significance level (e.g., " $p<.05$ "). Power, which is virtually always omitted, is 1 minus the probability of Type II error.

Power is important both before and after an experiment. Before an experiment, power can be used to plan rationally the number of subjects to be used. The experimenter must select a significance level (usually, $p=.05$), a minimal power level (power=.80 is recommended (Cohen, 1977)), and an effect

size. From power tables, the appropriate number of subjects can be determined. The most difficult selection is effect size, since it requires the experimenter to predict the cell means and the variance before the experiment is run. In this study, observed effect sizes are calculated and tabulated along with power. Knowledge of these observed values should be an aid to future experimental planning. Prediction can be based, at least partially, on past observation.

Power is also important after an experiment, especially for interpreting effects that lack significance. Many researchers are reluctant to accept the null hypothesis in this situation. In fact, a calculation of power reveals what should be done. If, for effects of interest, power is high, the null hypothesis might well be accepted. High power simply indicates that the posited effect size of interest would have been detected if it existed. If, on the other hand, power is low, judgment should be suspended until an experiment is run (or the existing one replicated) with more subjects or other precision-increasing refinements.

Literature Reviewed

Articles from journal articles and conference proceedings on human factors in computer programming were selected for power analysis. Technical reports and theses were not examined. This admittedly biases the results somewhat, since unpublished experimentation, especially that never committed to paper, is often suspected to have low power.

The literature of controlled experiments on software complexity was, notwithstanding the above, also omitted from the study. The reason is the fundamental difference in the goal of this area for explaining variance in human performance. Human factors experiments attempt to show that an experimental factor has a significant effect on human performance. As will be shown later, such a factor might typically account for 10 to 40% of the variance. Software complexity, on the other hand, tries to predict human performance as completely as possible. It typically can account for 60 to 80% of the variance.

Rules for Power Analysis

The rules for power analysis were as follows:

1. Only tests significant at $p \leq .05$ were examined even though other marginally significant results were reported. This practice further biases the observed power in an upward direction. Further, power was computed using $p = .05$ even if a lower p was stated. Only two-sided testing was used, even if the author(s) used one-sided tests.
2. Only ANOVA F-ratios, t tests, and correlation coefficients (r) were examined. Chi-square, binomial, and nonparametric tests were ignored. No tests on differences in means (Duncan, Newman-Keuls) were examined.
3. In ANOVAs, the significance of the overall mean and all interactions were ignored. The latter were ignored because interactions are not typically sought in most designs, are difficult to interpret in the framework of this study, and often could not be studied due to lack of information.
4. A maximum of 10 tests per experiment were analyzed for power. If an article reported more than one experiment, it could have up to 10 tests included for each experiment. This was done to avoid a bias in favor of experiments with many tests. The first ten tests presented were analyzed.

5. Sufficient information had to be present to do the power calculations (F-test: cell size, means; t-test: cell size; r: cell size). For F tests, the mean square error often had to be estimated from the expected value formula for F.
6. The test must have been on some aspect of human performance that was measured under controlled experimental conditions. Regressions between two variables, neither of which represented human performance, were ignored.

All F and t test measures of effect size were converted to the square root of percent variance explained to allow comparison with r (Cohen, 1977).

Results

The total number of tests that were power analyzed was 122. The power averaged .83; its standard deviation was .19. Only 36% of the tests had power less than the recommended value of .80. The effect size, expressed in terms of the square root of the percentage of variance explained, averaged .44; its standard deviation was .14. The effect size data were roughly normally distributed, though skewed slightly to the right. Using Cohen's terminology of small ($r=.10$), medium ($r=.25$), and large ($r=.50$) effects, 67% of the effects are medium up to large, and 28% are large (Cohen, 1977). Using recommended medium effect size in pre-experimental power analysis would appear to be quite conservative, since less than 5% of the effects are less than medium.

Similar power analyses of other published literature have been done. Comparison of this study with others is difficult because we calculated effect sizes whereas others assumed various sizes and then determined the power. When an effect size of $r=.50$ was assumed (the closest value to our observed $r=.44$), the following were observed:

<u>Study</u>	<u>Average Power</u>	<u>Tests with power<.80</u>
Chase and Chase, 1976	.86	28%
Brewer, 1972	.78	29%
Katzer and Sodt, 1973	.79	46%

It should not be assumed that actual effect sizes in these other areas are as large as $r=.50$.

Conclusion

The power of tests in human factors literature on computer programming is as high as that in other areas where power analytic studies have been done. The original hypothesis of low power is incorrect. Ideally, experimenters would begin to incorporate a power analysis into their research planning. The distribution of (significant) observed effect sizes, as given above, should assist this planning.

VARIANCE CONTROL

Individual differences in programmer performance are a major problem in designing experiments. The ratio of best to worst performance for a group of programmers is often claimed to be 10:1 or 20:1 (Grant and Sackman, 1967) (Curtis, 1980) (Dickey, 1981) (Curtis, 1981). This difference is much larger than the 1.5:1 and 4:1 found for experts and intermediate level users, respectively, in a text editing task (Card et al., 1983). It should be noted that these large differences have been observed primarily on debugging times.

If tests were available to predict these differences to some degree, the predictions could be accounted for in the experimental design. The experiment would then become more sensitive, i.e., better able to detect true effect differences or to use fewer subjects. The appropriate designs which account for tests (termed concomitant variables) are randomized block designs and analysis of covariance (ANOCVA). The former uses the concomitant variable to group subjects into relatively homogeneous blocks. Each subject in the block is then randomly assigned to a treatment. ANOCVA performs a regression on concomitant variable simultaneously with an analysis of variance on the independent and dependent variables. ANOCVA is, however, not likely to be useful for two reasons (Keppel, 1973). First, ANOCVA requires many assumptions be true for it to be valid. Second, it is superior to randomized blocks designs only when the concomitant variable is correlated $r > .60$ with programmer performance. Since this is unlikely, randomized block designs would be preferred.

The results reported here are based on an examination of the same literature used in the power study. Basically, I looked at regression studies and experiments which had already attempted to account for programmer differences. Space limitations preclude the inclusion of a table which would allow direction examination of the correlations.

For professional programmers, months of programming experience has been found to be a fair predictor for program reading and writing performance. Correlations of .50 were found between the logs of experience and program writing plus debugging time (Chrysler, 1978). Correlations of .45 and .78 were found between experience and program comprehension scores (Moher and Schneider, 1981). They also found a multiple correlation of .62 between experience plus number of computer science courses and program writing time. Their high correlations must not be viewed too enthusiastically, for they purposefully sought out very diverse groups of subjects. Higher correlations are expected under such situations (Montgomery and Peck, 1982).

For professional programmers, there does not appear to be any good predictor for debugging tasks. No significant correlations were found between experience and debugging time in (Grant and Sackman, 1967). Experience was not found significant on tasks of program comprehension, modification, or debugging in (Sheppard et al., 1979). This result is counter to the above findings of Chrysler and Moher and Schneider. They did find the number of known programming languages and the number of familiar FORTRAN concepts to be correlated with debugging performance for professionals with less than 3 years experience. This result did not hold for more experienced professionals.

For advanced computer science students, a number of highly predictive measures appear to be available. In (Moher and Schneider, 1981), multiple correlations of .66 to .74 were found between program comprehension and writing tasks and the regressors: number of computer science courses, computer science grade point, and years of programming. The advantage of these three regressors is that they are relatively independent.

For beginning programmers, many regressors were tried in (Barfield et al., 1983), (Lucas and Kaplan, 1974), (Mayer, 1975), and (Shneiderman, 1977). From the standpoint of having large correlation coefficients and appearing in more than one study, the best regressors would appear to be SAT-Math scores, college course grade(s) either in the introductory programming course or in calculus or chemistry, and years of experience programming. Given that many beginning students today will have personal computer experience, it should be included in any attempt to predict performance.

For both professional and student programmers, a pretest is a possible choice for a concomitant variable. If the experimental task is program comprehension, one or more initial program comprehension pretests (the same test(s) for all subjects) could be used as a concomitant variable. Correlations between 3 modification task scores ranged from .31 to .60 and between 3 modification scores and recall scores ranged from .39 to .49 (Shneiderman, 1977). Correlations between scores on reading and writing tasks varied from .63 to .69. One disadvantage of pretesting is additional resources invested in it. This may be especially so if multiple pretests must be given to determine a stable level of performance.

An alternative to any use of concomitant variables is repeated measures, in which a subject is run under every experimental condition in the experiment. The subject serves in effect as his or her own control. While this approach may at first seem to be ideal, problems can and do arise. Consult (Poulton, 1982) and (Greenwald, 1976) for details.

Conclusion

Methods have been discussed for reducing programmer variance through the use of a concomitant variable for randomized blocking. Using these results, it should be possible to increase substantially the precision of experiments on computer programming. Very little effort is required to sort subjects into relatively homogeneous blocks prior to random assignment to experimental conditions. Given the wide range of research from which these conclusions have been drawn, they should be regarded cautiously. The potentially large returns certainly merit investigation.

SUMMARY

This study has examined the literature on human factors in computer programming to study two aspects of programmer variance. The first was to determine if the reportedly large differences in programmers caused statistical tests to be of low power and effects small relative to noise. This appears to be untrue. The second aspect studied was methods for controlling for large programmer variance in experimental designs. A number of promising concomitant variables were identified for randomized blocking, which should be able to increase the precision of experiments in this area.

ACKNOWLEDGMENT

This research was supported under NSF Grant IST 82-17440.

REFERENCES

- Barfield, W., LeBold, W.K., Salvendy, G., and Shodja, S., 1983. Cognitive factors related to computer programming and software productivity. Proc. Human Factors Society - 27th Annual Meeting, 647-651.
- Card, S.K., Moran, T.P., and Newell, A., 1983. The Psychology of Human-Computer Interaction. Erlbaum, Hillsdale, NJ, 469 pp.
- Brewer, J.K., 1972. On the power of statistical tests in the American Educational Research Journal. Amer. Educ. Res. J., 9: 391-401.
- Chase, L.J. and Chase, R.B., 1976. A statistical power analysis of applied psychological research. J. Applied Psychol., 61: 234-237.

- Chrysler, E., 1978. Some basic determinants of computer programming productivity. *Comm. ACM*, 21: 472-483.
- Cohen, J., 1977. *Statistical Power Analysis for the Behavioral Sciences*. Academic, New York, 474 pp.
- Curtis, B., 1980. Measurement and experimentation in software engineering. *Proc. IEEE.*, 68: 1144-1157.
- Curtis, B., 1981. Substantiating programmer variability. *Proc. IEEE.*, 69: 846.
- Dickey, T.E., 1981. Programmer variability. *Proc. IEEE.*, 69: 844-845.
- Grant, E.E. and Sackman, H., 1967. An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Trans. Human Factors Elec.*, 8: 33-48.
- Greenwald, A.G., 1976. Within-subjects designs: To use or not to use? *Psychol. Bull.*, 83: 314-320.
- Katzer, J. and Sadt, J., 1973. An analysis of the use of statistical testing in communication research. *J. of Communication*, 23: 251-265.
- Keppel, G., 1973. *Design and Analysis: A Researcher's Handbook*. Prentice-Hall, Englewood Cliffs, NJ, 658 pp.
- Mayer, R.E., 1975. Different problem-solving competencies established in learning computer programming with and without meaningful models. *J. Educ. Psychol.*, 67: 725-734.
- Moher, T. and Schneider, G.M., 1981. Methods for improving controlled experimentation in software engineering. *Proc. Fifth Int. Conf. Soft. Eng.*, 224-233.
- Montgomery, D.C. and Peck, E.A., 1982. *Introduction to Linear Regression Analysis*. Wiley, New York, 504 pp.
- Poulton, E.C., 1982. Influential companions: Effects of one strategy on another in the within-subjects designs of cognitive psychology. *Psychol. Bull.*, 91: 673-690.
- Sheppard, S.B., Curtis, B., Milliman, P., and Love, T., 1979. Modern coding practices and programmer performance. *Computer*, 12: 12, 138-146.
- Shneiderman, B., 1977. Measuring computer program quality and comprehension. *Int. J. Man-Machine Studies*, 9: 465-478.

STATISTICAL METHODOLOGY IN THE LITERATURE ON HUMAN FACTORS IN COMPUTER PROGRAMMING

JOHN M. HAMMER

Center for Man-Machine Systems Research, Georgia Institute of Technology, Atlanta, Georgia 30332 (USA)

INTRODUCTION

This article examines some actual and recommended practices for design of experiments in human factors of computer programming. The first practice examined is the actual level of power in the statistical tests conducted on controlled experiments. Power is defined as the probability of accepting the alternative hypothesis (that a difference exists) when it is true. Power depends on the number of subjects, the squared difference in means relative to subject variance (termed effect size), and the commonly reported significance level, usually $p=.05$. Because programmer variance is usually considered to be relatively high, statistical power was hypothesized to be relatively low in this literature. This hypothesis was tested by calculating the power of tests in the published literature and comparing the average power to recommended levels and to other similar studies.

The second experimental practice examined was methods for controlling programmer variance. Basically, this was an examination of the literature for tests (e.g., grade point average, months of experience) that correlated with programmer performance. If good tests can be found, they can be used to make experiments more sensitive by accounting for the predicted performance in the experimental design.

POWER OF STATISTICAL TESTS IN THE LITERATURE

Power has been defined as the probability of accepting the alternative hypothesis of a difference due to treatments, given that this hypothesis is true. In general, statistical testing involves establishing two mutually exclusive hypotheses. The first is the null hypothesis (H_0) of no difference due to changes in the independent variable. The second is the alternative hypothesis (H_1) that this difference does exist. There are Type I and Type II errors which correspond to H_0 and H_1 , respectively. The probability of a Type I error (Type II error) is that of accepting H_0 (H_1) when it is false. Reported for virtually every statistical test is the probability of Type I error, or significance level (e.g., " $p<.05$ "). Power, which is virtually always omitted, is 1 minus the probability of Type II error.

Power is important both before and after an experiment. Before an experiment, power can be used to plan rationally the number of subjects to be used. The experimenter must select a significance level (usually, $p=.05$), a minimal power level (power=.80 is recommended (Cohen, 1977)), and an effect size. From power tables, the appropriate number of subjects can be determined. The most difficult selection is effect size, since it requires the experimenter to predict the cell means and the variance before the experiment is run. In this study, observed effect sizes are calculated and tabulated along with power. Knowledge of these observed values should be an aid to future experimental planning. Prediction can be based, at least partially, on past observation.

Power is also important after an experiment, especially for interpreting effects that lack significance. Many researchers are reluctant to accept the null hypothesis in this situation. In fact, a calculation of power reveals what should be done. If, for effects of interest, power is high, the null hypothesis might well be accepted. High power simply indicates that the posited effect size of interest would have been detected if it existed. If, on the other hand, power is low, judgment should be suspended until an experiment is run (or the existing one replicated) with more subjects or other precision-increasing refinements.

Literature Reviewed

Articles from journal articles and conference proceedings on human factors in computer programming were selected for power analysis. Technical reports and theses were not examined. This admittedly biases the results somewhat, since unpublished experimentation, especially that never committed to paper, is often suspected to have low power.

The literature of controlled experiments on software complexity was, notwithstanding the above, also omitted from the study. The reason is the fundamental difference in the goal of this area for explaining variance in human performance. Human factors experiments attempt to show that an experimental factor has a significant effect on human performance. As

will be shown later, such a factor might typically account for 10 to 40% of the variance. Software complexity, on the other hand, tries to predict human performance as completely as possible. It typically can account for 60 to 80% of the variance.

Rules for Power Analysis

The rules for power analysis were as follows:

1. Only tests significant at $p \leq .05$ were examined even though other marginally significant results were reported. This practice further biases the observed power in an upward direction. Further, power was computed using $p = .05$ even if a lower p was stated. Only two-sided testing was used, even if the author(s) used one-sided tests.
2. Only ANOVA F-ratios, t tests, and correlation coefficients (r) were examined. Chi-square, binomial, and nonparametric tests were ignored. No tests on differences in means (Duncan, Newman-Keuls) were examined.
3. In ANOVAs, the significance of the overall mean and all interactions were ignored. The latter were ignored because interactions are not typically sought in most designs, are difficult to interpret in the framework of this study, and often could not be studied due to lack of information.
4. A maximum of 10 tests per experiment were analyzed for power. If an article reported more than one experiment, it could have up to 10 tests included for each experiment. This was done to avoid a bias in favor of experiments with many tests. The first ten tests presented were analyzed.
5. Sufficient information had to be present to do the power calculations (F-test: cell size, means; t-test: cell size; r : cell size). For F tests, the mean square error often had to be estimated from the expected value formula for F.
6. The test must have been on some aspect of human performance that was measured under controlled experimental conditions. Regressions between two variables, neither of which represented human performance, were ignored.

All F and t test measures of effect size were converted to the square root of percent variance explained to allow comparison with r (Cohen, 1977).

Results

The total number of tests that were power analyzed was 122. The power averaged .83; its standard deviation was .19. Only 36% of the tests had power less than the recommended value of .80. The effect size, expressed in terms of the square root of the percentage of variance explained, averaged .44; its standard deviation was .14. The effect size data were roughly normally distributed, though skewed slightly to the right. Using Cohen's terminology of small ($r = .10$), medium ($r = .25$), and large ($r = .50$) effects, 67% of the effects are medium up to large, and 28% are large (Cohen, 1977). Using recommended medium effect size in pre-experimental power analysis would appear to be quite conservative, since less than 5% of the effects are less than medium.

Similar power analyses of other published literature have been done. Comparison of this study with others is difficult because we calculated effect sizes whereas others assumed various sizes and then determined the power. When an effect size of $r = .50$ was assumed (the closest value to our observed $r = .44$), the following were observed:

<u>Study</u>	<u>Average Power</u>	<u>Tests with power < .80</u>
Chase and Chase, 1976	.86	28%
Brewer, 1972	.78	29%
Katzer and Sadt, 1973	.79	46%

It should not be assumed that actual effect sizes in these other areas are as large as $r = .50$.

Conclusion

The power of tests in human factors literature on computer programming is as high as that in other areas where power analytic studies have been done. The original hypothesis of low power is incorrect. Ideally, experimenters would begin to incorporate a power analysis into their research planning. The distribution of (significant) observed effect sizes, as given above, should assist this planning.

VARIANCE CONTROL

Individual differences in programmer performance are a major problem in designing experiments. The ratio of best to worst performance for a group of programmers is often claimed to be 10:1 or 20:1 (Grant and Sackman, 1967) (Curtis, 1980) (Dickey, 1981) (Curtis, 1981). This difference is much larger than the 1.5:1 and 4:1 found for experts and intermediate level users, respectively, in a text editing task (Card et al., 1983). It should be noted that these large differences have been observed primarily on debugging times.

If tests were available to predict these differences to some degree, the predictions could be accounted for in the experimental design. The experiment would then become more sensitive, i.e., better able to detect true effect differences or to use fewer subjects. The appropriate designs which account for tests (termed concomitant variables) are randomized block designs and analysis of covariance (ANOCVA). The former uses the concomitant variable to group subjects into relatively homogeneous blocks. Each subject in the block is then randomly assigned to a treatment. ANOCVA performs a regression on concomitant variable simultaneously with an analysis of variance on the independent and dependent variables. ANOCVA is, however, not likely to be useful for two reasons (Keppel, 1973). First, ANOCVA requires many assumptions be true for it to be valid. Second, it is superior to randomized blocks designs only when the concomitant variable is correlated $r > .60$ with programmer performance. Since this is unlikely, randomized block designs would be preferred.

The results reported here are based on an examination of the same literature used in the power study. Basically, I looked at regression studies and experiments which had already attempted to account for programmer differences. Space limitations preclude the inclusion of a table which would allow direction examination of the correlations.

For professional programmers, months of programming experience has been found to be a fair predictor for program reading and writing performance. Correlations of .50 were found between the logs of experience and program writing plus debugging time (Chrysler, 1978). Correlations of .45 and .78 were found between experience and program comprehension scores (Moher and Schneider, 1981). They also found a multiple correlation of .62 between experience plus number of computer science courses and program writing time. Their high correlations must not be viewed too enthusiastically, for they purposefully sought out very diverse groups of subjects. Higher correlations are expected under such situations (Montgomery and Peck, 1982).

For professional programmers, there does not appear to be any good predictor for debugging tasks. No significant correlations were found between experience and debugging time in (Grant and Sackman, 1967). Experience was not found significant on tasks of program comprehension, modification, or debugging in (Sheppard et al., 1979). This result is counter to the above findings of Chrysler and Moher and Schneider. They did find the number of known programming languages and the number of familiar FORTRAN concepts to be correlated with debugging performance for professionals with less than 3 years experience. This result did not hold for more experienced professionals.

For advanced computer science students, a number of highly predictive measures appear to be available. In (Moher and Schneider, 1981), multiple correlations of .66 to .74 were found between program comprehension and writing tasks and the regressors: number of computer science courses, computer science grade point, and years of programming. The advantage of these three regressors is that they are relatively independent.

For beginning programmers, many regressors were tried in (Barfield et al., 1983), (Lucas and Kaplan, 1974), (Mayer, 1975), and (Shneiderman, 1977). From the standpoint of having large correlation coefficients and appearing in more than one study, the best regressors would appear to be SAT-Math scores, college course grade(s) either in the introductory programming course or in calculus or chemistry, and years of experience programming. Given that many beginning students today will have personal computer experience, it should be included in any attempt to predict performance.

For both professional and student programmers, a pretest is a possible choice for a concomitant variable. If the experimental task is program comprehension, one or more initial program comprehension pretests (the same test(s) for all subjects) could be used as a concomitant variable. Correlations between 3 modification task scores ranged from .31 to .60 and between 3 modification scores and recall scores ranged from .39 to .49 (Shneiderman, 1977). Correlations between scores on reading and writing tasks varied from .63 to .69. One disadvantage of pretesting is additional resources invested in it. This may be especially so if multiple pretests must be given to determine a stable level of performance.

An alternative to any use of concomitant variables is repeated measures, in which a subject is run under every experimental condition in the experiment. The subject serves in effect as his or her own control. While this approach may at first seem to be ideal, problems can and do arise. Consult (Poulton, 1982) and (Greenwald, 1976) for details.

Conclusion

Methods have been discussed for reducing programmer variance through the use of a concomitant variable for randomized blocking. Using these results, it should be possible to increase substantially the precision of experiments on computer programming. Very little effort is required to sort subjects into relatively homogeneous blocks prior to random assignment to experimental conditions. Given the wide range of research from which these conclusions have been drawn, they should be regarded

SUMMARY

This study has examined the literature on human factors in computer programming to study two aspects of programmer variance. The first was to determine if the reportedly large differences in programmers caused statistical tests to be of low power and effects small relative to noise. This appears to be untrue. The second aspect studied was methods for controlling for large programmer variance in experimental designs. A number of promising concomitant variables were identified for randomized blocking, which should be able to increase the precision of experiments in this area.

ACKNOWLEDGMENT

This research was supported under NSF Grant IST 82-17440.

REFERENCES

- Barfield, W., LeBold, W.K., Salvendy, G., and Shodja, S., 1983. Cognitive factors related to computer programming and software productivity. Proc. Human Factors Society - 27th Annual Meeting, 647-651.
- Card, S.K., Moran, T.P., and Newell, A., 1983. The Psychology of Human-Computer Interaction. Erlbaum, Hillsdale, NJ, 469 pp.
- Brewer, J.K., 1972. On the power of statistical tests in the American Educational Research Journal. Amer. Educ. Res. J., 9: 391-401.
- Chase, L.J. and Chase, R.B., 1976. A statistical power analysis of applied psychological research. J. Applied Psychol., 61: 234-237.
- Chrysler, E., 1978. Some basic determinants of computer programming productivity. Comm. ACM, 21: 472-483.
- Cohen, J., 1977. Statistical Power Analysis for the Behavioral Sciences. Academic, New York, 474 pp.
- Curtis, B., 1980. Measurement and experimentation in software engineering. Proc. IEEE., 68: 1144-1157.
- Curtis, B., 1981. Substantiating programmer variability. Proc. IEEE., 69: 846.
- Dickey, T.E., 1981. Programmer variability. Proc. IEEE., 69: 844-845.
- Grant, E.E. and Sackman, H., 1967. An exploratory investigation of programmer performance under on-line and off-line conditions. IEEE Trans. Human Factors Elec., 8: 33-48.
- Greenwald, A.G., 1976. Within-subjects designs: To use or not to use? Psychol. Bull., 83: 314-320.
- Katzer, J. and Sadt, J., 1973. An analysis of the use of statistical testing in communication research. J. of Communication, 23: 251-265.
- Keppel, G., 1973. Design and Analysis: A Researcher's Handbook. Prentice-Hall, Englewood Cliffs, NJ, 658 pp.
- Mayer, R.E., 1975. Different problem-solving competencies established in learning computer programming with and without meaningful models. J. Educ. Psychol., 67: 725-734.
- Moher, T. and Schneider, G.M., 1981. Methods for improving controlled experimentation in software engineering. Proc. Fifth Int. Conf. Soft. Eng., 224-233.
- Montgomery, D.C. and Peck, E.A., 1982. Introduction to Linear Regression Analysis. Wiley, New York, 504 pp.
- Poulton, E.C., 1982. Influential companions: Effects of one strategy on another in the within-subjects designs of cognitive psychology. Psychol. Bull., 91: 673-690.
- Sheppard, S.B., Curtis, B., Milliman, P., and Love, T., 1979. Modern coding practices and programmer performance. Computer, 12: 12, 138-146.
- Shneiderman, B., 1977. Measuring computer program quality and comprehension. Int. J. Man-Machine Studies, 9: 465-478.

Significance Testing of Rules in Rule-Based Models of Human Problem Solving

C. MICHAEL LEWIS AND JOHN M. HAMMER

Abstract—Rule-based models of human problem solving have typically not been tested for statistical significance. Three methods of testing rules—analysis of variance, randomization, and contingency tables—are presented. Advantages and disadvantages of the methods are also described.

INTRODUCTION

Many researchers have used rule-based systems to model human problem solving [1], [3], [6], [7], [11], [12]. Typically, the rule-based system has a large number of rules, each of which has several free variables that were adjusted during the modeling process. For the most part, significance testing of these rules has not been much of a consideration. It should be. It is certainly possible to describe N data perfectly with N rules using a trivial model that simply reproduces the data. While there is no evidence that this has happened in any of the research reported to date, there is a certain danger of overfitting a rule-based model.

In this article we present three methods for testing the statistical significance of rules and other components of rule-based models. Throughout this article we shall assume that the percentage of behavior matched (e.g., commands) is the performance measure of interest. Two of the testing approaches, however, are not limited to this measure. They may be used to study any performance measure, though it may well be possible for a rule to produce a statistically significant effect on one performance measure but not another. The remainder of this article contains a section on notation, three sections on testing by analysis of variance, randomization, and contingency tables, respectively, and two concluding sections on applicability of the various tests and validity of these models.

NOTATION

A rule-based system consists of three components. The first is a set of rules of the form *IF* condition *THEN* action. The meaning of the rule is that if *condition* is true, then *action* could be taken. For example, the following rules describe behavior at a traffic-light-controlled intersection:

IF	in intersection	THEN proceed
IF	yellow and arrival at intersection before the light turns red	THEN proceed
IF	yellow and arrival at intersection after light turns red	THEN stop
IF	green	THEN proceed
IF	red	THEN stop
IF	red and right turn	THEN proceed

If this model can successfully match human behavior, then the rules form a model of the human. Often, the rules are interpreted as a model of the human's knowledge. Intuitively, the better the model matches human behavior, the better the model.

The rules can be transformed easily into a computer program as follows. First, control statements are added that cause the program to examine the rules repeatedly and execute those whose conditions are true. Second, in order to compare model and subject actions, an input statement is added before the first rule. This statement reads the state vector (e.g., the lights, the traffic, short term memory) that was available to the human when his or her decision was made. The program looks something like this the following:

```

WHILE TRUE DO BEGIN
  READ(STATE);
  IF (in intersection) THEN proceed
  ELSEIF (yellow) AND (predict arrival at
    intersection before light turns red) THEN proceed
  ELSEIF (yellow) AND (predict arrival at
    intersection after light turns red) THEN stop
  ELSEIF (green) THEN proceed
  ELSEIF (red) AND (right turn) THEN proceed
  ELSEIF (red) THEN stop
END;
```

The second component of a rule-based system is a conflict resolution strategy. It selects the rule to execute when multiple conditions are true. In the above example, a rank-order resolution strategy was shown. It simply uses the first rule that matches. The ranking of rules can then be interpreted as a subject's strategy. Some other conflict resolution strategies include random selection, meta-knowledge, and backtracking. A random selection strategy simply picks at random one of the many matching rules. A meta-knowledge strategy has a higher level rule-based system that chooses which rule to execute. A backtracking strategy will, if necessary, try all possible matches. It should also be noted that it may be possible to write the rule conditions so that there is always exactly one rule that matches.

The third component of a rule-based system is the input and internal variables. The input variables correspond to external data. The internal variables correspond to human short-term memory, which may be changed by the action part of rules. Both internal and input variables are examined by the condition part of rules.

Evaluation of Models

When comparing subject and model performance, the model is usually run open-loop without any knowledge of subject actions. In other words, the model can simply be treated as another subject. When comparing subject and model behavior, the model is usually run closed-loop as follows. The model has as input the same state vector the subject saw. The model chooses an action, and then it is recorded whether the subject and model agree. Then the subject's action is used to control the system, and the process repeats. The reason for always following the subject's action is as follows. If the subject and model action differed and both were used, then the state vectors would be unequal after applying these actions. The model and the subject would then be working on different problems, and a comparison of their actions would make little sense.

The following sections on testing rule-based models will specify ways in which the model will be modified and then run. The typical modifications are to delete or modify one or more rules. Running a model, perhaps in a modified form, means to compare its overt behavior, (e.g., commands) to a subject's and determine the percentage in agreement.

Manuscript received April 5, 1985; revised August 12, 1985. This work was supported by NASA-Ames Research under grant NAG 2-123.

The authors are with the Center for Man-Machine Systems Research, Georgia Institute of Technology, Atlanta, GA 30332, USA.

IEEE Log Number 8405909.

ANALYSIS OF VARIANCE

The analysis-of-variance approach is the simplest of the three approaches for testing rule significance. To use it, each rule in the model is equated with an independent variable. The meaning of the variable is that at its high level, the rule is in the model, and at its low level, the rule is deleted from the model. The rule-based model is then run 2^N times (for each subject), which corresponds to a run with each possible subset of rules present. It must make sense for the model to do nothing, or else the model must be augmented before testing with a special nondeletable rule that applies when no other rule applies. The resulting data can then be analyzed as an N -way factorial.

To economize on model runs, fractional factorial designs should be used. The full factorial design, proposed above, will estimate the effects of many high-order interactions that cannot occur. In fact, the interpretation of an interaction is that the corresponding rules interact. An example would be two rules, the first of which stores some value in a temporary variable and the second of which uses the temporary variable. Such rule interaction is common, but rarely do many rules interact. An inspection of the rule-based model will reveal what interactions could occur. It should be possible to create experimental designs which test only the desired interactions.

The testing of condition components of rules is also possible. In this case the reduction in error attributable to the greater specificity provided by the additional condition can be evaluated. Suppose, for example, that a significance test of each of the conjunctive conditions of a rule is desired. For example

IF condition₁ AND condition₂ AND condition₃ THEN

Proceeding as before, three independent variables might be equated, one with each of the three conditions. A three-way ANOVA could be run to test each of the three clauses. It would most likely be necessary to estimate the value of the response at the point where all three conditions have been deleted from the rule. Obviously, this process could be extended to cover all of the conditions for all of the rules in the model.

The testing of groups of rules as a whole is also possible. To do this, an independent variable is equated with several rules, not just one as was done initially. The experimental interpretation is that the entire set of rules is either present or absent from the model during an experimental run. This pooling of rules corresponds to a supersaturated experimental design and may be the only economical means of testing models with many rules. One logical choice for pooled rules would be interacting rules. Another choice would be the modeler's organization of rules into groups (e.g., S -rules and T -rules [6]).

Analysis of variance makes several assumptions, one of which is that error residuals are normally distributed. Moderate departures from this assumption do not produce large deviations in calculated and actual significance levels. If the normality assumption is known or seriously thought to be incorrect, an approximate technique [4] may be used. Simply, the data are replaced with their ranks, and the remainder of the analysis of variance calculations remain unchanged. The significance levels produced by this method are reported to be nearly equal on normally distributed data to that produced by the standard F -test. The rank transformation is more robust with respect to the distribution of the data, though it is not a distribution-free test. Finally, the hypothesis being tested here is whether the presence of a rule (or some other similar entity) explains a significant amount of variance in the subjects' performance. This significance is independent of the significance of other rules (or other entities) but may be dependent on the conflict resolution strategy. It is important to note the hypothesis because the next section tests somewhat different ones.

RANDOMIZATION

The second approach to testing a rule involves forming a randomization distribution by randomly permuting a rule. Suppose a particular rule is under test. Its action can be replaced by

a random action (e.g., a random number generator that chooses commands according to *a priori* frequencies). The model, with a single modified rule, can be run many times. Its matching performances can be considered a randomization distribution. The model in its unaltered form can then be run, and its resulting performance be referred to the randomization distribution. If its matching were higher than 95 percent of the randomly generated values, the null hypothesis could be rejected at the five-percent level (one-sided). The null hypothesis would be that a random action would be as suitable as the proposed action in the rule under test. The empirically determined significance level is partial in that it is potentially dependent on all the other rules being present in the model as well as conflict resolution strategy.

The condition part of a rule can be tested by a very similar method. There is a minor difficulty in that a random number generator in the condition part of a rule does not appear to make sense. A solution would seem to be to create various mutant conditions by randomly selecting condition clauses from other rules in the model. The null hypothesis being tested here is that random conditions are as suitable as the proposed condition in the rule under test. The significance level attained is partial just as the one obtained in testing actions.

An entire rank-order conflict resolution strategy may also be tested by randomization. Basically, a randomization distribution of performances can be obtained by running all possible rank orderings (or a Monte Carlo sample) of rules. The performance of the model with the original rank ordering can be referred to this distribution as above. The significance level obtained is dependent on the rules.

CONTINGENCY TABLES

Contingency tables are used to analyze nominal data. If the following is a rule-based model:

IF condition₁ THEN action₁
IF condition₂ THEN action₂
:
IF condition_n THEN action_n.

Then a contingency table may be set up as follows:

	action ₁	action ₂	...	action _n
condition ₁			...	
condition ₂				
:			:	
condition _n				
NOT (condition ₁ OR ... OR condition _n)				

The last row in the table covers the conditions that are not covered by any rules. The observed data fill the table in the obvious way: for a given state vector and subject action, the unique condition which holds is determined, and the cell under the subject's action is incremented. A model that matched the data perfectly would have all zero entries off the diagonal.

Certain restrictions must be met to employ contingency tables.

- 1) Conditions must be mutually exclusive (2 rules cannot fire at the same time).
- 2) Actions must be overt.
- 3) Each action must be unique (2 rules cannot issue the same action).

These restrictions may be met in a variety of ways. Mutual exclusivity will be satisfied by any model containing conflict resolution, rank-ordering, or disjoint rule provisions. The unique action requirement may be accommodated by phrasing composite rules in which constituent rules prescribing the same action are joined by disjunction.

The performance of the rules in matching the data can be evaluated with a chi-square or similar tests. The hypothesis is tested whether conditions and actions are independent, i.e., whether there is a significant difference between the proportions given the rules and the overall proportions. As a consequence, rules containing infrequently used actions receive more latitude using these tests than they do under a simple percentage of commands matched measure.

Testing a set of rules is also possible as follows. The null hypothesis is that there is no relationship between the action and the conditions aside from the relationship that is already described by the existing rules. Consider the test for the rule:

		action ₁	...	action _n
X1 = 1	X2 = 1	Delete		
	X2 = 2			
X1 = 2	X2 = 1	Delete		
	X2 = 2			

Two statistics are computed. The first is a maximum likelihood estimate of chi-square (G^2) for the complete table. The second is a test of quasi-independence [2] for a reduced table in which cells corresponding to rule(s) under test are excluded. This corresponds in the preceding table to one cell per row for conditions covered by the rule(s). If the original G^2 is significant and the quasi-independent one is not, this implies that the rules capture the dependency of the actions on the conditions. While attractive in directly referencing observables, this method requires large samples with replications of observed combinations of variables. (Unobserved combinations are treated as structural zeros.)

Other Statistics

A nonparametric analog to the coefficient of determination R^2 is τ_b [8], which may be used to determine the percentage of variance explained in the actions by a rule or rule set. Thus

$$\tau_b = \frac{\sum_i \frac{1}{X_{i+}} \sum_j X_{ij}^2 - \frac{1}{N} \sum_i X_{i+}^2}{N - \frac{1}{N} \sum_i X_{i+}^2}$$

X_{ij} = table entry in row i , column j ,

$$X_{i+} = \sum_j X_{ij}$$

$$X_{+j} = \sum_i X_{ij}$$

N = total number of observations.

Individual rules, the disjunction of rules issuing a particular action, or the complete rule set consolidated into disjunctions by action can be evaluated in this way. If uncovered observations are excluded, τ_b may be interpreted as the extent to which actions covered by the rule are explained. If all observations are present, a $N + 1$ st category should be formed following the distribution of the uncovered actions. This τ_b is interpretable as the extent to which rules explain all the actions.

Values of τ_b are asymptotically related to χ^2 , allowing significance testing. Thus

$$\chi_{(I-1)(J-1)}^2 = (N-1)(I-1)\tau_b.$$

This statistic tests the hypothesis that $\tau_b = 0$, corresponding to the premise that there is no relation between conditions and the actions prescribed by the rule(s).

A similar statistic proportional reduction in error (PRE) [2] measures the reduction in error achieved by predicting actions based on the rules rather than assigning the modal action under

all rules and is given by

$$\text{PRE} = \frac{\sum_i P_{i+} - P_{+m}}{1 - P_{+m}}$$

where

$$P_{i+} = \max_j (P_{ij})$$

$$P_{+m} = \max_j (P_{+j})$$

$$P_{ij} = N_{ij}/N.$$

As demonstrated by this potpourri of procedures, a unified technique for testing rule significance based on multinomial sampling is yet to be developed. PRE answers the pragmatic question of gains in prediction. The quasi-independence procedure provides its complement by testing for unmodeled consistencies. Rules can be simultaneously tested in a contingency table but their contributions to rule set performance will remain unknown. τ_b allows both significance testing and estimation of effects but cannot evaluate rule set performance without pooling rules by action.

APPLICABILITY OF VARIOUS TESTING METHODS

For testing the degree to which a model's behavior matches a subject's, all three methods will work. A contingency table is clearly the best, however, since it requires the minimum in computation. Randomization is clearly the worst technique because of the large amount of computation and the partial significance levels it produces. A fractional factorial ANOVA is clearly superior to randomization on both of these points. ANOVA and randomization can both be used to test rules that modify internal, unobservable states. Contingency tables cannot.

For testing overall performance measures, (e.g., time to solution, total errors) only randomization and ANOVA are suitable, with ANOVA preferred. Ordinarily, much more emphasis is placed on behavior than on performance, since behavior is much more difficult to model. There are situations in which testing hypotheses about both performance and behavior is desirable. One might want to show that a certain set of rules will affect behavior but not performance. For example, Morris and Rouse [10] have observed that theoretical training given process control operators often fails to change their performance. It would be interesting to test this concept analytically in a rule-based system. For example, a group of rules might be identified as the intended consequences of theoretical training. The model might be run with and without these rules, using ANOVA to evaluate performance measures and contingency tables to evaluate behavioral differences.

The randomization method can be used on two hypotheses. The first, and more important, is to test the significance of a rank ordering of rules. This would seem to be the only way to test this type of resolution strategy. The second use is to test the hypothesis that part of a rule performs no better than random. This test would seem to be of little use, since ANOVA can test nearly the same hypothesis.

VALIDITY

The previous methods are generally devoted to evaluation of rule performance and do not address the issue of rule validity. Just as a high R^2 does not imply that all terms of its regression equation are significant, a high τ_b does not vouchsafe for the future predictiveness of its rules. This distinction becomes important in the identification phase of rule-based modeling. Unlike identification based on parameter estimation, the identification of rules requires a search of the space of possible rules. An inductive pattern matcher must consider a large number of potential rules. In evaluating identification it becomes necessary

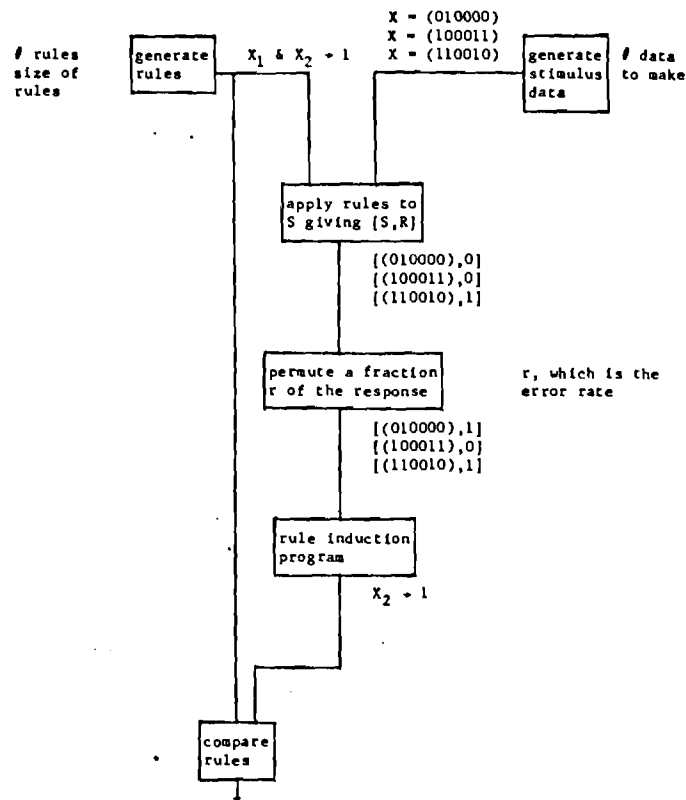


Fig. 1. Block diagram for rule induction with an example output from each block.

to account for the probability of finding rules of comparable quality by chance. To answer this question the structure of the event space (observed combinations of condition variables), distribution of actions, and extent of search (set of possible rules) must be considered simultaneously.

Eilbert and Christensen [5] refer to this problem as contrivedness: "... the tendency of a search procedure to uncover apparent patterns where none exist." They suggest a randomization test for measuring the extent to which a search procedure uncovers contrived rules. The data consist of many pairs of state vectors with subject responses. The state vectors are left undisturbed, but the responses are randomly permuted. The resultant permuted data has reasonable state vectors paired with random responses. Contrivedness is the degree to which the search procedure can make sense of this random data. When many permuted data sets are searched, the search procedure results from a randomization distribution against which the results from the original, unpermuted data can be referred. While the previously mentioned randomization test will give an idea of how opportunistic the search procedure is, it does not say how to refine the search procedure so as to avoid contriving rules.

CONCLUSION

This article has identified several ways of testing a rule-based model of human problem solving. The amount of testing seems to be on a par with the size of the model. Left unresolved for the most part was the problem of contrivedness of automatic rule identification. It seems fitting to close with the description of an interesting and difficult question in identification of rules. As stated earlier, many cognitive models have been built using rule-based models. Sometimes these models are built when the investigator has access to the subject's thinking. This is always the case in developing a rule-based expert system. Other investigators, particularly those running experiments with humans, may have only the data (i.e., commands) to examine.

An important theoretical question is the limits to identification of rules from data that contain response errors. While there has been work in machine learning, it does not seem that anyone has examined this question [9]. It does seem important, because it bears on our ability to construct models. This problem also seems to be very difficult to solve formally. Hence, a preliminary investigation could be done via simulation, as shown in Fig. 1. Basically, the approach is to generate some rules and some random stimuli, apply the rules, add noise, and try to identify the rules from the noisy data.

The following would seem to affect identification:

- 1) the amount of data and its coverage of the stimuli domain,
- 2) size and number of rules,
- 3) the number of times a rule fires, and
- 4) the level of noise.

It might also be interesting to investigate the addition of oracle variables in rule identification. An oracle variable is an extra variable (beyond the original stimulus vector) that provides information that ordinarily is not available. The first oracle variable might be a single bit to tell whether the response was in error. Another set of oracle variables would identify which rule fired. Yet another set of oracle variables could identify the variables that are part of the rule that fired. While these oracle variables may appear to be practically giving the solution to the identification program, they do not. These variables would be treated the same as any of the real stimulus variables. The identification program would have to infer the meaning of these variables in order to make use of them.

While it does appear theoretically interesting to determine how much oracle variables can add, there are important practical benefits as well. Oracle bits could approximate the hunches of a human investigator. For example, the investigator may suspect certain data to be in error, a certain rule to have fired, or that only certain variables could be influencing the operator's decision.

(from a verbal protocol). These hunches are a second order human-machine system: the investigator's attempt to identify (with a program) the rules of the human in the first-order human-machine system.

REFERENCES

- [1] J. R. Anderson, *The Architecture of Cognition*. Cambridge, MA: Harvard, 1983.
- [2] Y. M. M. Bishop, S. E. Fienberg, and P. W. Holland, *Discrete Multivariate Analysis: Theory and Practice*. Cambridge, MA: MIT, 1975.
- [3] S. K. Card, T. P. Moran, and A. Newell, "Computer text editing: An information processing analysis of a routine cognitive skill," *Cognitive Psychology*, vol. 12, 1980.
- [4] W. J. Conover, *Practical Nonparametric Statistics*, 2nd ed. New York: John Wiley, 1980.
- [5] R. F. Eilbert and R. A. Christensen, "Contrivedness: The boundary between pattern recognition and numerology," *Pattern Recognition*, vol. 15, no. 3, 1982.
- [6] R. M. Hunt and W. B. Rouse, "A fuzzy rule-based model of human problem solving," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-14, no. 1, 1984.
- [7] A. Knaeuper and W. B. Rouse, "A rule-based model of human problem solving behavior in dynamic environments," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 6, Nov. 1985.
- [8] R. J. Light and B. H. Margolin, "An analysis of variance for categorical data," *J. Amer. Statistical Association*, vol. 66, 1971.
- [9] R. Michalski, J. Carbonell and T. Mitchell, *Machine Learning*, Palo Alto, CA: Tioga, 1983.
- [10] N. M. Morris and W. B. Rouse, "The effects of type of knowledge upon human problem solving in a process control task," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 6, Nov. 1985.
- [11] A. Newell and H. A. Simon, *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.
- [12] W. B. Rouse, S. H. Rouse, and S. J. Pellegrino, "A rule-based model of human problem solving performance in fault diagnosis tasks," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-10, no. 7, 1980.



DESIGNING TOMORROW TODAY

E-24-616

Georgia Institute of Technology
School of Industrial and Systems Engineering
Atlanta, Georgia 30332-0205
(404) 894-2300

January 29, 1986

Ms. Lynn Boyd
Office of Contract Administration
Campus

Dear Ms. Boyd:

This letter and its attachments are a progress report for the period January 1984 to January 1985 for NSF IST 82-17440 (E24-616). Please arrange to forward it to the appropriate official at NSF.

The remainder of this letter describes the technical aspects of the work performed. The actual period might more properly be considered June 1984 to January 1986, not the one stated above. The remainder of this letter will briefly weave together a background for these articles, which are:

Hammer, J.M., Significance testing of the keystroke-level model, submitted for publication.

Lewis, C.M. and Hammer, J.M., Significance testing of rule-based models, to appear in IEEE Trans. Systems, Man, and Cybernetics, 1986.

Hammer, J.M., Inference of rule-based models: A research prospectus (a concept paper to interest funding agencies).

As an overview, a concern for methodology has been the primary focus during the period covered. Along this line has been

1. reexamination of the statistical validity of a seminal work in human-computer interaction,
2. the development of several methods of testing rule-based models of cognition,
3. a novel way of looking at the problem of identifying rules, which is interesting both as computation and cognition.

The first article, ("Significance testing of the keystroke-level model"), is a reexamination of Card, Moran, and Newell's keystroke-level model. Their approach to predicting the time for a task is to break it into subtasks, determine the time for each type of subtasks, and then add the appropriate number of subtask times. This approach is basically a linear model; the most common way of testing such a model is to use multiple linear regression, which the original authors did not use. The attached article describes multiple linear regression testing of the model, which gives greater ability to test hypotheses about the model. The conclusions the authors originally drew do not appear to be as strong under this more powerful form of testing.

The second article, ("Significance testing of rules in rule-based models of human problem solving"), describes three methods for testing rule-based models of human problem solving. Many researchers have built such models that fit the data overall. There has not been any testing on individual rules to determine if each contributes to model fit. As the article shows, testing is very straightforward, for it uses conventional statistical methods (i.e., ANOVA and fractional factorials, randomization, and Chi-square).

The most interesting part of the second article (appears in the conclusion) is the concept of rule identification. This is developed more fully in the third paper ("Inference of rule-based models"). Although significance testing can show that a given rule is statistically significant, it cannot show that the given rule is identical or close to one used by the observed process. A model containing rules equal that of an observed process would be more scientifically interesting than a model whose rules were only statistically significant in mimicing the process

behavior. The approach of rule identification gives rise to three problems, one in computational theory and two in cognitive science. The computational question is first, how to identify a rule-based model that uses a finite amount of unobservable memory. In addition, it is important to determine the distance between the estimated rules and the process rules. This distance likely depends on a large number of factors such as the amount of data, the amount of memory in the process, the noise, etc.

The first cognitive question is the use of the above algorithm to identify an expert's method of performing an interactive task. The second cognitive question is how to modify the algorithm to make it identify rules as humans do. More specifically, what cognitively plausible limitations can be placed on the inference algorithm so that its inferences are similar to those of a novice user. The user's task is to use some interactive device and to form a mental model of its behavior. The inference method would observe the same data as did the human and would form a rule-based model of the interactive device. The human's rules and the inference method's rules would then be compared. An overall goal of this research would be to study mental models in a more objective way.

To conclude, there is a validity theme running throughout this work. It begins in statistics and ends in identification and cognition. I am quite excited by the ideas contained in the third article. I hope you find them interesting.

Sincerely,

John M. Hammer
Assistant Professor

INFERENCE OF RULE-BASED MODELS:
A RESEARCH PROSPECTUS

John M. Hammer
Center for Man-Machine Systems Research
Georgia Institute of Technology
Atlanta, Georgia 30332

ABSTRACT

This proposed research is concerned with using an inductive rule inferencer to identify and model a human who is operating or learning a complex interface. It is proposed first to modify an existing induction program [Michalski 1980] so that it can infer rules using variables beyond those provided as input to it. These hidden variables correspond either to human short-term memory or a mode in a complex interface. The second part of the proposed research would use the rule inferencer to build a rule-based model of an expert who is using a complex interface. Specifically, the model would predict how the expert chooses an editing method. The third part of the proposed research would use the rule inferencer as a model of how a novice user builds a rule-based mental model of a complex interface. The rules produced by the inferencer, which would also be attempting to build a rule-based mental model of the interface, would be viewed as optimum. These optimum rules would be compared to the user's rules in order to detect systematic biases. These systematic biases would be used to modify the inductive identifier to make its rules more like the user's. A suitably modified inductive inferencer could then be considered to be a cognitive model of human learning.

INTRODUCTION

Given a set of data, how might the mechanism producing that data be determined? One of the most frequently used mechanisms in modeling human problem solving is the rule-based model (e.g., [Anderson 1983], [Newell and Simon 1972], and [Rouse 1980]). Many rule-based models have been built, but there has been little consideration of using an inductive rule inferencer to construct these models. The potential benefits of using an inferencer include speed, problem size, objectivity, and testability. As for speed and size, an inductive rule inferencer can probably analyze a large data set faster than a human. An important advantage of an inference method is its objectivity. While results can be tailored to some extent by limits on inputs and the form of allowed inferences, the results are repeatable. It should be possible to test the inference method to determine if it is capable of structure recovery.

The remainder of this section contains a description of the domain to be studied and a description of the inductive inference problem. The remaining sections of the paper are on:

- the inductive inferencer and the changes needed to make it usable on the problems in the proposed research.
- how the inferencer will be used to study the expert's choice of method.
- how the inferencer will be used to model the novice's acquisition of a mental model.
- the relationship of the proposed research to that of Ohlsson and Langley, VanLehn, and Tatsuoka.
- a conclusion.

A Description of the Domain Being Considered

There are three general characteristics of domains to which this methodology could be applied. The first is that the system be essentially

discrete. A discrete system has discrete inputs, states, and outputs. In contrast, the topic of most work in mental models is on analog systems: motion, heat, and electricity, [Gentner and Stevens 1983]. A second characteristic, which is implied by the first, is that there is a rule-based description of the system. This means only that the system behavior can be described by a rule-based system, which is theoretically equivalent to whatever means are used to implement the system. Third, the discrete inputs may be relatively large in number. The implications for induction of this are discussed later in this section.

Any system that has these characteristics could be used as the domain to test the inference methods. Some examples are interactive computing, operating a complex, discrete device such as a telecommunication system, or fault diagnosis of a complex system.

The domain proposed is interactive computing, specifically text editing. Both experts and novices will be studied. The expert task is choosing from a large number of commands available that could accomplish the expert's goal [Hammer and Rouse 1982]. For example, suppose the goal of positioning the editor to another point in the text. Some possible methods include using a search string, scrolling the display and looking for the target, counting the relative distance to the target, pointing with a mouse, etc. Within each of these methods there may be sub-methods that represent different strategies (e.g., different search strings). The choice of positioning method may depend on display variables, previous methods used, and characteristics of the text. The research goal would be to infer a rule-based model of the expert's choice of method. Furthermore, the inferencer would be constrained to produce rules that were cognitively plausible, a topic developed further in a later section.

The inferencer will also be used to study the way a novice understands a text editor. By understand, we mean finding out if the user knows what the editor will do in response to a specific sequence of keystrokes (e.g., ESCAPE A RETURN). We shall not be concerned with whether the user can use the knowledge, even though this is an important question. The research goal would be to modify the inferencer (in cognitively plausible ways) to produce rules that are similar to the novice's.

A Description of the Inference Problem

The phenomena to be modeled is described formally as follows. There is a chronological sequence of data. Each datum contains an input (stimulus) and response. The input is a vector of discrete variables. The response is a single discrete variable.

$$\langle I_{11}, I_{12}, \dots, I_{1N}, R_1 \rangle$$

.

$$\langle I_{M1}, I_{M2}, \dots, I_{MN}, R_M \rangle$$

The inference problem is to build a rule-based model that would produce the sequence of responses from the sequence of inputs. It is often the case that a rule of the model depends only on some subset of the input variables. Furthermore, a rule may depend on unobservable memory variables. These memory variables may be changed by other rules in the model.

The formal view of the problem is applied to modeling the expert's choice as follows. The inputs are the variables displayed to the expert. The response is the expert's chosen method or command. The unobservable memory variable is the expert's goal. The problem is to build a rule-based model that

predicts the expert's responses from knowledge of the inputs. Note that the rules will make explicit how the unobservable variables are used and changed.

The formal view of the problem applied to novice learning is as follows. The input vector corresponds to the values displayed to the novice plus the input command entered by the novice. The response is the response of the interface to the command. The unobservable memory variables correspond to the modes and settings of the interface that are not displayed to the user. The inference problem of the novice is to identify a rule-based model of the interface. The research problem is to modify a computer-based inferencer so that its inferences are similar to that of the novice.

THE INDUCTIVE RULE INFERENCER

The motivations for using an inductive rule identifier include speed, size, objectivity, and testability. The latter two are important considerations from a methodological point of view. The use of a rigorous inference method would tend to remove subjectivity from the construction of rule-based models. For the most part, rule-based models have been built by humans using subjective inference from verbal protocols or the raw input-response data described earlier. While using subjective methods does not mean that its results are incorrect, an objective method is preferable. An additional advantage of an objective, computer-based method is that it would be usable in intelligent CAI courseware.

Testability is an additional advantage of an objective rule inferencer. Its ability to identify rules may be measured. In the section on identifying expert's rules and in Appendix A, a plan for testing structure recovery is

outlined. This plan, summarized briefly, would determine the extent to which the inferred rules would approximate the actual rules.

Inferencer Requirements

Three major requirements are made of an inferencer to be used in this research. First, it must be capable of identifying rules using few of the input variables when many potential variables are available. The reason for this is that realistic interfaces have a large number of variables, of which only a small number are important at any one time. A second requirement is to infer the use of memory. As described earlier, memory is a part of both interfaces and expert human operators. A third requirement is either that the inferred rules be in a cognitively plausible form or that the inference process be cognitively plausible. The specifics of this are deferred until the sections on expert and novice research. For the most part, the changes required by cognitive plausibility to the inferencer are relatively minor.

Michalski [1980,1983] has developed INDUCE, a practical algorithm for identifying input-output relationships from data. This algorithm can find rules involving few variables even when there are many input variables from which to choose. It cannot infer missing variables (e.g., the use of short term memory). Thus, it would have to be modified to do so. The remainder of this section describes how that would be done.

Modifications to INDUCE

The basis for adding an observable variable to a model is a choice between two models. One way of choosing between two models would be to select the one whose structure is closer to a known, a priori structure. One alternative when there is no known, a priori structure (e.g., no known memory limit) is to select the simpler model. The remainder of this section describes how the

current version of INDUCE would respond to data that depended on unobservable variables. It will also be obvious how to take advantage of this.

Suppose that INDUCE is attempting to identify a rule-based process that depends on one unobservable memory that may contain any one of a finite set of symbols. Since this unobservable is unavailable to INDUCE, it will search the input space for other variables whose random variables just happen to explain the data. Thus, the inferred rules will be the conjunction of an excessive number of clauses. Each rule explains only a relatively small number of the observed data. On the other hand, if the unknown variables were known to INDUCE, there would be fewer rules, each of which would cover more of the data. Thus, successful identification of unobservable variables will result in a simpler model.

To estimate the unobservable variable, the following process would be used. First, produce an inferred rule set from observable-only data. This rule set would have the characteristics described above. Then, form an augmented set of data by the following clustering process. Label each datum with the rule(s) that cover that datum. Find the median distance from a given rule to every other rule in the data, where the distance between two data is the time or number of chronological steps between them. The result of this will be a matrix of inter-rule distances. Cluster analysis can then be used to group the rules. Finally, each datum is augmented with one additional nominal variable: the cluster that covered that datum. The cluster, which is based on chronological proximity, is assumed to correspond to an operator's goal. The resultant augmented data is reanalyzed with INDUCE.

Note that the number of clusters is a free parameter in the above procedure. It would be varied to produce the maximum simplification of the model.

EVALUATION ON EXPERTS

The first use of the inferencer will be to build a rule-based model of an expert who is using a text editor. As stated earlier, the editing task to be modeled is the choice of method. The issues that are relevant to this research are the nature of cognitively plausible rules and the structure recovery and significance testing.

Cognitively Plausible Rules

The basis for cognitively plausible rules comes from our understanding of skilled performance. The following are usually held characteristic of an expert.

1. The expert can recognize patterns in the input.
2. Small differences in the input are recognized and taken advantage of to produce superior performance.
3. Backtracking and problem solving do not occur (at least for the tasks considered here).
4. A decision may be based on a large number of inputs, which have been organized into patterns to reduce memory load. The number of patterns used in a decision, however, is probably small. The patterns are not known a priori, unfortunately.
5. While there is pattern recognition in the expert, there are some cues that the expert would not use because they would take too long to calculate. For example, an exact distance is a cognitively implausible cue, though an estimated distance is not.
6. The activation of a previously used method may remain high enough to cause it to be reused.

The leap from the basis to the limits on rule form is somewhat tenuous.

Following are the limits and their justification (if necessary).

1. A rule may depend on a goal, which is an unobservable memory variable.
2. A rule may use only a limited number of inputs. The limit is larger for an expert than a novice.
3. If the input is represented at a higher level than the raw input, then the patterns must be cognitively plausible to compute. Basically, this is a question of how to represent the input to the inferencer.

Ideally, we would like to infer a representation level, but it is not clear how to do this.

4. The rule may use only simple pattern matching. There is no backtracking.
5. The rule may depend on the previous action.

Additionally, a rank-ordered conflict resolution strategy will be used. This is mostly a result of how the inferencer produces output. It would not be easy to change this. Finally, it would be desirable to be able to input hunches that certain decisions depended on certain variables. The sources of these hunches could be verbal protocols or eye tracker data. The only modification to INDUCE would be to bias it in favor of certain variables.

Significance Testing/Structure Recovery

Appendix A describes a significance testing/structure recovery problem in rule-based system identification. The problem is that after a set of rules has been identified, either by a human or by a program, it is still unknown how close the estimated rules are to the actual rules. The appendix describes an ambitious program for testing the effects of a wide variety of factors on the success of rule identification. To reduce the scope of that program, the following local testing is proposed. Suppose as a part of the research previously outlined that a three rule system is identified from 67 data, where each datum has an input vector of length fifteen. Then, the ability of the rule inferencer to identify rules would be tested under exactly those conditions. Several hundred simulations would be run with 67 data, each with input of width fifteen. The known, a priori rule sets would consist of two, three, and four rules. The rules used would be both similar and dissimilar to those originally identified, to determine if the inferencer was sensitive to such changes. This procedure would give a local measure of the capabilities of the rule inferencer.

INDUCTIVE RULE INFERENCE AS A MODEL OF RULE-BASED MENTAL MODEL FORMATION

This section proposes modeling the human who is learning a new, complex interface as an inductive inferencer. Viewed generally, this is clearly what the human is doing: attempting to ascertain the general operation of the interface from the specifics of a sequence of situations. A case could easily be made that inference is the human's preferred way of learning. Humans seem to prefer examples to explanatory text in documentation, at least for initial learning.

In this problem we want to concentrate on how the human forms a mental model. Our definition of mental model is the ability to predict the operation of the interface. Specifically, this means describing or displaying the interface to the user, telling the user what the next command is, and asking if the user knows what the response will be. Of course, there are other definitions of mental model. This is not to say that ours is right or wrong, only that it is the one used here. Furthermore, it may be that the user has internalized the operation of the interface (to the extent that the above questions can be answered) but still cannot use the internalization. This problem is not considered here.

The role of the inductive rule inferencer would be to serve as a model of the human. The inferencer would have access to the same data as the human. By using the inferencer, we expect to find systematic differences between its rules and the human's. These differences are due to differences in the inference methods of the inferencer and the human. Initially, the inferencer would not serve as a suitable theory of how the human infers a mental model. We intend to incorporate the systematic biases into the inferencer to make it a suitable model of rule-based mental model formation. The biased inferencer would then produce the same kind of rules as would humans. The remainder of

this section describes the kinds of cognitively plausible biases that might be incorporated into the inferencer.

Cognitive Bias in Induction

A number of cognitive biases in inference are presented and justified here. Some of them will clearly occur and some are merely plausible. In other cases there may be support for inference limits for certain kinds of constructs, but it may not be clear what form the limits take.

The first bias would be to limit the data available to the inferencer. INDUCE normally uses all of the data to infer rules. A plausible cognitive limit would be to use only the K most recent data, where K is relatively small. This limit would be consistent with what humans could retain in memory.

A second bias would be to limit the number of input variables in the data. For example, the input might contain ten variables. The induction process might be constrained to use only four of these; the others are ignored. While this kind of limit is almost certainly present in humans, it is difficult to model unless the four variables are known. It may be possible to use the display salience of the variables to choose the limited set that are attended to.

A third bias would be to expect rules to depend on surface variables rather than complex patterns built up out of surface variables. This is simply the complement to the expert's ability to use patterns.

A fourth bias is to expect rules to use only a small number of variables (three or four). This limit would be lower than that of the expert.

A fifth bias has to do with unobservable variables. Unfortunately, it is difficult to say what form this bias might take, although it seems certain that humans will have difficulty with this kind of inference. Of course,

discovering this phenomena or incorporating it into the model does not explain it.

To conclude, most of the inference limits proposed here are fairly simple: there is a limited ability to make use of the data. Our intent is first to gain more precision in describing these biases or, alternatively, to discover new ones. Following that, the biases are to be incorporated into the inferencer to form a cognitively plausible model of inference.

RELATED WORK

Ohlsson and Langley [1985], VanLehn [1983], and Tatsuoka [1985a,b,c] have conducted related research in the use of inductive or statistical inference in identifying models of arithmetic subtraction errors. The remainder of this section first briefly discusses their research and then discusses the differences between theirs and ours.

Ohlsson and Langley [1985] have studied automated diagnosis of subtraction bugs. Their program DPF searches for a sequence of intermediate states (a path) between the initial problem formation and the given student solution, which may be correct or incorrect. The search is best-first, where best is determined by an evaluation function. This evaluation function is intended to take cognitive plausibility into account. For example, it will not compute intermediate results that are never used, nor will it compute intermediate results twice. Its heuristics include minimizing memory due to the number of intermediate results, satisfying previously established goals and in the preferred order, achieving maximum satisfied subgoals per step, and minimizing the length and number of errors of the inferred path.

VanLehn [1983] has built a program that learns how to subtract from seeing worked examples. The program is given a sequence of ten lessons, each of which

teaches a different facet of subtraction (e.g., borrow, borrow from zero, etc.). The inductive part of the model constructs program that correctly solve the examples. Some of these programs contain systematic bugs, which are the result of improper, though plausible inference.

The execution of the programs produces buggy subtraction. The bugs exhibited by the program at various lesson points are collected into a large set. This set and the set of systematic bugs produced by students exposed to similar lessons are compared.

Tatsuoka and her colleagues [1985a,b,c] have developed statistical clustering techniques for classifying errors according to known bugs. Basically, there are a set of features to classify a problem worked incorrectly. This is input to a cluster analysis, which groups a number of solutions by these features. If a student sometimes uses a buggy rule, then there will be a cluster of responses that can be traced back to that rule. The advantage of this statistical approach is that it is insensitive to nondeterministic bugs.

Relationship of Proposed to Existing Research

The strongest connection between the two are the concepts of cognitive plausibility. These limits are used in the justification for limitation on search and induction in VanLehn's and Ohlsson and Langley's work. On the other hand, subtraction bugs seem to be limited to subtraction. Repair Theory [Brown and VanLehn 1980] describes how a buggy procedure might be modified to run when it does not quite fit the problem. It is not clear what there is to be repaired when the novice is forming a mental model. There may be an opportunity for repairs when the user recalls how the interface works. If the recall provides only a partial answer, repairs may be made to fill in the unknown. If the repairs are similar to Repair Theory, then it may be that the

mental model has a knowledge representation like a procedure. If the repairs are different, then there is evidence for a separate representation for mental models.

The weakest connection between the proposed and existing research has to do with the problem domain. The following table summarizes the differences between subtraction and human-computer interaction.

<u>Attribute</u>	<u>Subtraction</u>	<u>Human-Computer Interaction</u>
Previous research	Much-bug libraries	Very little-exploratory research-problem space uncertain.
Problem characteristics	Highly positional Small number of inputs One correct solution method No feedback on errors Solution unknown to student	Somewhat positional Large number of inputs Several solution methods Partial feedback on errors Text solution known to user
User	Problems disappear with maturation	Problems remain
Memory	Use of memory is closely related to the problem	Identification of memory use is more removed from problem and more difficult
Data	Temporally sparse	Temporally dense

Finally, there is a small but important difference in the research question. The existing research is concerned with bugs in a procedure. The proposed research is concerned with novice understanding of the underlying model. If the proposed research were to be done on subtraction, the question would be whether the novice understood the positional number system.

CONCLUSION

The proposed research has three noteworthy features. First, it investigates human interaction with a complex interface, an area that is generally recognized to be of considerable importance. Second, it uses an objective methodology - inductive rule inference - to study both expert and novice behavior. Third, the models built a cognitively plausible, not just ad hoc mechanisms for data explanation.

The plan is to use a 36 month period as shown below:

<u>months</u>	<u>description</u>
6-9	modifications to INDUCE
12	model of an expert
12-15	model of a novice

The estimated annual budget for this project is \$110,000.

APPENDIX A

A THEORETICAL INVESTIGATION INTO RULE IDENTIFICATION

To study the problem of rule identification, a theoretical investigation is proposed, which is summarized in Figure 1. A simple version of the process is described here. It begins with a known set of rules. Input data¹ (stimuli) are then generated, and the rules are executed on the data to produce output (responses). A few of these stimuli-response pairs are randomly chosen to be errors; their responses are permuted. This noisy data is then given to a rule induction program, which attempts to generate rules from data. Note that this program has access only to the stimulus-response pairs, not to the original rules. The output of the induction program is a set of estimated rules. The original and estimated rules are then compared. Ideally, the two sets of rules should be identical.

Table 1 lists a number of factors that would seem to affect rule identification. For example, the amount of data has an effect on all inference procedures. For this particular problem, the ratio of the number of data to the number of rules may be more useful. Another example is the complexity of the rules, both in their specificity and in their use of short-term memory. Obviously, any rule that depends on short-term memory (a hidden mechanism) is inherently more difficult to identify.

In principle, it might be possible to prove theorems on the effects on identification by various factors shown in Table 1. In practice, these proofs seem to be very difficult. Therefore, simulation will be used to examine

¹Throughout this prospectus the data are assumed to come from a human who is operating (or repairing) a complex system (e.g., a computer, a communication system, a malfunctioning engine). The stimuli are the variables that are perceivable on the human-machine interface. The responses are the actions or commands issued by the human.

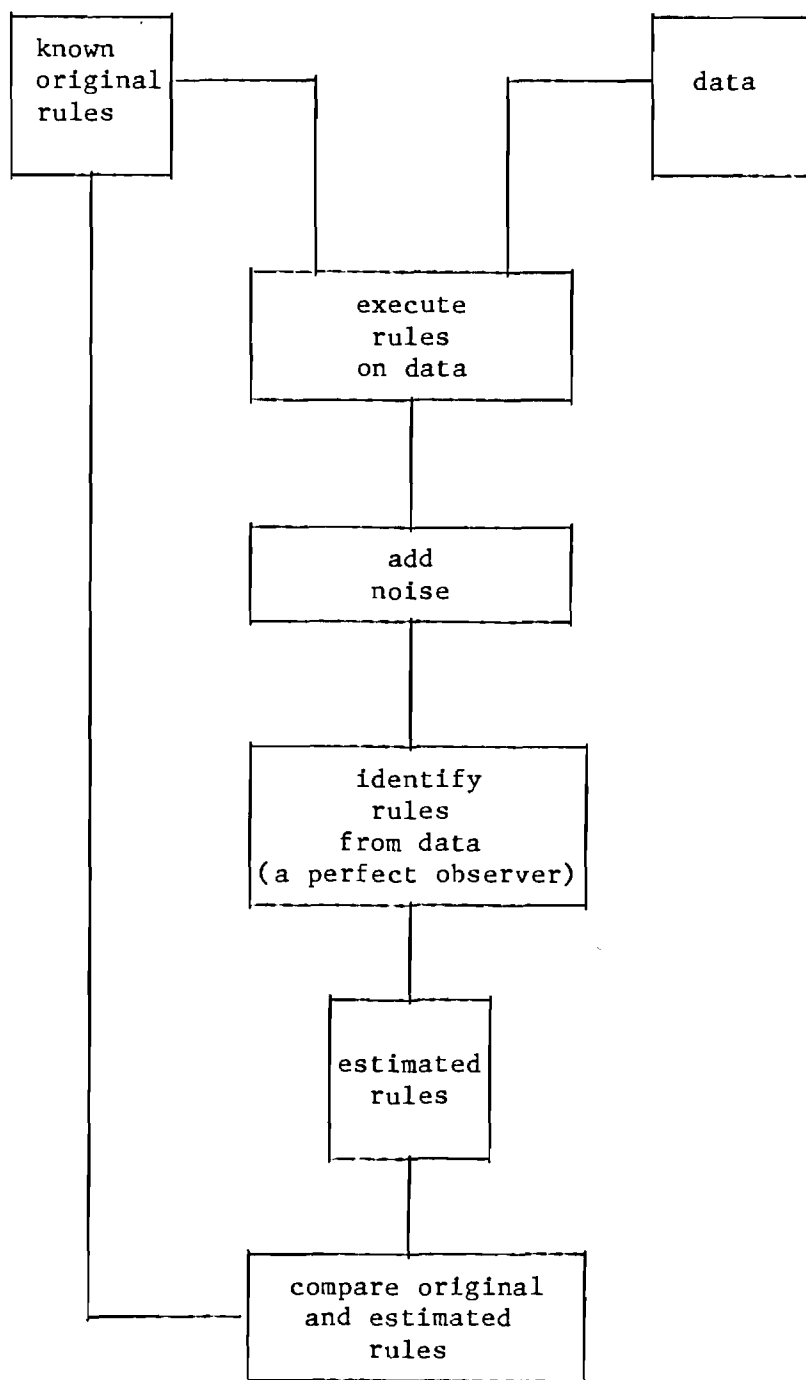


Figure 1. Diagram of the rule identification problem

Table 1. Factors that potentially affect rule identification.

Data

1. The amount of data.
2. The distribution of data in the input space. If the input is viewed as occurring in a multi-dimensional space, the degree to which the space is covered uniformly (or alternatively clustered in some areas) would seem to affect it.
3. Correlation in the data. If two input variables are correlated, then there naturally should be difficulty deciding which one a rule should use.

Noise

1. The level of noise in the data.
2. The structure of noise. Errors could occur for multiple reasons:
 - a. random
 - b. systematic - an incorrect rule is chosen for execution
 1. cue missed (attention failed)
 2. cue change missed (change expectation failed)
 3. rule preconditions nearly match
 4. rule had previously been chosen and had retained some activity

Rules

1. The number of rules.
2. The complexity of rules. The number of variables and the number of clauses.
3. The number of times a rule fires. A rule with very specific preconditions may be difficult to identify, especially if the noise were relatively strong.
4. The use of memory.

effects of these factors. The effects of the factors would be described in terms of effects on confidence intervals.

Confidence intervals require a measure of distance between two rules. A distance may be calculated as follows. Two rules may be compared if and only if their right hand sides (i.e., the actions) are equal; otherwise, the rules are incomparable. The distance between two rules is then the difference between the two boolean functions in their left hand sides. Each of these two functions can be expressed as a set of minterms according to the laws of boolean algebra. The absolute distance between these two sets is the number of minterms that they do not have in common. The relative distance is the absolute distance divided by the total number of terms.

The distance between two sets of rules would be considered to be the sum of the distances between individual rules that are paired under some between-set matching. If it is not obvious how to match the rules, then the minimal distance could be defined as the minimum across all possible matchings.

Important Theoretical Issues Related to this Study

There are several important theoretical issues that might be resolved by the methodology to be developed. First, suppose a rule-based model can be made to match much of the action-by-action data of a human, who is assumed to be following a set of rules. Does a high degree of action-by-action matching imply the model's rules are equal to the human's? The simulation evaluation would tell us the conditions under which this is true.

The second question is whether rule-based models are appropriate models of human problem solving. This issue occasionally receives intense debate. If a rule inferencer which produces optimum rule sets is not too successful at matching behavior, the appropriateness of this type of model would be in

question. Currently, we do not know the answer to this because the degree of match can be controlled by the investigator's willingness to add additional rules to the model. Adding rules will improve the degree of match but brings the danger of an overfit model [Lewis and Hammer 1985].

The final question involves inference of a model of human error. Suppose that the human's rules are complete and correct. A human error, then, could be regarded as a failure in the control mechanism (or conflict resolution) that selects a rule for execution. Many mechanisms have been suggested (see [Norman 1981,1983] for examples). It would be interesting to attempt to distinguish inductively the various error mechanisms that were operating on a known set of rules. This is inference of the control mechanism which presumably could be approached in the same manner as inference of rules.

APPENDIX B

RELATED RESEARCH

Described below are two areas of research that are funded by NASA-Ames Research Center. Dr. Everett Palmer is the grant monitor. Both of these are supervised by Dr. Hammer. There is, of course, other research in the Center.

Aiding the Operator During Novel Fault Diagnosis

Human operators are often present in complex systems to diagnose novel failures. Unfortunately, specific training and written procedures are impossible for a novel failure because it cannot be foreseen. The research problem is to design a computer aid to assist the operator in diagnosing the failure [Yoon and Hammer 1985].

The aid contains two components: a qualitative model and a decision-making bias recognizer. The qualitative model has a component level representation of the device. This representation describes both the correct mode of operation as well as several known incorrect modes of operation. The operator's task is to determine the mode of operation of each component in the device. This assignment of modes must both explain the failure symptoms and be consistent. A consistent assignment obeys every constraint that a component imposes on its neighbors. To a certain extent, the qualitative model is able to solve this mode assignment problem, although the computation is combinatorially explosive. Note that if a component fails in a mode not described by the qualitative model, the operator will be able to extend the model by adding a new description.

The decision making bias recognizers observe the operator's actions on the interface. If a bias is detected (e.g., anchoring on an initial hypothesized failure), the operator is alerted or warned about the problem.

The evaluation plan is to build a model of the Orbital Refueling System, a

shuttle payload for refueling orbiting satellites. This system is larger and more complex than systems described by existing qualitative models.

Engineering students will then diagnose failures both with and without the aid.

Using $F=ma$ to Predict Aircraft Maneuvering Errors

One approach to building an error monitoring system is to code an expert system to watch for out-of-tolerance situations. The approach taken here is different. To predict maneuvering, deep knowledge is used. This knowledge is the aerodynamic equations of motion (represented symbolically), which represent the forces acting on the aircraft. The problem that is repetitively solved is to determine if a constraint will be violated in the near future by a particular control action the pilot might take. If not, other constraints and other controls are considered.

If a problem could occur at a particular time, then this knowledge represents a plan for avoiding constraint violation. In particular, this plan solves a very sticky problem of when automation should take control from the pilot. As time begins to run out, the pilot is warned about constraint violation. At the last possible instant, automation takes over if the pilot has not already solved the problem.

This approach using deep knowledge would seem to be more generable, reliable, and verifiable than an ad hoc expert system.

REFERENCES

- Anderson, J.R. 1983. The Architecture of Cognition. Harvard: Cambridge, MA.
- Brown, J.S. and VanLehn, K. 1980. Repair theory: A generative theory of bugs in procedural skills. Cognitive Science, 4.
- Gaines, B.R. 1976. Behavior/structure transformations under uncertainty, Int. J. Man-Mach. Stud., 8.
- Gentner, D. and Stevens, A.L. (eds.). 1983. Mental Models. Lawrence Erlbaum: Hillsdale, NJ.
- Gold, E.M. 1967. Language identification in the limit. Inf. Control, 10.
- Hammer, J.M. and Rouse, W.B. 1982. The human as a constrained optimal text editor. IEEE T. Sys., Man, Cyber., 12.
- Hammer, J.M. 1984. Statistical methodology in the literature on human factors in computer programming, in Salvendy, G. (ed.) Human-Computer Interaction. Elsevier: New York.
- Hammer, J.M. 1985. Significance testing of the keystroke-level model, submitted to IEEE T. Sys., Man, Cyber.
- Lewis, C.M. 1985. Identification Methods for Rule-Based Models, Ph.D. thesis in progress, School of Psychology, Georgia Institute of Technology, Atlanta, GA.
- Lewis, C.M. and Hammer, J.M. 1985. Significance testing of rules in rule-based models, to appear in IEEE T. Sys., Man, Cyber.
- Maryanski, F.J. and Booth, T.L. 1977. Inference of finite-state probabilistic grammars, IEEE T. Comput., 26.
- Michalski, R.S. 1980. Pattern recognition as rule-guided inductive inference, IEEE T. Pattern Anal. Mach. Intell., 2(4).
- Michalski, R.S. 1983. A theory and methodology of inductive learning, in Michalski, R.S., Carbonell, J.G., and Mitchell, T.M. (eds.) Machine Learning. Tioga: Palo Alto, CA.
- Newell, A. and Simon, H. 1972. Human Problem Solving. Prentice-Hall: Englewood Cliffs, NJ.
- Norman, D.A. 1981. Categorization of action slips, Psych. Review, 88(1).
- Norman, D.A. 1983. Design rules based on analyses of human error, Comm. ACM, 26(4).
- Ohlsson, S. and Langley, P. 1985. Identifying Solution Paths in Cognitive Diagnosis. Carnegie-Mellon Robotics Institute Report, CMU-RI-TR-85-2.

- Rouse, W.B. 1980. System Engineering Models of Human-Machine Interaction. North Holland: New York.
- Tatsuoka, K.K. 1985. Diagnosing Cognitive Errors: Statistical Pattern Classification and Recognition Approach. University of Illinois at Urbana-Champaign, CERL Report 85-1-ONR.
- Tatsuoka, K.K. and Yamamoto, K. 1985. Application of Component Scoring to a Complicated Cognitive Domain. University of Illinois at Urbana-Champaign, CERL Report 85-2-ONR.
- Tatsuoka, K.K. and Tatsuoka, M.M. 1985. Bug Distribution and Pattern Classification, University of Illinois at Urbana-Champaign, CERL Report 85-2-ONR.
- Van der Mude, A. and Walker, A. 1978. On the inference of stochastic regular grammars, Inf. Control, 38.
- VanLehn, K. 1983. Felicity Conditions for Human Skill Acquisition: Validating an AI-Based Theory. Xerox PARC Report CIS-21.
- Yoon, W.C. and Hammer, J.M. 1985. Aiding the operator during novel fault diagnosis, Proc. 1985 IEEE Conf. Sys., Man, and Cyber., pp. 362-365.

SIGNIFICANCE TESTING OF THE KEYSTROKE-LEVEL MODEL

John M. Hammer

Center for Man-Machine Systems Research

Georgia Institute of Technology

Atlanta, GA 30332

ABSTRACT

The Keystroke-Level Model [Card et al., 1980b] predicts the time for an expert user to complete a task using an interactive computer system. To use the model, the task is expressed as a sequence of subtasks such as keystrokes, mental delays, mouse pointing. The predicted task execution time is the sum of the subtask times.

While the overall model fits the data well, the contributions of the various subtasks to the model have not been tested for significance. In this article, we show how to do this as well as test the significance of the rules for placing mental delays. We also examine the data of Allen and Scerbo [1983], who among others found that the Keystroke-Level Model under-predicts task times.

INTRODUCTION

The use of linear models to predict task time for human-computer interaction is discussed here. In the first part, the Keystroke-Level Model [Card et al., 1980b] is presented and reviewed. Its components are then tested using linear regression. The results show, first, that it is simple to build and test these models. Second, simplifications and alternate interpretations of the Keystroke-Level Model are discussed. The second part of the paper examines the data of Allen and Scerbo [1983]. They as well as Roberts and Moran [1983] and Gould and Alfaro [1984] found the model under-predicted task times. A linear regression model is used to examine their data. Regression will allow us to determine the maximum to which a given linear model will fit the data.

LITERATURE REVIEW

The Keystroke-Level Model predicts the time for an expert to complete a routine task using an interactive computer system. To predict the time, the system designer must express the task as a sequence of elementary operators. The total execution time is

$$T_k * N_k + T_p * N_p + T_h * N_h + (T_{N_d} * N_d + T_{L_d} * L_d) + T_m * N_m + T_r$$

where the symbols are defined in Table 1. The N 's for overt actions come from counting the number of operators necessary to do the task in an optimal manner. The T 's for these actions were estimated from other sources of data — typing tests, pilot experiments, etc. For the mental operator, N_m is estimated by applying a set of five rules (shown below). T_m was estimated by regression. The model was able to explain 95% of the variance in average task times. From these results, Card et al. argue that the Keystroke-Level Model

is suitable for use in system design.

Tests of the Keystroke-Level Model

Roberts and Moran [1983] tested a set of benchmarks for evaluating editors. One of their performance measures was the time for an expert to complete a sequence of editing tasks. On the eight different editors tested, they found the Keystroke-Level Model to under-predict consistently the average time. The under-prediction, they claim, was from the subjects' using longer, more conservative methods than those of the model. Only relative differences in model predictions of approximately 25% or more were reliable predictors of real differences.

Allen and Scerbo [1983] repeated the above benchmark on the ED text editor. When using the optimal methods assumed by the Keystroke-Level Model, the prediction was 61% of the actual time. When subjects' actual methods were used, the predictions increased to 77% of the actual time. Based on these observations, they argue first that methods need to be predicted because of the improvement in accuracy. Second, they argue that more accurate parameter values alone will not improve accuracy. Instead, a more explicitly cognitive model is needed to predict the variety of cognitive delays that occur.

Gould and Alfaro [1984] conducted an experiment on text editing using a display editor, a simulated handwriting recognition system, and a simulated voice recognition system. They noted that the Keystroke-Level Model predictions were 33% to 50% of the time required for editing. They attribute the under-prediction to a difference in experimental conditions. For example, Card et al. conducted their experiments while placing a heavy emphasis on speed. The other studies reported above were conducted under more naturalis-

tic settings.

STUDY ONE

Card et al. used regression to estimate the value of T_m . Because the model is a linear combination of terms, linear regression can be used to estimate simultaneously all the T 's and test the significance of each term. Before explaining these results, a brief review of regression is given.

Regression

Regression is a method for building models from data. For example, suppose a model is desired of the following form:

$$T_{out} = T_0 + T_1 * N_1 + T_2 * N_2 + \dots + T_n * N_n$$

T_{out} is the output (response) and the N_i are inputs (regressors), all of which are given to the regression program. The program will estimate the T_i 's that cause the model to best fit the data. Furthermore, the regression program will provide the significance of each T_i , the probability that the T_i is different from zero. If it is not significant, then the corresponding term $T_i * N_i$ can be dropped from the model. (The remaining coefficients must then be reestimated to make the new model fit the data). This is justified because an insignificant term does not contribute to the predictive or descriptive power of the model.

The usual performance measures for a regression model are the coefficient of determination R^2 and the error mean square MS_E . R^2 is usually described as the fraction of variance explained. It is the amount of squared deviation of the response that can be attributed to variation in the regressors. All other things constant, a higher R^2 is better. MS_E is a measure of the squared dis-

tance between the model's predicted response and the observed response.

Before accepting a regression model, diagnostic tests should be performed to insure that it is not unduly influenced by a few data. The following were performed. The first test was for an outlier in the regressor space. Any datum that was an outlier (more than three standard deviations out) in the Mahalanobis distribution was removed. The second test was for data that exerted an undue influence on the coefficients. A datum with a Cook's distance > 1 was removed. In addition, diagnostic plots of residuals were examined.

Advantages and Disadvantages of Regression

Most of the advantages have already been mentioned: significance testing of terms, alternate model examination, tests for fit and outliers. A major advantage of regression is that it gives a lower bound on the model error, MS_E . This is because the coefficients are chosen to minimize the error between the predicted and actual response. Any other set of coefficients (say, one that makes more physical sense or is closer to a priori estimates of the coefficients) will have a larger error. Note that regression eliminates bias.

Both bias and error are important in the Keystroke-Level Model. Regression can be used only to study the latter. The importance of error from a practical standpoint can be seen in the following example. We would be pleased with a model that had estimates one-half of actual and a relative error (after rescaling) of 1%. A model with an underestimate of 1% and a relative error of 50% would be far less satisfactory.

The disadvantages of regression are potential traps that can befall the analyst who does not conduct tests of the model. The first is the danger of an overfit model. The coefficients are determined from the data, which may cause the model to look better than it is. This difficulty may be overcome by using many more data than free variables (regressors). The second danger is that the estimated coefficients may not be close to our a priori estimates of them. For example, many of these estimates of T_k in the results below are larger than the measured typing rate.* There are several possible explanations for this. The first is that the difference is due to a mild form of overfitting. This in particular can be misleading, because the regression model will not predict new observations any better than a model that had not been overfit. A second explanation might be that the regression estimate is better. It might seem reasonable that the typing rate during editing is slower than that of a typing test.

A third danger is that the regression model may be based on correlation, not causality. For example, Kendall and Yule [1950] observed a strong relationship ($R^2 = .98$) between the number of mental defectives per 10,000 population and the number of radio receiver licenses issued in the U.K. for the years 1924-1937. This relationship is nonsensical. What is really happening is that detection of mental competency and the cost of electronics are changing with time. The danger of correlation is avoided by using regressors that are certainly direct influences on the response.

A final danger is that the regression model may be controlled by only a few of the many data. An example of this would be an outlier in the regressor

*Rather than looking at the point estimate of the coefficient, one might examine some confidence interval about the estimate to see if it contains the a priori value.

space. It would inflate R^2 , thus making the model look better than it is. Data with undue influence were removed as described earlier by examining Mahalanobis' and Cook's distances.

Regression Testing of the Original Keystroke-Level Model

The original Keystroke-Level Model was tested with linear regression using the original 32 data from Table 9 of [Card et al., 1980b, p404]. The 32 tasks are four editing tasks, each done with three different text editors, five graphics editing tasks, each done with three different graphics editors, and five executive tasks, each done with a different executive program. For each task, the data are the observed task time (response) and the regressors the time for K, P, H, M, D, and R. It should be noted that the observed task time is the average of between 9 and 38 observations from a number of subjects. There are almost 900 observations behind the 32 task times.

In addition to testing the basic model, the seven different programs were entered as indicator variable regressors. There were three indicator variables for the three text editors, three for the three graphics editors, and one variable for the collection of executive programs. The indicator variable can show if the model is sensitive to the interactive system used.

The number of deleted M's were also entered into the regression. The Keystroke-Level Model has five rules, based on assumed user chunking behavior, which determine where M's should appear in the model. The rules are as follows.

0. Insert an M before every K that is not part of an argument string.
Insert an M before every P that selects a command.

An indicator variable is a regressor that can take a value of either zero or one.

1. If an operator following an M is fully anticipated in the operator preceding the M, delete the M.
2. If a string of MK's belong to a cognitive unit (e.g., a command name), delete all M's but the first.
3. If a K is a redundant terminator (e.g., the terminator of a command immediately following the terminator of its argument), delete the M before the K.
4. If a K terminates a constant string (e.g., a command name), then delete the M before the K. Keep the M before a variable string (e.g., an argument string).

Rule 0 was applied to determine the initial set of M operators. (The tasks are described in [Card et al., 1983].) Rules 1 through 4 were then applied to each task and the number of M operators deleted by each rule for each task was tabulated. The following expression is being added to the model for testing

$$T_{m_1} * N_{m_1} + T_{m_2} * N_{m_2} + T_{m_3} * N_{m_3} + T_{m_4} * N_{m_4}$$

N_{m_i} is the number of M operators deleted by rule i. T_{m_i} is the time for an M operator deleted by rule i. If the model is correct, the M_1 through M_4 terms will not be statistically significant.

Because of the physical interpretation of the operators, a problem containing zero K, P, H, D, M, and R should take zero time to execute. Hence, the intercept was forced to be zero. It should be noted that an intercept might be a suitable interpretation of average acquisition time, if that time had been included in the response. This interpretation depends on the assumption that the intercept is a constant independent of problem size. It should also be noted that allowing an intercept would not have substantially changed

the results reported here.

Model Criteria

The selection of the "best" regression model is partially a subjective matter and also dependent on the future use of the model [Montgomery and Peck, 1982] [Draper and Smith, 1981]. The potential uses of a model are data description, prediction, control, and parameter estimation. Since prediction is the purpose of the Keystroke-Level Model, the following criteria were used to select the best model.

1. Each term in the model must be statistically significant.
2. The MS_E should be minimized.

Results

The BMDP9R statistical program, all possible subsets regression, was run using the terms for K, P, H, D, M, R, and the seven indicator terms. (The deleted mental operator analysis appears later.) Some of the better models are shown in Table 2. All models are fit to the same data. Equation four contains the same terms as the original Keystroke-Level Model, though all coefficients were determined by regression. Some of these models explain more variance than the original model; none of them is much better than another.

The regression results can be interpreted as follows. First, the H operator, part of the original model, does not account for any variance. This is easily explained in that an H nearly always accompanies a P or a D. Thus, once P and D are in the model, the addition of H explains nothing. Second, the Executive term, as originally suggested by Card et al., is significant; Executive tasks take about 5 seconds less than would otherwise be predicted.

None of the editor indicator terms is significant. This suggests that the model predicts well for editors, and that new values for T_m and T_k are needed when Executive tasks are modeled.

The M operator is significant in the full model. There is a practical problem in that the unique variance explained by any of the operators is low when other operators are in the model (see Table 3). Since M does not explain much variance, it could be dropped from model 2 (giving model 5). There is a theoretical consideration that allows weakly significant terms to be dropped from a regression model, which was first observed by Box and Wetz [1973] (see also [Suich and Derringer, 1977]). They observed that statistical significance was insufficient for prediction. To be useful, the calculated F-ratio for the regression needs to exceed the critical ($p=.05$) value by a multiple of 4 or 5 at the least. Ellerton [1978] has observed that the same is true of partial F-tests (such as those on a single coefficient). This explains why dropping M would not affect prediction accuracy.

If the Executive task data is removed and the regression repeated, M may also be dropped for this reason, as its $F(1,22)=9.18$ is less than four times the critical value, 4.30. This practice may be useful with M, since it is much more difficult to count the number of M's in a task. This is primarily due to the need to consider the rules for placing M's. This is not necessary with physically observable actions. It should also be noted that the contribution of the D operator is also weakly significant. This may be due to it

being used only in 3 of the 32 tasks.

Testing the Deletion Rules

As described earlier, the number of deleted M operators was tabulated and entered as terms. Due to the high correlation between M1 through M4 and the original terms, a two step procedure was used. First, all significant terms (i.e., K, P, D, R, M, and Executive) were forced into the model. Then, M₁ through M₄ were tested to see if any of them explained additional significant variance.

M₁ through M₃ were not significant; their estimates T_{m_1} , T_{m_2} , and T_{m_3} were very close to zero. Only M₄ was significant ($T_{m_4} = 1.87$, $F=7.62$). This would suggest that rule four deletes an M that might better be left in the model. This tends to confirm the observation that users sometimes pause after committing themselves to a course of action [Allen and Scerbo, 1983]. An alternative explanation is that M₄'s are correlated with something else that is increasing the execution time.

This result is related to an interesting, more general problem known as the statistical significance of rule-based models. A rule-based model is a set of situation-action pairs that can be used to model human behavior, typically problem solving [Card et al., 1980a] [Rouse et al., 1980] [Hunt and Rouse, 1984] [Newell and Simon, 1972] [Anderson, 1983]. An open question is how to test the statistical significance of these rules. Testing should be done because there is a danger of an overfit model. For example, a rule-based model with as many rules as data points is clearly overfit. For some signifi-

cance testing solutions to this problem, see [Lewis and Hammer, 1985].

Summary

A regression model fit to the original data demonstrated the following. For the original Keystroke-Level Model, the operators K, P, D, M, and R are significant. The H operator is not. Executive systems were significantly overestimated, but there were no differences between editors. The M and D operators, while significant, do not add to the predictive power of the model. Of the four rules for deleting M operators, three appear to be correct but one appears to delete a delay that might exist.

STUDY TWO

The second part of this article uses the data from [Allen and Scerbo, 1983] to test some further hypotheses. The data came from an experiment that they conducted. Six users experienced with the ED text editor did 62 tasks from Roberts' benchmark [Roberts 1979]. Each task consisted of two subtasks: search and edit. During the search subtask, the editor was positioned to the line that was to be changed. During the edit subtask, the line was changed. Sometimes, a search did not occur because the editor was already positioned at or near the next change. For each subtask, they predicted the number of M and K operators that should occur. After collecting the data, they also identified the actual number of M and K operators used by each subject for each subtask.

The data used in this article were as follows. The search and edit subtask times were combined. The overall, error-free time, less the acquisition time, was used as the dependent variable. For some data sets, the error time was included in the response. The independent variables, or regressors, were either the predicted (rule-based) number of M and K or the observed (method-restricted) number of M and K.

The questions studied are presented on separate sections below. In brief, these questions are as follows. The first is the degree to which a linear model will fit this new set of data. The second question is whether an improved fit is obtained by using the method-restricted rather than rule-based number of operators. The third question is the effects on fit of including tasks with errors. The fourth question is on the fit of a linear model on

individual performance in comparison to group performance.

Linear Model Fit

The first question examined was the relative error in a linear model fit to these data. As noted earlier, the Keystroke-Level Model has under-predicted times on all uses of the model except on the data from which it was created. If regression is used to re-estimate all of the coefficients, this bias will disappear. What will remain, however, is the MS_E error term. This term, when divided by the average task time, can be regarded as a relative error.

The regression model is shown on line 1 of Table 4. The relative error is 38%, which is larger than the 22% observed by Card et al. It is not possible to compare these two relative errors with each other or with a regression on predicted and observed times from Roberts and Moran [1983]. The reasons for this are as follows. Card et al. had up to ten replications of a given task for each of four subjects. Up to forty data were averaged and then used for model building. Averaging reduces the noise and between-subject variance and hence the relative error. Allen and Scerbo had six subjects perform each task once and could only average across subjects. Roberts and Moran's relative error is not comparable because it is based on a long sequence of tasks, not on single tasks. The effects of averaging data are discussed in a later section.

A related question is the reasonableness of the estimated values for T_m and T_k . While both appear reasonable, the value for T_k of .26 is different from the typing test keying rate of .19 seconds/keystroke. The actual value

is well withing a 95% confidence interval of the estimated value ($.26 \pm .18$).

Linear Model Fit with Observed Operator Counts

The second question is what effect using method-restricted predictions has on the linear model fit. The results above are from rule-based predictions, which are predictions of the minimum M and K needed to do a task. It is also possible to make predictions from what a subject actually did (method-restricted predictions). Thus, it is an important question as to how much improvement there is with method-restricted predictions. Linear models are less attractive if method-restricted predictions are substantially better. To test this, the observed, or method-restricted number of M and K were regressed against the time. The results are shown in line 3 of Table 4. There is no improvement. The explanation for this is simple. The observed M and K are strongly correlated with the predicted (see Table 4, lines 4 and 5).

Linear Model Fit with Error Time Included

Question 3 was on the effects of including error time in the time to be predicted. This is important because errors do occur, and it might be desirable to compensate for them in overall predictions. On the other hand, if including errors causes large decreases in accuracy, removing errors may be worth the effort. Line 6 in Table 4 shows that including errors has only a modest impact on the relative error.

Model Fit to Individual Users

If a linear model approach is valid, it should be as suitable a model of individuals as it is of data averaged across individuals. To test this, six models were constructed of the six individual users. The regressors used were

the observed number of keystrokes and mental operators. Only tasks with no error time were used. The results were shown in Table 5. As can be seen, the linear model does not fit individuals as well as it does the average (Table 4, line 3). There are two reasons why averaged data is fit better than individual data. The first is that individual differences are removed. If this were the dominant phenomenon, it could be claimed that a linear model describes task aspects but not individual aspects of performance. The second phenomenon that occurs in averaged data is the reduction in random noise. Since there is no replication of tasks by the same subject in Allen and Scerbo's data, it is not possible to determine which of these phenomena dominates.

To investigate the sources of variance more fully, the individual performance data of the three text editors POET, SOS, and BRAVO on four tasks each was examined. These data are the individual replications before averaging from the experiment of Card et al. A random effects one-way ANOVA was used to compare between- and within-subjects variance. Within-subjects variance corresponds to noise (assuming there is no learning). Between-subjects difference corresponds to individual differences. The results are shown in Table 6. Nine of the twelve between-subjects differences are significant. Typing speed differences were not taken into account in this analysis, and they may be an important factor, especially in longer problems like T4. These results suggest that the Keystroke-Level Model may take advantage of the

averaging of both subject differences and noise.

CONCLUSION

Multiple linear regression has been used to evaluate the significance and power of individual terms of the Keystroke-Level Model. It would also seem appropriate to have used this method to develop the original model on the original, unaveraged data. Though the same model would result, R^2 would have been lower due to the presence of noise variance. It would also have been possible to test subject factors to account for individual differences. Replicated data also suggest a lack of fit test. This test compares the variance of the average to predicted value against the variance due to replication (noise and individual differences). The lack of fit test should not be significant for the regression model to be accepted. For the original Keystroke-Level Model, the lack of fit is significant ($F(31,854)=3.2, p<.001$). Removing the Executive tasks does not change this ($F(26,700)=2.7, p<.001$). When the lack of fit test is significant, the conclusion is that one or more terms are missing from the model.

Also demonstrated was the ability to build easily models of phenomena not previously examined. For example, including error time and tasks with non-prescribed methods was relatively easy for the data of Allen and Scerbo. It would also be possible to model data without a priori estimates of coefficients. Both of these practices could save analysts considerable time in model building. It would also be possible to investigate other kinds of delays, such as time delays before or after a command, delimiter, or keystroke, without making a detailed examination of the data.

Further improvement in the Keystroke-Level Model would seem to require a more exact way of predicting how the user deviates from the optimal sequence (would remove under-prediction) or predictions of mental operations that are not highly correlated with overt physical actions (would reduce relative error under a regression testing procedure). Both of these improvements would seem to be difficult undertakings. It would be relatively easy to describe the suboptimality, and relatively difficult to predict when they do and do not occur.

There is an alternative design approach that uses a description of a suboptimality. This approach is to design the suboptimality out of the system. By this it is meant that the interface does not require the user to take an action that is often done suboptimally. Consider the following extended example. Suppose the user wants to position the editor down to a line in the file. This line may or may not be on the display, but assume it is 25 lines away. The user does not know this distance exactly and chooses to have the next 20 lines displayed. This request is then repeated. Time is wasted while 15 lines are needlessly displayed.

It would seem to be difficult to predict the number of lines requested (e.g., 20), unless the user's habits dictated a request for 20 lines or the available command always displayed 20. If neither of these were true, a more precise model would require as a component a model of the user's understanding of the text and the distances within it.

To design a command to support this kind of positioning is difficult. Any fixed decision (e.g., offer a single keystroke command to display 20 lines) will not work well in every situation. Any command requiring as an argument the number of lines to display will be much slower to type. Neither

eliminates the user's suboptimality of not knowing the precise distance to the desired line.

To design the user's suboptimality out of the system is to offer commands that allow the user to edit optimally without perfect knowledge of the text. For this example, the command could simply start the display scrolling downward. When the editor reached the desired spot, the user could strike a key to stop it. Under this arrangement, single keystroke commands could be used, large amounts of excess text need not be displayed (which would cause the user to wait), the display runs at the maximum viewable speed, difficult design decisions (which are not globally optimum anyway) are avoided, and the user does not need to know how far away the desired line is.

While this example makes use of conventional terminal displays rather than bit-mapped workstation displays, the same general approach applies to much of human-computer interaction. For further information on how this approach was applied throughout an editor design, see [Hammer, 1984].

ACKNOWLEDGMENT

This research was supported by the National Science Foundation under Grant IST-82-17440.

REFERENCES

- Allen, R.B., and Scerbo, M.W., "Details of command language keystrokes," ACM Transactions on Office Information Systems, 1(2), 1983.
- Anderson, J.R., The Architecture of Cognition, Harvard: Cambridge, 1983.
- Box, G.E.P. and Wetz, J.M., "Criteria for judging adequacy of estimation by an approximating response function," T.R. 9, Dept. of Statistics, University of Wisconsin, Madison, Wisconsin, 1973.
- Card, S.K., Moran, T.P., and Newell, A., "Computer text editing: an information-processing analysis of a routine cognitive skill," Cognitive Psychology, 12(1), 1980a.
- Card, S.K., Moran, T.P., and Newell, A., "The Keystroke-Level Model for user performance time with interactive systems," Communications of the ACM, 23(7), 1980b.
- Card, S.K., Moran, T.P., and Newell, A., The Psychology of Human-Computer Interaction, Lawrence Erlbaum: Hillsdale, NJ, 1983.
- Draper, N.R., and Smith, H. Applied Regression Analysis, 2nd ed., Wiley: New York, 1981.
- Ellerton, R.R.W., "Is the regression equation adequate? - A generalization," Technometrics, 20(3), 1978.
- Hammer, J.M., "A display editor with random access and continuous control," International Journal of Man-Machine Studies, Vol. 21, pp. 203-212, 1984.
- Lewis, C.M. and Hammer, J.M., "Significance testing of rules in rule-based

models of human problem solving," to appear in IEEE Transactions on Systems, Man, and Cybernetics, 1985.

Hunt, R.M., and Rouse, W.B., "A fuzzy rule-based model of human problem solving," IEEE Transactions on Systems, Man, and Cybernetics, 14(1), 1984.

Kendall, M.G. and Yule, G.U., An Introduction to the Theory of Statistics, Charles Griffin: London, 1950.

Montgomery, D.C., and Peck, E.A., Introduction to Linear Regression Analysis, Wiley: New York, 1982.

Newell, A. and Simon, H. A., Human Problem Solving, Prentice-Hall: Englewood Cliffs, NJ, 1972.

Roberts, T.L. Evaluation of Computer Text Editors, Report SSL-79-9, Xerox Palo Alto Research Center, Palo Alto, CA, 1979.

Roberts, T.L., and Moran, T.P., "The evaluation of text editors: Methodology and empirical results," Communications of the ACM, 26(4), 1983.

Rouse, W.B., Rouse, S.H., and Pellegrino, S.J., "A rule-based model of human problem solving performance in fault diagnosis tasks," IEEE Transactions on Systems, Man, and Cybernetics, 10(7), 1980.

Suich, R. and Derringer, G.C., "Is the regression equation adequate -- one criterion," Technometrics, 19(2), 1977.

Symbol	Meaning
T_k	time per keystroke
N_k	number of keystrokes
T_h	time for moving hand to mouse from keyboard or vice versa
N_h	number of homes
T_p	time for pointing with a mouse
N_p	number of pointings
N_d	number of lines drawn
L_d	length of lines drawn
T_{N^d}	constant time to draw a line
T_{L^d}	time to draw a line that is proportional to length
T_m	time for a mental delay
N_m	number of mental delays
T_r	response time for computer

Table 1. Symbols for the Keystroke-Level Model.

Model	K	P	H	D	R	M	Exec.	MS _E	R ²
1	1.52 32.49	.92 50.84	1.25 .85	.87 14.44	.92 34.22	.68 10.43	-4.85 7.29	2.20	.979
2	1.54 33.64	.94 55.35		.85 14.06	.92 34.69	.67 10.37	-4.97 7.73	2.20	.979
3	1.13 20.98	.95 46.92		.92 13.40	.69 21.81	.95 21.25		2.45	.972
4	1.12 20.61	.94 42.90	1.50 1.00	.94 13.84	.69 22.00	.95 21.25		2.45	.973
5	2.33 437.23	1.00 49.84		.63 6.35	.87 23.23		-7.70 17.72	2.58	.970
6	2.33 367.87	1.13 59.44			.85 18.58		-7.55 14.36	2.78	.963

Table 2. Models, term significance, and performance measures.

The table is interpreted as follows. Each row is a model as determined by BMDP9R. If an entry contains numbers, then the operator at the top of the column is a term in the model. For example, in model two, K, P, D, R, M, and Executive are in the model but H is not. The two numbers are the coefficient (top) and the F-test value (bottom). The F value becomes significant ($p=.05$) at about $F=4$. The coefficient is expressed relative to the time used in the original model. For example, if the coefficient of P is .92, then the time for a P is .92 times the time for a P in the original model, which was 1.1 seconds. Also shown in the rightmost two columns are the error mean square and R^2 .

Operator	Contribution to R^2
K	.028
P	.046
D	.012
R	.029
M	.009
Executive	.006

Table 3. Unique contribution to R^2 by terms in model two.

No.	Model	R^2	MS_E	Average Task	Relative Error	Units
1	$2.10Mr + .26Kr$.91	6.24	16.6	38%	seconds
2	Keystroke-Level	.97	2.44	11.3	22%	seconds
3	$1.30Mm + .26Km$.91	6.21	16.6	37%	seconds
4	$Km = 1.31Kr$.95	6.26	25.2	25%	keystrokes
5	$Mm = 1.21Mr$.98	1.12	6.9	16%	mental operators
6	$2.92Mr + .24Kr$.88	8.98	20.9	43%	seconds

Table 4. Models of Allen and Scerbo's data.

Km (Mm) denotes the method-restricted number of keystrokes (mental operators). Kr (Mr) denotes the rule-based number of keystrokes (mental operators).

User	Model	R^2	MS_E	Average Time	Relative Error
1	1.1M+.08 <u>K</u>	.76	5.0	7.7	65%
2	2.5M+.13 <u>K</u>	.81	10.7	19.3	55%
3	2.4M-.03 <u>K</u>	.86	7.1	12.7	56%
4	.88 <u>M</u> +.51K	.80	8.6	15.2	57%
5	2.8M-.06 <u>K</u>	.69	11.8	15.0	79%
6	.03 <u>M</u> +.56K	.92	4.4	12.3	36%

Table 5. Models of Individual Users.

Underlined terms are not significant.

Editor	Task	F-ratio		Significance
POET	T1	F(3,27)	1.85	ns
	T2	F(3,19)	12.56	p<.001
	T3	F(3,21)	20.54	p<.001
	T4	F(3,16)	40.33	p<.001
SOS	T1	F(3,27)	4.12	p<.05
	T2	F(3,28)	5.08	p<.01
	T3	F(3,33)	16.43	p<.001
	T4	F(3,13)	25.44	p<.001
BRAVO	T1	F(3,28)	3.58	p<.05
	T2	F(3,28)	1.64	ns
	T3	F(3,34)	2.08	ns
	T4	F(3,29)	3.94	p<.05

Table 6. Significance of between- to within-subjects differences.

SIGNIFICANCE TESTING OF RULES IN RULE-BASED MODELS
OF HUMAN PROBLEM SOLVING

C. Michael Lewis

John M. Hammer

Center for Man-Machine Systems Research
Georgia Institute of Technology
Atlanta, Georgia 30332

INTRODUCTION

Many researchers have used rule-based systems to model human problem solving [1,3,6,7,11,12]. Typically, the rule-based system has a large number of rules, each of which has several free variables that were adjusted during the modeling process. For the most part, significance testing of these rules has not been much of a consideration. It should be. It is certainly possible to describe N data perfectly with N rules using a trivial model that simply reproduces the data. While there is no evidence that this has happened in any of the research reported to date, there is a certain danger of overfitting a rule-based model.

In this article we present three methods of testing the statistical significance of rules and other components of rule-based models. Throughout this article we shall assume that the percentage of behavior matched (e.g., commands) is the performance measure of interest. Two of the testing approaches, however, are not limited to this measure. They may be used to study any performance measure, though it may well be possible for a rule to produce a statistically significant effect on one performance measure but not another. The remainder of this article contains a section on notation, three sections on testing by analysis of variance, randomization, and contingency tables, respectively, and two concluding sections on applicability of the various tests and validity of these models.

NOTATION

A rule-based system consists of three components. The first is a set of rules of the form IF condition THEN action. The meaning of the rule is that if condition is true, then action could be taken. For example, the following rules describe behavior at a traffic light-controlled intersection:

IF In intersection	THEN proceed
IF Yellow and arrival at intersection before the light turns red	THEN proceed
IF Yellow and arrival at intersection after light turns red	THEN stop
IF Green	THEN proceed
IF Red	THEN stop
IF Red and right turn	THEN proceed

Figure 1. Rules for traffic lights.

If the above model can successfully match human behavior, then the rules form a model of the human. Often, the rules are interpreted as a model of the human's knowledge. Intuitively, the better the model matches human behavior, the better the model, all other things equal.

The rules can be transformed easily into a computer program as follows. First, control statements are added that cause the program to examine the rules repeatedly and execute those whose conditions are true. Second, in order to compare model and subject actions, an input statement is added before the first rule. This statement reads the state vector

(e.g., the lights, the traffic, short term memory) that was available to the human when his or her decision was made. The program looks something like this:

```
WHILE TRUE DO BEGIN
  READ(STATE);
  IF      (in intersection)                THEN proceed
  ELSEIF  (Yellow) AND (predict arrival at
            intersection before light turns red)  THEN proceed
  ELSEIF  (Yellow) AND (predict arrival at
            intersection after light turns red)   THEN stop
  ELSEIF  (Green)                          THEN proceed
  ELSEIF  (Red) AND (right turn)            THEN proceed
  ELSEIF  (Red)                            THEN stop
END;
```

Figure 2. Rules in a program.

The second component of a rule-based system is a conflict resolution strategy. It selects the rule to execute when multiple conditions are true. In the above example, a rank-order resolution strategy was shown. It simply uses the first rule that matches. The ranking of rules can then be interpreted as a subject's strategy. Some other conflict resolution strategies include random selection, meta-knowledge, and backtracking. A random selection strategy simply picks at random one of the many matching rules. A meta-knowledge strategy has a higher-level rule-based system that chooses which rule to execute. A backtracking strategy will, if necessary, try all possible matches. It should also be

noted that it may be possible to write the rule conditions so that there is always exactly one rule that matches.

The third component of a rule-based system is the input and internal variables. The input variables correspond to external data. The internal variables correspond to human short-term memory, which may be changed by the action part of rules. Both internal and input variables are examined by the condition part of rules.

Evaluation of Models

When comparing subject and model performance, the model is usually run open-loop without any knowledge on subject actions. In other words, the model can simply be treated as another subject. When comparing subject and model behavior, the model is usually run closed-loop as follows. The model has as input the same state vector the subject saw. The model chooses an action, and then it is recorded whether the subject and model agree. Then, the subject's action is used to control the system, and the process repeats. The reason for always following the subject's action is as follows. If the subject and model action differed and both were used, then the state vectors would be unequal after applying these actions. The model and the subject would then be working on different problems, and a comparison of their actions would make little sense.

The following sections on testing rule-based models will specify ways in which the model will be modified and then run. The typical modifications are to delete or modify one or more rules. Running a model, perhaps in a modified form, means to compare its overt behavior, (e.g., commands) to a subject's and determine the percentage in agreement.

ANALYSIS OF VARIANCE

The analysis of variance approach is the simplest of the three approaches for testing rule significance. To use it, each rule in the model is equated with an independent variable. The meaning of the variable is that at its high level, the rule is in the model, and at its low level, the rule is deleted from the model. The rule-based model is then run 2^N times (for each subject), which corresponds to a run with each possible subset of rules present. It must make sense for the model to do nothing, or else the model must be augmented before testing with a special, nondeletable rule that applies when no other rule applies. The resulting data can then be analyzed as an N-way factorial.

To economize on model runs, fractional factorial designs should be used. The full factorial design, proposed above, will estimate the effects of many high order interactions that cannot occur. In fact, the interpretation of an interaction is that the corresponding rules interact. An example would be two rules, the first of which stores some value in a temporary variable and the second of which uses the temporary variable. Such rule interaction is common, but rarely do many rules interact. An inspection of the rule-based model will reveal what interactions could occur. It should be possible to create experimental designs which test only the desired interactions.

The testing of condition components of rules is also possible. In this case the reduction in error attributable to the greater specificity provided by the additional condition can be evaluated. Suppose, for example, that a significance test of each of the conjunctive conditions of a rule is desired. For example,

IF condition₁ AND condition₂ AND condition₃ THEN

Proceeding as before, three independent variables might be equated, one with each of the three conditions. A three-way ANOVA could be run to test each of the three clauses. It would most likely be necessary to estimate the value of the response at the point where all three conditions have been deleted from the rule. Obviously, this process could be extended to cover all of the conditions for all of the rules in the model.

The testing of groups of rules as a whole is also possible. To do this, an independent variable is equated with several rules, not just one as was done initially. The experimental interpretation is that the entire set of rules is either present or absent from the model during an experimental run. This pooling of rules corresponds to a supersaturated experimental design, and may be the only economical means of testing models with many rules. One logical choice for pooled rules would be interacting rules. Another choice would be the modeler's organization of rules into groups (e.g., S-rules and T-rules [6]).

Analysis of variance makes several assumptions, one of which is that error residuals are normally distributed. Moderate departures from this assumption do not produce large deviations in calculated and actual significance levels. If the normality assumption is known or seriously thought to be incorrect, an approximate technique [4] may be used. Simply, the data are replaced with their ranks, and the remainder of the analysis of variance calculations remain unchanged. The significance levels produced by this method are reported to be nearly equal on normally distributed data to that produced by the standard F-test. The

rank transformation is more robust with respect to the distribution of the data, though it is not a distribution-free test. Finally, the hypothesis being tested here is whether the presence of a rule (or some other similar entity) explains a significant amount of variance in the subjects' performance. This significance is independent of the significance of other rules (or other entities) but may be dependent on the conflict resolution strategy. It is important to note the hypothesis because the next section tests somewhat different ones.

RANDOMIZATION

The second approach to testing a rule involves forming a randomization distribution by randomly permuting a rule. Suppose a particular rule is under test. Its action can be replaced by a random action (e.g., a random number generator that chooses commands according to a priori frequencies). The model, with a single modified rule, can be run many times. Its matching performances can be considered a randomization distribution. The model in its unaltered form can then be run, and its resulting performance be referred to the randomization distribution. If its matching were higher than 95% of the randomly generated values, the null hypothesis could be rejected at the 5% level (one-sided). The null hypothesis would be that a random action would be as suitable as the proposed action in the rule under test. The empirically determined significance level is partial in that it is potentially dependent on all the other rules being present in the model as well as conflict resolution strategy.

The condition part of a rule can be tested by a very similar method. There is a minor difficulty in that a random number generator in the condition part of a rule does not appear to make sense. A solution would seem to be to create various mutant conditions by randomly selecting condition clauses from other rules in the model. The null hypothesis being tested here is that random conditions are as suitable as the proposed condition in the rule under test. The significance level attained is partial just as the one obtained in testing actions.

An entire rank order conflict resolution strategy may also be tested by randomization. Basically, a randomization distribution of performances can be obtained by running all possible rank orderings (or a

Monte Carlo sample) of rules. The performance of the model with the original rank ordering can be referred to this distribution as above. The significance level obtained is dependent on the rules.

CONTINGENCY TABLES

Contingency tables are used to analyze nominal data. If the following is a rule-based model:

IF condition₁ THEN action₁
 IF condition₂ THEN action₂
 .
 .
 .
 IF condition_n THEN action_n

then, a contingency table may be set up as follows:

	action ₁	action ₂	action _n
condition ₁			. . .	
condition ₂				
.			.	
.			.	
.			.	
condition _n				
NOT (condition ₁ OR...OR condition _n)				

Figure 3. A contingency table for rules.

The last row in the table covers the conditions that are not covered by any rules. The observed data fill the table in the obvious way: for a given state vector and subject action, the unique condition which holds is determined, and the cell under the subject's action is incremented. A model that matched the data perfectly would have all zero entries off the diagonal.

Certain restrictions must be met to employ contingency tables:

1. Conditions must be mutually exclusive (2 rules cannot fire at the same time)
2. Actions must be overt
3. Each action must be unique (2 rules cannot issue the same action)

These restrictions may be met in a variety of ways. Mutual exclusivity will be satisfied by any model containing conflict resolution, rank-ordering, or disjoint rule provisions. The unique action requirement may be accommodated by phrasing composite rules in which constituent rules prescribing the same action are joined by disjunction.

The performance of the rules in matching the data can be evaluated with a chi-square or similar tests. The hypothesis is tested whether conditions and actions are independent, i.e., whether there is a significant difference between the proportions given the rules and the overall proportions. As a consequence, rules containing infrequently used actions receive more latitude using these tests than they do under a simple percentage of commands matched measure.

Testing a set of rules is also possible as follows. The null hypothesis is that there is no relationship between the action and the conditions aside from the relationship that is already described by the existing rules. Consider the test for the rule:

IF ($x_1 = 1$ or $x_1 = 2$) and ($x_2 = 1$) THEN action₁

		Action ₁	Action _n
X1 = 1	X2 = 1	Delete		
	X2 = 2			
X1 = 2	X2 = 1	Delete		
	X2 = 2			

Figure 4. Table for testing a set of rules.

Two statistics are computed. The first is a maximum likelihood estimate of chi-square, (G^2) for the complete table. The second is a test of quasi-independence [2] for a reduced table in which cells corresponding to rule(s) under test are excluded. This corresponds in a table such as figure 4 to one cell per row for conditions covered by the rule(s). If the original G^2 is significant and the quasi-independent one is not, this implies that the rules capture the dependency of the actions on the conditions. While attractive in directly referencing observables, this method requires large samples with replications of observed combinations of variables. (Unobserved combinations are treated as structural zeros.)

Other Statistics

A nonparametric analogue to the coefficient of determination R^2 is τ_b [8] which may be used to determine the percentage of variance explained in the actions by a rule or rule set.

$$\tau_b = \frac{\sum_i \frac{1}{X_{i+}} \sum_j X_{ij}^2 - \frac{1}{N} \sum_i X_{+j}^2}{N - \frac{1}{N} \sum_i X_{+j}^2}$$

X_{ij} = table entry in row i , column j

$X_{i+} = \sum_j X_{ij}$

$X_{+j} = \sum_i X_{ij}$

N = total number of observations

Individual rules, the disjunction of rules issuing a particular action, or the complete rule set consolidated into disjunctions by action can be evaluated in this way. If uncovered observations are excluded, τ_b may be interpreted as the extent to which actions covered by the rule are explained. If all observations are present, a $N+1$ st category should be formed following the distribution of the uncovered actions. This τ_b is interpretable as the extent to which rules explain all the actions.

Values of τ_b are asymptotically related to χ^2 allowing significance testing.

$$\chi_{(I-1)(J-1)}^2 = (N-1)(I-1)\tau_b$$

This statistic tests the hypothesis that $\tau_b = 0$, corresponding to the premise that there is no relation between conditions and the actions prescribed by the rule(s).

A similar statistic, PRE (proportional reduction in error) [2] measures the reduction in error achieved by predicting actions based on the rules rather than assigning the modal action under all rules.

$$PRE = \frac{\sum_{i=1}^I P_{im} - P_{+m}}{1 - P_{+m}}$$

where

$$\begin{aligned} P_{im} &= \max_j (P_{ij}) \\ P_{+m} &= \max_j (P_{+j}) \\ P_{ij} &= N_{ij}/N \end{aligned}$$

As demonstrated by this potpourri of procedures, a unified technique for testing rule significance based on multinomial sampling is yet to be developed. PRE answers the pragmatic question of gains in prediction. The quasi-independence procedure provides its complement by testing for unmodeled consistencies. Rules can be simultaneously tested in a contingency table but their contributions to rule set performance will remain unknown. τ_b allows both significance testing and estimation of effects but cannot evaluate rule set performance without pooling rules by action.

APPLICABILITY OF VARIOUS TESTING METHODS

For testing the degree to which a model's behavior matches a subject's, all three methods will work. A contingency table is clearly the best, however, since it requires the minimum in computation. Randomization is clearly the worst technique because of the large amount of computation and the partial significance levels it produces. A fractional factorial ANOVA is clearly superior to randomization on both of these points. ANOVA and randomization can both be used to test rules that modify internal, unobservable states. Contingency tables cannot.

For testing overall performance measures, (e.g., time to solution, total errors) only randomization and ANOVA are suitable, with ANOVA preferred. Ordinarily, much more emphasis is placed on behavior than on performance, since behavior is much more difficult to model. There are situations in which testing hypotheses about both performance and behavior is desirable. One might want to show that a certain set of rules will affect behavior but not performance. For example, Morris and Rouse [10] have observed that theoretical training given process control operators often fails to change their performance. It would be interesting to test this concept analytically in a rule-based system. For example, a group of rules might be identified as the intended consequences of theoretical training. The model might be run with and without these rules, using ANOVA to evaluate performance measures and contingency tables to evaluate behavioral differences.

The randomization method can be used on two hypotheses. The first, and more important, is to test the significance of a rank ordering of rules. This would seem to be the only way to test this type of resolution strategy. The second use is to test the hypothesis that part of a rule performs no better than random. This test would seem to be of little use, since ANOVA can test nearly the same hypothesis.

VALIDITY

The previous methods are generally devoted to evaluation of rule performance and do not address the issue of rule validity. Just as a high R^2 does not imply that all terms of its regression equation are significant, a high r_b does not vouchsafe for the future predictiveness of its rules. This distinction becomes important in the identification phase of rule-based modeling. Unlike identification based on parameter estimation, the identification of rules requires a search of the space of possible rules. An inductive pattern matcher must consider a large number of potential rules. In evaluating identification it becomes necessary to account for the probability of finding rules of comparable quality by chance. To answer this question the structure of the event space (observed combinations of condition variables), distribution of actions, and extent of search (set of possible rules) must be considered simultaneously.

Eilbert and Christensen [5] refer to this problem as contrivedness, "...the tendency of a search procedure to uncover apparent patterns where none exist." They suggest a randomization test for measuring the extent to which a search procedure uncovers contrived rules. The data consist of many pairs of state vectors with subject responses. The state vectors are left undisturbed, but the responses are randomly permuted. The resultant permuted data has reasonable state vectors paired with random responses. Contrivedness is the degree to which the search procedure can make sense of this random data. When many permuted data sets are searched, the search procedure results form a randomization distribution against which the results from the original, unpermuted data can be referred. While the previously mentioned randomization test will give an

idea of how opportunistic the search procedure is, it does not say how to refine the search procedure so as to avoid contriving rules.

CONCLUSIONS

This article has identified several ways of testing a rule-based model of human-problem solving. The amount of testing seems to be on a par with the size of the model. Left unresolved for the most part was the problem of contrivedness of automatic rule identification. It seems fitting to close with the description of an interesting and difficult question in identification of rules. As stated earlier, many cognitive models have been built using rule-based models. Sometimes these models are built when the investigator has access to the subject's thinking. This is always the case in developing a rule-based expert system. Other investigators, particularly those running experiments with humans, may have only the data (i.e., commands) to examine.

An important theoretical question is the limits to identification of rules from data that contain response errors. While there has been work in machine learning, it does not seem that anyone has examined this question [9]. It does seem important, because it bears on our ability to construct models. This problem also seems to be very difficult to solve formally. Hence, a preliminary investigation could be done via simulation, as shown in Figure 5. Basically, the approach is to generate some rules and some random stimuli, apply the rules, add noise, and try to identify the rules from the noisy data.

The following would seem to affect identification:

1. the amount of data and its coverage of the stimuli domain
2. size and number of rules
3. the number of times a rule fires
4. the level of noise

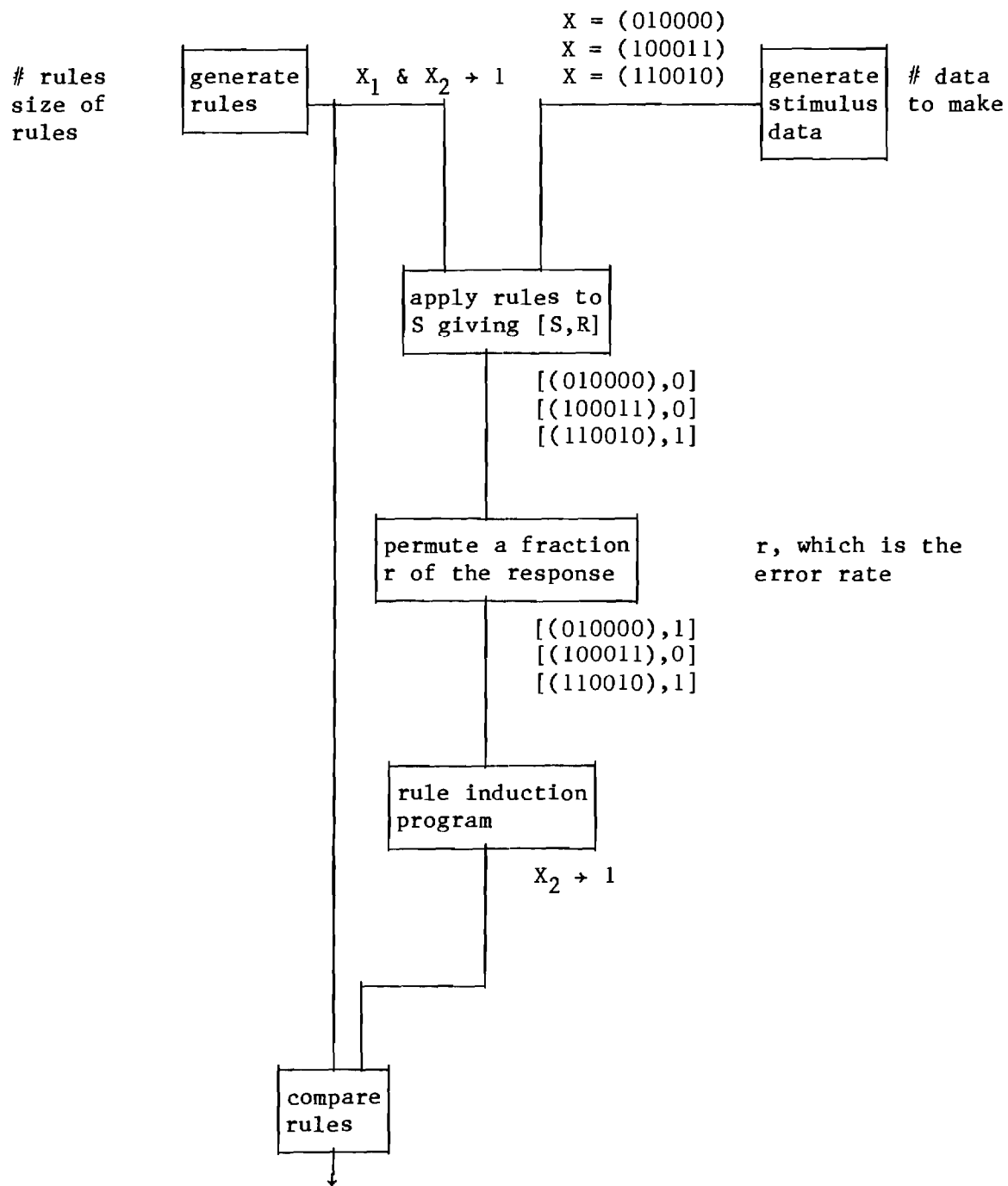


Figure 5. Block diagram for rule induction.
An example output from each block is shown.

It might also be interesting to investigate the addition of oracle variables in rule identification. An oracle variable is an extra variable (beyond the original stimulus vector) that provides information that ordinarily is not available. The first oracle variable might be a single bit to tell whether the response was in error. Another set of oracle variables would identify which rule fired. Yet another set of oracle variables could identify the variables that are part of the rule that fired. While these oracle variables may appear to be practically giving the solution to the identification program, they do not. These variables would be treated the same as any of the real stimulus variables. The identification program would have to infer the meaning of these variables in order to make use of them.

While it does appear theoretically interesting to determine how much oracle variables can add, there are important practical benefits as well. Oracle bits could approximate the hunches of a human investigator. For example, the investigator may suspect certain data to be in error, a certain rule to have fired, or that only certain variables could be influencing the operator's decision (from a verbal protocol). These hunches are a second order human-machine system: the investigator's attempt to identify (with a program) the rules of the human in the first-order human-machine system.

REFERENCES

- [1] Anderson, J.R., The architecture of cognition, Cambridge, MA: Harvard, 1983.
- [2] Bishop, Y.M.M., Fienberg, S.E., and Holland, P.W., Discrete Multivariate Analysis: Theory and Practice, Cambridge, MA: MIT, 1975.
- [3] Card, S.K., Moran, T.P., and Newell, A., "Computer text editing: An information processing analysis of a routine cognitive skill," Cognitive Psychology, Vol. 12, 1980.
- [4] Conover, W.J., Practical Nonparametric Statistics, 2nd ed., New York: Wiley, 1980.
- [5] Eilbert, R.F. and Christensen, R.A., "Contrivedness: The boundary between pattern recognition and numerology," Pattern Recognition, Vol. 15, No. 3, 1982.
- [6] Hunt, R.M. and Rouse, W.B., "A fuzzy rule-based model of human problem solving," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 14, No. 1, 1984.
- [7] Knaeuper, A. and Rouse, W.B., "A rule-based model of human problem solving behavior in dynamic environments," IEEE Transactions on Systems, Man, and Cybernetics, in press.
- [8] Light, R.J. and Margolin, B.H., "An analysis of variance for categorical data," Journal of the American Statistical Association, Vol. 66, 1971.
- [9] Michalski, R., Carbonell, J., and Mitchell, T. (eds.), Machine Learning, Palo Alto, CA: Tioga Publ., 1983.
- [10] Morris, N.M. and Rouse, W.B., "The effects of type of knowledge upon human problem solving in a process control task," IEEE Transactions on Systems, Man, and Cybernetics, in press.
- [11] Newell, A. and Simon, H.A., Human Problem Solving, Englewood Cliffs, NJ: Prentice-Hall, 1972.
- [12] Rouse, W.B., Rouse, S.H., and Pellegrino, S.J., "A rule-based model of human problem solving performance in fault diagnosis tasks," IEEE Transactions on System, Man, and Cybernetics, Vol. 10, No. 7, 1980.

AN INFORMATION-THEORETIC MODEL OF HUMAN SEARCH
STRING SELECTION IN TEXT EDITING

A THESIS

Presented to
The Faculty of the Division of Graduate Studies

By

Robert C. Andes, Jr.

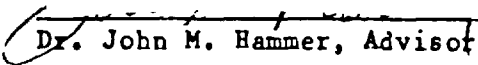
In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Industrial Engineering

Georgia Institute of Technology

September, 1987

AN INFORMATION THEORETIC
MODEL OF HUMAN SEARCH STRING SELECTION
IN TEXT EDITING

Approved:


Dr. John M. Hammer, Advisor

Dr. William B. Rouse

Dr. Russell G. Heikes

Date approved by Advisor Sept 20, 1987

ACKNOWLEDGEMENTS

I would like to thank my advisor, John Hammer, for all of the conceptual input, ideas, and endless editing advice provided in the completion of this thesis. Without his valuable guidance, this thesis would never have happened.

I would also like to thank Dr. Russ Heikes for statistical advice in the design of experiments and analysis of the data. A special thanks goes out to Richard Robison who provided guidance on the use of C for simulation and creative use of UNIX during simulation and collection of empirical data.

Finally, an especial thanks is extended to my spouse Susan for her love, patience, encouragement, and editing help during the entire thesis production process. Thank you, Susan.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	i
LIST OF TABLES	iv
LIST OF FIGURES	v
SUMMARY	vi
CHAPTER I: INTRODUCTION	1
CHAPTER II: A REVIEW OF THE LITERATURE	4
Types of models	5
Mathematical Models of Text	6
Word-Based Models of Text	13
Word Frequency Based Editing Strategies	14
Models of Human Performance in Text Editing	16
Important Variables to Current Research	25
CHAPTER III: A MODEL OF THE HUMAN SEARCH STRING SELECTION PRO- CESS	27
Model Introduction	27
Model Overview	28
Model Implementation	29
Analysis of Model Components	32
Model Input	32
Estimation of Scaling Factors	33
Summary	37
CHAPTER IV: AN EMPIRICAL STUDY OF SEARCH STRING SELECTION	39
Introduction	39
Text Searching Experiment	40
Independent Variables	40
Familiarity	40
Distance Information	41
Problem Type	41
Dependent Variables	42
Successes	42
Search Strings	42
Blocking Factors	43
Statistical Design	43
Subjects	46
Training	46
Procedure	47
Experimental Results and Analysis	49
Discussion	50

A Reanalysis of the Data	53
Frequency Estimation Experiment	54
Subjects	54
Procedure	54
Estimation Task Results	55
Discussion	56
CHAPTER V: COMPARATIVE ANALYSIS OF MODEL VERSUS HUMAN PERFOR-	
MANCE	58
Introduction	58
Performance Analysis	58
Search String Length Comparison	60
Discussion	61
CHAPTER VI: AN ALTERNATE MODEL OF SEARCH STRING SELECTION	64
Background On Alternate Model	64
Alternate Model Overview	65
Model Implementation	67
Analysis of Model Components	67
Text File Transition Matrix Process -	68
Computer Model of Text Editing Environment	69
The Simulation Process	69
Model Comparison	71
CHAPTER VII: CONCLUSIONS	73
Empirical Study	73
Models	75
Conclusions and Suggestions for Future Research	76
APPENDIX A	79
APPENDIX B	87
APPENDIX C	90
APPENDIX D	107
REFERENCES	113

LIST OF TABLES

Table		Page
3-1	Scaling factor regression results	35
4-1	Mean Number of Correct Responses By Familiarity and Distance Information Condition	50
4-2	ANOVA Results	50
4-3	Regression results of estimation task by familiarity type	55

LIST OF FIGURES

Figure		Page
2-1	Upper and Lower Bounds for the Entropy of English (from Shannon, 1951)	11
2-2	Text Searching Procedure Using the POET Editor	17
3-1	Two state Markov Process	29
4-1	Experimental Design and Linear Model	44
5-1	Mean Subject vs. Model Search String Lengths. Search Successes Under Familiar Condition	60
5-2	Mean Subject vs. Model Search String Lengths for Successful Searches Under Unfamiliar Condition	61

A empirical study is conducted and human behavior is compared to model performance, with accurate estimates of human performance obtained. An alternate model of search string selection is also given.

CHAPTER I

INTRODUCTION

The introduction of the word processor into computing and writing environments has changed the way man transposes his thoughts into the written text. The "word processor" allows the freedom to write more efficiently and to easily rethink and alter the ideas with a visual reference (e.g., either with a printed copy or the CRT display).

The primary environment for this thesis is the text editor. In this environment, the human frequently desires to change some part of the existing text. During this task the text editor becomes an indispensable tool, particularly in large files in which the desired text is not visible until the editor is positioned close to the text to be altered. There are several methods used to locate text in a large file (e.g., line-feed methods, screen by screen visual scanning, use of search commands).

Possibly the most efficient method of locating text is by the editor search command, or pattern scanning. The subject of this thesis is text searching behavior. Most modern editors search text by taking a character string, or sequence of letters, numbers, etc., as its argument. When invoked, it will position the cursor at the first character of the first occurrence of the specified character string in

the text file. The first occurrence of the string may not be the desired location, resulting in a search failure. The human will then have to reissue the command to look for the next occurrence, which can be time consuming. The efficient search, therefore, relies on the human's knowledge of the text content and searching strategy to quickly and accurately locate the text of interest.

Text searching within a text editor is a small part of document preparation. However, modeling of such a process is justified: Typing and document preparation comprise several billions of dollars of the nation's gross national product. By the same token, computer programs are written and maintained using the same text editors as those used in document preparation. Often the maintainer or editor of a large production software system is not the original programmer. Since the editor of a document is typically not the writer, it can be seen how the use of text editors can greatly enhance speed and efficiency of producing both documents and computer programs. By understanding the cognitive and probabilistic processes that underly text editing behavior, methods and tools can be developed to improve the text editing environment.

This thesis addresses the issue of how humans select the length of a search string in text editing tasks, particularly while editing computer program code. Editing computer programs introduces more complexity into the searching tasks due to the different syntactical nature of the text and the expanded character set used in programming.

As the length of the given search string increases, accuracy and keying costs increase. For the purpose of this research, it is assumed that a human will increase the search string length until the desired probability of success (e.g., finding the desired location on the first try) exceeds some threshold. A model is proposed that represents the human's estimated probability of search success in terms of a predefined threshold of information-theoretic bits. An increase in search string length is related to the value of the threshold. The probability of finding the desired text is modeled as a two-state Markov process. A more detailed explanation of the model is contained in Chapter 3.

CHAPTER II

A REVIEW OF THE LITERATURE

The literature pertinent to search string selection involves models of text and models of human text processing performance. Models of text are especially interesting from the digital computing viewpoint, since most text processing is now performed on the computer word processor. These models can be deterministic or probabilistic. Text processing studies typically analyze human performance within a specific context, in order to identify the underlying cognitive structures and processes responsible for the observed behavior.

The literature reviewed in this chapter will be used to identify the variables most pertinent to human performance modeling in text editing. Probabilistic models are given considerable attention in this review; the model developed in this thesis is probabilistically-based. Time and rule-based models of the human in text editing are also discussed, supplying important variables to the model. Finally, the issues of text familiarity and probabilistic knowledge are examined. It is important to note that there are two sets of variables used in this study. One set is the variables pertinent to the development of a search string selection model. These will be incorporated into the computer model. The second set is manipulated as independent factors in an empirical study.

Types of models

Analysis of human text creation behavior has influenced two major types of models: mathematical models of text and models of the human information processor.

Mathematical models have used statistics and conditional probabilities to predict the occurrences of letters and/or words in text environments. For example, an information theoretic model related to the entropy and redundancy of the English language was developed by Shannon (1951). Zipf (1949) has empirically shown that word usage frequency is directly related to word length and can be modeled in terms of probability. Ehrlich, Damon, and Cooper (1983) have demonstrated the utility of word frequency-based strategies in text editing.

In contrast, performance modeling is a technique used to model human information processing performance in text editing. Models of the human can be either time-based or rule-based in nature. Model accuracy is determined by comparing model and human performance. Time-based models estimate the amount of time required to execute a particular editing task (e.g., the length of time to complete the insertion of a word in a line of text). The research of Card, Moran, and Newell (1983a) focuses on operations conducted during human computer interaction. A common application of such a model would be used to estimate human performance within a newly developed editor, in lieu of actual experimentation.

Rule-based models of human text editing behavior describe the human's strategies in specific situations. For example, Card, Moran, and Newell (1983b) modeled selection of search methods when the distance to desired text varied from one to several lines of text.

Only a few accurate models of the human information processor in text editing tasks exist and these models vary in content.

Reviewed here are model types from two perspectives: those that model text itself and those that model human performance within the editing context. Although the nature of the models are quite different, important variables to the model of search string selection are extracted from both model types and incorporated into the thesis model. Additionally, the model variables extracted from the literature are also analyzed in a validation experiment.

Mathematical Models of Text

Mathematical models of text generation from letters, words, or phrases have been developed using stochastic process models and information theory (Shannon, 1951; Edmundsen, 1955; Barnard, 1955; Cover and King, 1978; Markov, 1913). For some of these models (e.g., Markov, Shannon), the underlying assumption is that text can be looked upon as the result of a stochastic process whose successive outcomes are constrained by semantics, reflected in the given probability distributions.

For example, the Markov chain (Markov, 1913) was developed to analyze the sequential organization of letters in terms of a stochastic process. The discrete process of sequential letter dependencies is described as state transitions, or the discrete Markov process. The general case can be described as: There exist a finite number of possible "states" of a system: S_1, S_2, \dots, S_n . In addition there is a set of transition probabilities; $p_i(j)$ the probability that if the system is in state S_i it will next go to state S_j . This theory is applied such that a letter is produced for each transition from one state to another. The states correspond to the "residue of influence" from the preceding letters. When state dependencies increase, the residue of influence increases by an associated conditional probability.

One especially relevant theory based on the Markov chain is Shannon's (1951) information theory analysis. Shannon's theory is based on the generation of text by providing successive approximations to English using Markov chains of higher and higher order. The Shannon study is the mathematical basis of this thesis.

Shannon proposed that a method of estimating the entropy and redundancy of a language (e.g., the English language) could be constructed from the knowledge of the statistics possessed by those who speak the language. Information Theory provided a consistent mathematical basis for investigations into encoded information transmitted over various media. With the introduction of digital computers to mass communication networks, a great deal of interest was expressed in digital encoding methodologies; Shannon's work was

dedicated to finding a suitable, economical way to encode the symbols of a language in this format.

Shannon's theory on the redundancy of a language is based on the relative entropy of that set of symbols. Entropy measures how much average information (i.e., reduction in uncertainty) is produced by each letter of a text in a printed language. If this language is then translated into bits, the entropy (H) of the language is the average number of bits per letter of the original language. The redundancy, on the other hand, measures the amount of potential loss of information in the language due to its syntactical structure and frequency statistics. For example, in English there is a strong tendency for "t" to be followed by "h", reducing the average information transmitted by each letter.

The general concept was that humans can be presented only partial information, and based on the accumulated knowledge of the statistics of the language used (e.g., syntactic and phonetic conventions), inferences can be made pertaining to the entire content of the information. For example, if shown the letters "roo", a human has some reasonable idea of the letter that immediately follows the second "o" (e.g., t, k, f, m, etc.). This inference structure could then be employed to allow for more economical data transmission. Conceptually stated, the knowledge possessed about a certain set of symbols in a constrained environment (e.g., a printed language) influences the human's decisions about the content and arrangement of these symbols.

The basic method for calculating the relative entropy and redundancy of a language is described as a series of approximations $F_0, F_1, F_2, \dots, F_n$ that account for successive statistics of the language and approach H as a limit (Markov chains). These approximations are made by using a successively conditional probability involving one more letter of preceding text. For example, a probability would be calculated for the number of times the letter "d" followed the trigram "wor", then a probability would be calculated for the number of times the letter "s" followed the 4-gram "word", etc. In Shannon's case, the approximations were made using an empirical estimator.

F_n , or "N-gram entropy" is commonly defined as the entropy calculation per letter for the Nth letter (based on conditional probabilities through (N-1) letters), where:

$$\begin{aligned} F_n &= - \sum_{i=0}^n p(b_i, j) \log_2 p_{b_i}(j) \quad (1) \\ &= - \sum_{i=0}^n p(b_i, j) \log_2 p(b_i, j) + \sum_{i=0}^n p(b_i) \log p(b_i) \end{aligned}$$

Where b_i is a block of (N-1) letters, and:

j is the Nth letter,

$p(b_i, j)$ is the probability of the Nth letter, j ,
with conditional probability dependency on the contents
of b_i .

The probability $p(b_i, j)$, or the probability of the N-gram (e.g., the number of times the letter string "abcd" (a "4-gram") occurs in a printed language) is determined empirically from the probabilities of the letter N-gram. The N-gram is estimated by extrapolation, or in Shannon's case, by estimation. The quantity $p_{b_i}(j)$ is also determined

empirically by extrapolating the conditional probability of the letter j after the preceding letter block b_i for the language of interest.

Shannon conducted a single subject study to test his theory. The subject was given a sequence of N letters from incomplete passages in a story, and probability tables for two and three letter combinations (known as digram and trigram tables, respectively). Shannon asked him to estimate the next letters in the story. If the subject guessed correctly, he was informed of the success. If the subject guessed incorrectly, he was prompted to guess until correct. This process was repeated until the passage was complete.

Of a total of 129 letters to be guessed, 89 (69%) were guessed correctly. Functionally, N was increased as the subject correctly guessed the next letters of the passage. The errors in prediction were most likely at the beginnings of words and syllables. Shannon explained that generally, good prediction would require no more than N letters of preceding text, and N could be small (e.g., 4 or 5 letters).

In another experiment, Shannon changed the experimental stimuli to one hundred samples of English text selected at random, each 15 letters in length. The subject was required to guess the text, letter by letter, for each sample as described in the previous experiment. In other words, one hundred samples were obtained with N varying from 0 to 14 preceding letters.

Shannon found that prediction gradually improved with increasing knowledge of past guesses, similar to the results of the first experiment. Shannon also calculated the relative number of bits of

information associated with a next letter in a sequence of letters (that will form a word) in terms of the letter's position in the string. Figure 2-1 depicts this relationship.

The above graph indicates a functional asymptote in the curve beyond five previously known letters. A significant conclusion can be drawn from this graph. It appears that the human acquires a fixed amount of information about the actual text after N becomes equal to, or greater than five or six characters in a string. This conclusion was exploited in the model developed in this thesis.

Cover and King (1978) extended Shannon's experimental findings for guessing the next letter in a sequence. They asked subjects to place successive bets (of real currency) on next letters in a sequence of letters forming words. The authors used the same basic experiment as

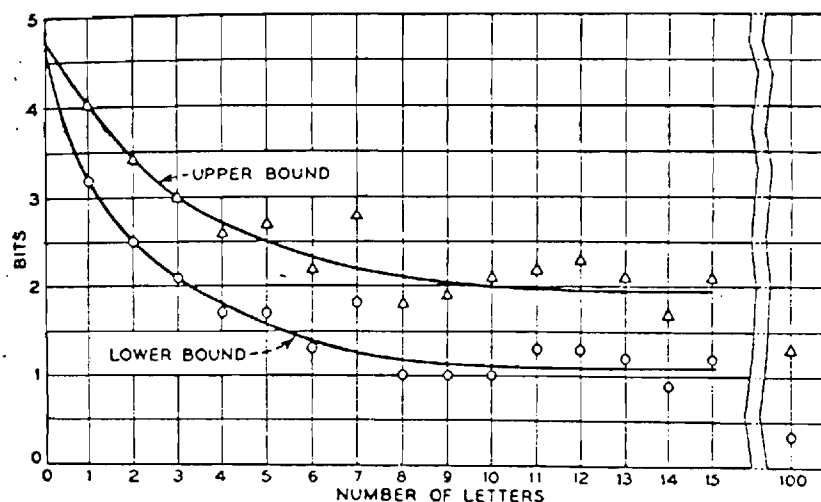


Figure 2-1 Upper and Lower Bounds for the Entropy of English (from Shannon, 1951).

Shannon; the subjects guessed the next possible letter until they guessed correctly. Additionally, Cover and King gave subjects letter digram and trigram probability tables to aid with probability estimation. Betting was included in the experiment to induce subjects to concentrate on accurate character estimation with a reward of even money given for a correct guess. Subjects were allowed to read as much of the passage as they desired up to the specified passage to familiarize themselves with the writer's style. Results showed that subjects were quite skilled in estimating next characters. This supports Shannon's claim that statistical knowledge of a language can be used to guess letters in a syntactically structured sequence.

Results of other language entropy studies (Barnard, 1955) indicate that Shannon's results can be extended to other languages. In Barnard's study, comparative figures for word-letter entropies were calculated for French, German, and Spanish. Barnard discovered that average word length bears directly on the letter entropy values for a given language. For example, all of the languages analyzed in Barnard's study reflected the same letter entropies as English, except for German. The average word length in German is longer than the others, primarily due to the fact that German words are a combination of other words to describe a complex concept. With the above results in mind, it can be extrapolated that similar languages (in syntax, etc.) would reflect similar entropy values.

sufficiently large to enable a more accurate modeling of a printed language in the form of zero-memory sources with words as symbols.

This correction effectively decreases the entropy per word estimate for all "English-like" languages to approximately 2.1 bits, instead of the original 2.62 bits per letter calculated by Shannon. Also, the bits/letter entropy is decreased to 2.1 bits vs. the original Shannon calculation of 3.3 bits. The correction must be considered to have had effect on the shape of Figure 2-1. This correction was also considered in the model construction.

Ellis and Hitchcock (1986) have produced Zipf functions for the UNIX operating system command language. These functions reflect the same results as the original analysis of English, further illuminating the flexibility of the law for printed language.

Word Frequency Based Editing Strategies

As previously shown, statistical knowledge of text can have a distinct effect on the subject's ability to fill in missing information. The use of word frequencies for developing a suitable text editing strategy was studied by Ehrlich, Damon, and Cooper (1983). Their study, which is the most closely related research to this thesis, analyzed the effects of knowledge on search strategy within a text editing task.

Four graduate students involved in preparing at least one large scale document (e.g., a master's thesis) served as subjects for the experiments. Working under the hypothesis that the subject's knowledge

of the text content and structure would affect search string selection strategy, they recorded subjective estimates of word frequencies in the subject's text file. Words were categorized according to type (e.g., nouns, verbs, etc.) and then the subject was asked to estimate how many times the subject thought the words occurred in the file.

Although the case studies produced varying profiles of word frequency, they do provide a consistent overall result. All of the subjects could estimate the frequency of occurrence of words, no matter what type, to within ± 2 occurrences of the lexical root. Additionally, subjects appeared to be biased toward lower estimates for words or word roots occurring more than ten times in the manuscript.

These results indicate the subject's familiarity with the text had a large bearing on the ability to estimate word frequencies until the frequency exceeds ten occurrences. Also, it seemed that the words most associated with frequency misjudgement also reflected "conceptually-based confusions." By this they meant that the conceptual encoding of the manuscript's topic, ideas, and main points by the subject were organizationally inaccurate. An example of this concept is editing a portion of a manuscript to convey a different main idea from what it previously had represented.

The authors concluded that the knowledge of the manuscript can assist the writer attempting to locate information during text editing. Use of this knowledge, however, must be tempered by an understanding of its limitations, namely, that surface details (e.g., actual syntactic and sentence structure) are not readily available. They also found that

low frequency (1 or 2 occurrences) words or labels have a much higher probability of being properly placed in position in the manuscript's hierarchy of ideas.

Models of Human Performance in Text Editing

Rule-based and temporal models of human performance behavior in a text editing environment have also been developed (Card, Moran, and Newell, 1983, 1983b; Kintsch and van Dijk, 1978). The models analyzed in this section provide an alternate method of modeling the text environment and human behavior in the text environment.

A model of computer user behavior, called the Keystroke Level Model (KLM) was developed as a design tool for estimating task execution times within different operating environments. The model's primary purpose is to supply the system designer with an estimated time required to accomplish a given task with a particular interactive computer system.

This model was tested within several text editing environments. The basic experiment consisted of several subjects editing manuscripts on different text editors. The authors developed a task execution model for specific editing tasks based on subtask time estimators. The total time to execute a task was the sum of the subtask execution times comprising the overall task. Mean subtask execution times were empirically determined for the following:

- K -- execute a keystroke
- P -- point to an object with a mouse
- H -- home hand on keyboard or mouse

D -- draw a line (with mouse)
 M -- think about next action (mental time estimator)
 R -- response time

where:

$$T_{\text{execute}} = T_K + T_P + T_H + T_D + T_M + T_R$$

The T_{execute} estimator is the sum of all the subtask times.

An experimental model developed using the KLM involved a search task. Subjects had to locate text, then edit a section or paragraph of the text. Using the editor "POET," the only editor similar to the one used in this thesis, the following subtask model was developed. The subtask representation for the search procedure is given in Figure 2-2.

The searching task described by Figure 2-2 can be interpreted as follows. The user thinks about how to initiate a search command (M), then issues the editor command to initialize the search by keystroking (K) the symbol ("). Then 7 characters are entered that represent the search string (7K[string]). The search string is terminated (") and a request is given to print the desired line (/). An edit command is then issued to begin editing the text. From this model, the user's

Indicate search string	--> MK["]
Type search string	--> 7K[string]
Terminate search string	--> MK["]
Print line	--> K[/]
Issue edit command	--> M 2K[e <CR>]

Figure 2-2 Text searching procedure
 using the POET text editor.

keystrokes and time for execute a text search are determined.

The model overtly describes how the task is accomplished; however, it has no provisions for estimating mental operations involved in the task except for the M operator. A time estimate for the M operator was determined for the search task model by subtracting the overt actions in a subtask from the subtask execution time containing the M operator. This estimator is sufficient for task execution time approximations, but appeared more coarse-grained than that required for the mental operations pertinent to this thesis. For example, how the subject generated the search string is not specified.

More importantly, the model does not say anything significant about the actual cognitive effort or strategy involved in generating a searching strategy and issuing an accurate search string. Additionally, the search string issued was not analyzed or recorded. The model does, however, represent the time -- including the search string selection process -- involved in text searching.

A rule-based model of human-computer interaction in various tasks was also developed by Card, Moran, and Newell (1983b). The GOMS model (for Goals-Operators-Methods-Selection) was developed to predict human computer interaction sequences through the analysis of intended goals during the task.

The GOMS model breaks a task into subtasks by supplying:

- A set of Goals,
- A set of Operators used to achieve the goal,
- A set of Methods used to achieve goals,
- A set of Selection rules for choosing among

competing methods for goals.

Using the above constructs, subgoalings was used to decompose the overall search-replace task to a finer grained level. The part of the model concerned with editor positioning is:

```
Goal: Locate-line
      [Use-LF-method]
      [Use-QS-method]
```

The strategies identified for locating specific text in the file were: the query-search method (QS-method) and the line-feed (LF-method) method. The human issues a search command with fixed length (5 characters) character strings to locate text using the query-search method. The line-feed method is used primarily for locating text to edit when the text is close to the present location.

A typical rule example for a subject could be represented as:

```
IF the distance to the next modification is less than
or equal to 3 cm., THEN use the LF-method.
ELSE use the QS-method.
```

Rules similar to the example are used by the model to predict human behavior based on the task requirements (e.g., the number of lines to the target text).

Card, Moran, and Newell demonstrated a subject's selection of search methods systematically on features of the task environment. The most important characteristic of the environment is the distance (in number of lines of text) between the current line and desired line of text. All subjects used the LF-method when the text was close enough. Close enough was defined differently for different subjects. In their experiment, the threshold generally differed depending on the type of

terminal used (e.g., a teletype was used in one condition; the subject switched to QS-method sooner due to slow response time of the terminal). Once the threshold could be established for a particular subject, modeling of the behavior was consistent over all subjects.

Subjects reported that the QS-method (search) appeared to be more difficult than the line-feed method, which was reflected in the slightly longer times to generate search strings vs. line-feeding. In general, the QS-method was preferred over the LF-method in experimental settings when the desired text was off-screen.

Although GOMS provides a finer-grained analysis of the strategy chosen to search for text based on distance than the KLM, it nonetheless provides no insight to the actual processes used by the human in the course of generating the search strategy and search token. GOMS predicts (with some accuracy) when a human chooses to search for text; it gives no indication of how many characters should be used or how a strategy will develop for text searching. Additionally, the subjects were instructed to limit their search strings to 5 characters, in spite of the distance to be traversed to the target text or the frequency of string occurrence in the file being edited. In the study, a search string of 5 characters was sufficient to position the editor at the desired location on the first try. In a less constrained text editing environment, however, 5 character search string limits would result in a higher rate of search failures. Also, the GOMS model was developed primarily for environments containing only a small amount of text over short distances.

Hammer and Rouse (1982) developed a model of the human as a constrained optimal editor. In their paper, investigations were conducted into the procedures and strategies used during intraline text editing. There were two phases to the study. The first modeled optimal keystroke solutions for intraline editing problems using several editor command sets. Optimal keystroke solutions were then compared with those generated by human subjects on the same editing problems. The second incorporated those factors determined to affect suboptimal human performance into a constrained model of optimal keystroke solutions. Several text editors were modeled.

The modeling of intraline editing keystroke solutions is as follows. The editing solution to the problem is determined with a typical problem characterized below. The user must alter the line of text "John and Mary" to be "John or Mary". To execute the task the user must: a.) reuse "John ", b.) delete "and", c.) insert "or", d.) reuse " Mary". Once the execution sequence is determined, the model takes the two lines of text (original and altered) and attempts to make the altered out of the original using all known editing commands from the particular editor of interest.

Subject performance was compared with optimal solutions produced by the model. It was found that the subjects were often suboptimal in their editing sequences due to: a.) limited knowledge of the possible editing commands available, b.) use of extra keystrokes to execute the same task, or c.) the use of estimation when the counting of characters was necessary (e.g., issuing an approximate number of characters to move

instead of actually precounting the characters to move).

The second part of the Hammer and Rouse (1982) study constrained the model to resemble more accurately subject performance within the specified editing tasks. Constraints imposed on the model were those found to be pertinent to user editing performance from the first phase of the study. Individually computed command sets for each subject in the experiment were input to the model to simulate a subject's limited knowledge of available commands. Also, the model was constrained to estimate large distances in a way consistent with observations.

Descriptive categories were also input to the model to describe keystrokes in excess of the constrained optimal solution. Several categories were identified, from open-loop cursor positioning to human error. However, only the command selection category is of interest to this thesis. It was found that subjects often conducted text searches under a "search string too long" situation, where the search string is longer than necessary to do the desired positioning.

It was found that the model was able to describe or predict 90 percent of all keystrokes. The Hammer and Rouse (1982) model was concerned only with intraline editing; however, subjects' tendency toward suboptimality in text searches indicates a consistent underlying mechanism in selecting search string lengths.

A study that bears mention in the context of the present research is a model of text comprehension developed by Kintsch and van Dijk (1978). They have shown that language users can provide missing links in a word sequence based on their contextual knowledge of the facts

within the text. The facts can allow humans to make inferences based on the fuzzy probabilistic relationships between the frequency of words and the text context.

Kintsch and van Dijk modeled this relationship using a propositional notation scheme to reflect these context effects within the text. The idea behind the propositional notation is to represent the meaning of text with a structured list of propositions. Propositions are composed of concepts. Basically, a concept is represented as a decomposition of an idea into the operators and actions producing the idea. For example, the sentence: "The professor decided to do an experiment" would yield the proposition: (do, professor, experiment). Propositions are ordered in the text base according to the way in which they are expressed in the text. This representation scheme allows the experimenter to convert the entire gist of a story into a format easily input to a computer.

Although the discussion of the model of text comprehension did not include operational characteristics or code, Kintsch and van Dijk described the basic input/output mechanisms of the model. The model takes three inputs: The first is S, the short-term memory capacity. The short term memory has been shown to be affected by reading skill; good readers are capable of holding more text in short-term memory than poor readers. The second input is N, the number of text propositions accepted per mental processing cycle. To justify this input, the authors theorize that the reader's knowledge of the text affects the meaning derived from the text. Unfamiliar material would have to be

processed in smaller chunks than familiar material, and N should be directly related to familiarity (e.g., N increases with knowledge of the text). The final input to the model is P , the reproduction probability of a particular proposition. The probability P is called the reproduction probability because it combines both storage and retrieval information. Under the same comprehension conditions, "the value of P may vary depending on whether the subject's task is recall or summation." (Kintsch and van Dijk, 1978) A proposition is reproduced with probability P each time it has participated in a processing cycle. Basically, if a proposition is selected $N-1$ times to be included in the short-term memory, S , it has N chances of being stored in long-term memory. More specifically, the reproduction probability is $1 - (1-p)^N$. This probability is similar to the Markov process discussed earlier. The authors also indicated P was directly related to familiarity.

It was found by Kintsch and van Dijk that the reader's comprehension of printed text varied significantly with familiarity of the experimental text. This finding was consistent whether the text was familiar to the subject prior to the experiment or the familiarity was increased artificially by having the subjects re-read the material and interpret the meaning of certain propositions. Although certain words were not remembered after a period of up to three months, the extent of the "gist", or general idea of the text was remembered according to level of familiarity with the material before the actual test. Model results confirmed this finding, indicating familiarity as an important quantity to text processing. According to the representation for P , the

familiarity induced by inclusion of a proposition in long-term memory also increases the reproduction probability of the proposition.

Important Variables to Current Research

To properly model the human in an experimental text searching task, insight must be gained into the human mental processes contributing to search string selection.

The method chosen for the model of search string selection presented in this thesis is based on the mathematical models of text, specifically Shannon's information theory. The rule-based, temporal, and cognitive models reviewed provide additional variables identified that affect search string selection.

Based on the literature, four variables will be considered within an information theoretic model of search string selection:

- 1.) The bits of information per character, or B_c , in a text file will be calculated for each experimental environment. B_c will be utilized in numerically determining a character's contribution to the total information theoretic bits for the search string.
- 2.) The text environment content and the human's level of familiarity (structural and propositional) with the text will also be considered. This factor is derived from the Kintsch and van Dijk study. Familiarity also be included in the empirical study.
- 3.) The user's estimate of search success based on knowledge of file

contents will also be considered. Inclusion of this variable is consistent with Ehrlich, Damon, and Cooper (1983). Consistent with this study, knowledge of particular file contents will be analyzed with the variable Problem Type.

4.) The character distance from current to desired cursor location in the text editor will also be considered. The variable distance is transformed into distance information cues within the empirical study. This variable is supplied by the GOMS model research of Card, Moran, and Newell (1983b).

Other factors influencing the human's ability to locate strings in a file may consist of knowledge of text structure (e.g., a particular programming language structure), text editing experience, and general knowledge of the language syntax. These qualitative factors will also be considered during model evaluation. In the following chapter, the model is developed from the variables of interest.

CHAPTER III

A MODEL OF THE HUMAN SEARCH STRING SELECTION PROCESS

This thesis studied human search string selection with respect to the human's knowledge of the language in the file being edited. The literature review suggested variables that may affect the process of search string selection. These variables were incorporated into the model and were manipulated in the Chapter 4 validation experiment to verify the predictive accuracy of the model.

The present chapter is divided into two sections: model design and model implementation. The level of implementation detail increases as the chapter progresses.

Model Introduction

The literature discussed in Chapter 2 indicated a human's statistical knowledge of a printed language has a significant effect on the ability to predict the next character in a sequence. Additionally, the human's memory of specific groups of characters previously used (e.g., words) has been shown to effect the strategy used in locating specific text in a text editing environment. Preliminary results of a

study conducted by Hammer (1984) indicate that the human does indeed possess significant knowledge of programming and syntactical structure, as well as character relationships within programming environments.

If the theories reviewed here are universal, then statistical modeling of human knowledge of a programming language should reflect the same basic constructs as with natural languages. Based on the literature and experimentation, a computer model simulating human statistical estimation of text within an editing environment was developed.

Model Overview

The model developed for the thesis (written in LISP, see Appendix A), chooses search string lengths for search problems simulating human performance in a probabilistic text searching task.

The model's task is to choose the length of the search string such that the probability of reaching the desired point exceeds a user-chosen threshold. Generally, the model's probability of success increases with search string length. If the model does possess perfect knowledge of the text, an optimal (shortest) string exists which will cause the editor's search to move to the desired point on the first attempt. The model does not possess perfect knowledge; instead, it adopts a probabilistic view of the text. In each search problem, the model chooses a search string with a sufficient number of information theoretic bits to cross the distance to the desired point in the file.

Model Implementation

Modeling human search string selection behavior is accomplished in distinct stages. First, the threshold number of bits for each problem is calculated as follows. The editor's search process is modeled as a two-state Markov process: the failure state is absorbing (F) and corresponds to the search failing. The success state (S) is the initial state. It possesses a low probability P_1 transition to the failure state and a high probability transition $(1-P_1)$ to itself.

The Markov process is used to estimate P_1 as follows: After a number of transitions equal to the distance from the current to the desired point, the probability of being in the absorbing state is $(1 - \text{the human's desired probability of success, or } P_g)$. Essentially, the Markov process models the small probability P_1 of finding the search key at every point in the text between the current and desired point. The occurrence of the search key before the desired point corresponds to the P_1 probability, or search failure. Once in the absorbing state, the

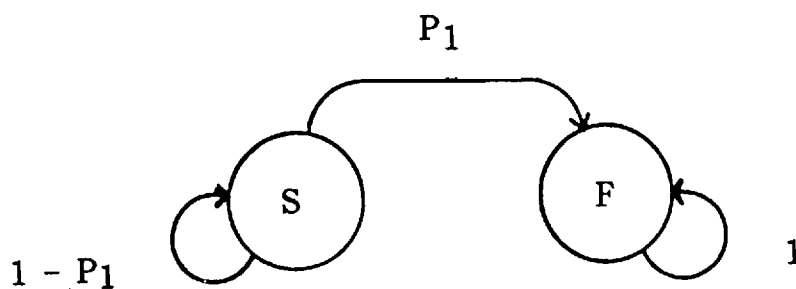


Figure 3-1

Two state Markov Process.

probability of the search failing is 1.

The Markov process emulates the editor's search process probabilistically. When the search command is issued, the editor initiates a character by character matching process between the search key and the text being edited. The process will continue until either:

- a.) the search key is found before the desired distance is traversed,
- b.) the editor reaches the end of the file, or c.) the editor reaches the character distance indicated. In either condition a.) or condition b.), the search is viewed as a failure. Condition c.) denotes search success, or the traversal of the desired character distance where the desired search key is known to reside. As long as the matching process does not find a pattern match for the search key before the desired distance, the Markov process continues to loop onto the success state (S) and the search continues. Should the editor match the search key before the indicated character distance, the Markov process traverses the P_1 arc to the failure state (F). Once the Markov process reaches the failure state, F, it does not exit F, and the search fails.

The inputs to this section of the model are the number of transitions D (for distance) and the overall probability of success P_g . The output is the single step probability P_1 . P_1 is then converted to the threshold number of bits, H_{D,P_g} . H_{D,P_g} can be interpreted as the number of bits required to cross over the necessary distance to the desired point.

Once the number of threshold bits is determined, the model selects the shortest search string with bits greater than or equal to the required threshold. The total bits in the search string are equal to the sum of the bits for each character (e.g., "a", "b", etc.), multiplied by a scaling factor. This factor is based on the position of each character in the search string.

The scaling factor was suggested by Shannon's (1951) observations that the bits per character associated with the next character in a sequence, given N preceding characters, is a function of the position in the string. The scaling factor is applied such that after the first character in the search string, it is expected that each additional character contributes a decreasing amount of bits to the search string bits total.

Or,

$$H(c,i) = H(c) * S(i)$$

where,

$H(c,i)$ = bits for character c in the i th position

$H(c)$ = bits for a character

= H (the number of occurrences of character c /
the total characters in the file)

and,

$S(i)$ = the scaling factor, such that:

$S(1) = 1$ and,

$S(K)$ = regression estimator for $K = 2, \dots, N$,

$S(K+1) = S(K)$ for $K > \text{some } N$,

$N \approx 5,6$

Analysis of Model Components

The model description has provided an overview of the concepts and techniques incorporated into the model. The sections under the present topic describe the model components in greater detail.

Model Input

The model receives four inputs:

1. The potential search string, from which the model selects a substring,
2. The character distance from the beginning of the file to the desired position,
3. The individual character frequencies from the file being searched,
4. An array of scaling factors.

The potential search string consists of characters remaining on the text line following the desired cursor position.

Individual character frequencies are calculated as number of occurrences divided by the total number of characters in the file.

The scaling factors are a series of values that reduce bits of information contributed in each position as the search string length is extended. For example, the number of bits contributed to the search string by the first character is a product of the first scaling factor and the bits value associated with the chosen character (e.g., the probability of the character in the file converted to a bit value (H)). The second character of the search string is the product of the second

scaling factor and its bit value, and so on.

Estimation of Scaling Factors

Initially, it was intended the bits of information versus character position graph from Shannon's (1951) study of the English language would provide adequate scaling factors for the model. However, estimators to the Shannon curve and mathematical approximations of the curve yielded incorrect results during pilot studies. Two reasons contributing to model failure were identified.

A reexamination of the pertinent studies in the literature review (e.g., Yavuz, 1974) and empirical analysis using the Shannon curves revealed that the bits of information associated with the early (e.g., second character, third character, etc.) characters in the selected search string were too high for the search string model. As a result, model search strings were consistently too short. Additionally, the text used in the original studies was based on a natural language. This thesis uses a programming language character set which expands the character set considerably. Therefore, an estimator for the scaling factors was required.

The original Shannon curve plotted $H(c,i)$, or bits per character, versus position in the character string. The Shannon curve shape was used as a model for the search string scaling factor values. Beginning with $S(1) = 1$, a series of scaling factors (i.e., S_2, S_3, \dots) were determined that would reflect the Shannon curve with an asymptote at the

fifth or sixth character position. The asymptote was approximated according to the asymptote identified in Shannon's original analysis of the English language.

The appropriate scaling factors were determined through regression analysis according to the equation:

$$H_{D,P_s} = S(1)*B_c[1] + S_2*B_c[2] + \dots + S_5 \sum_{i=5}^n B_c$$

where:

H_{D,P_s} = number of bits required to travel
the distance to the desired position

S_i = scaling factor for ith position

B_c = bits value for the particular character

$[f, \dots, n]$ = position of character in search string

n = length of string

The two-stage Markov process was employed to calculate the transition probability (P_1) for each searching problem. P_1 was then converted to H_{D,P_s} and used as the response variable in the regression analysis.

Character frequency values for the first five characters in each potential string were converted to bits (B_c) and used as regressors. The two sides of the equation were assumed equal according to the definition of H_{D,P_s} . The threshold value must be enough to traverse the necessary distance D to reach the desired point in the file. The equation term, $\sum_{i=5}^n B_c$ represents the bits of information contributed to the search string beyond four characters. Only the first five values had to be estimated, since $S(5) = S(5+i)$ for all $i \geq 0$. Basically, all B_c for $i \geq 5$ are summed and multiplied by the S_5 estimator. In each series of scaling factors, the first position (S_1) is always 1

since the first character of the string contributed the entire B_c value to the search string. Therefore, only four coefficients need be estimated by regression.

Linear regressions were run on the two files used in the validation experiment using the MINITAB Statistical Package. Regression results and calculated coefficients are found in Table 3-1. The variables are named $B_c[2]$ through $B_c[5]$ to denote position in the scaling factor array. The regression was run with no intercept in the equation.

Regression results for familiar file:

The regression equation is:

$$H_{P,Ds} = 0.131 B_c[2] + 0.249 B_c[3] + 0.149 B_c[4] + 0.162 B_c[5]$$

PREDICTOR	COEF	STDEV	T-RATIO
-----------	------	-------	---------

No constant

$B_c[2]$	0.1313	0.1798	0.73
----------	--------	--------	------

$B_c[3]$	0.2487	0.1356	1.83
----------	--------	--------	------

$B_c[4]$	0.1495	0.1490	1.00
----------	--------	--------	------

$B_c[5]$	0.16228	0.08718	1.86
----------	---------	---------	------

standard deviation = 1.264

SOURCE	DF	SS	MS
Regression	4	758.39	189.60
Error	46	73.44	1.60
Total	50	831.84	

Table 3-1 Scaling Factor Regression Results.

[2,...,5] denotes position in string.

Regression results for the unfamiliar file:

The regression equation is:

$$H_{P,Ds} = 0.086 B_c[2] + 0.427 B_c[3] + 0.267 B_c[4] - 0.084 B_c[5]$$

PREDICTOR	COEF	STDEV	T-RATIO
No constant			
$B_c[2]$	0.0864	0.1471	0.59
$B_c[3]$	0.4270	0.1627	2.62
$B_c[4]$	0.2670	0.1399	1.91
$B_c[5]$	-0.0838	0.1376	-0.61

standard deviation = 1.264

SOURCE	DF	SS	MS
Regression	4	882.72	220.68
Error	46	73.44	1.60
Total	50	956.16	

Table 3-1 Scaling Factor Regression Results
(continued)

A discussion of the regression results is required. All coefficients were considered valid until two successive values were non-significant. In the familiar file regression, this value was at 6; the value was 4 for the unfamiliar environment. At this point, it was assumed that the curve had reached its asymptote. The coefficient at the asymptotic position was used to fill out remaining positions in the array. In terms of the t-test, regression coefficients were included, provided they are significant. The B_2 regressor was always non-significant. However, there is an explanation for this observation. Limited information is added to the character string by the second character. This is reflected in the observation that a 2-gram, or a sequential probability of 2 characters, does not generally possess much information. With the addition of a third character, the information level increases significantly. As a result of this reasoning, the B_2 regressor was included in the scaling coefficients.

During the simulation, the total bit value of a character in the search string, $H(c,i)$, was calculated according to equation (1) using the character frequency and the i th scaling factor. If the search string chosen by the model was longer than 5 characters, the bits for the additional characters were added to the bits of the fifth regressor.

Summary

In this chapter, a model of human search string selection was developed around the user-defined bits threshold, H_{D,P_s} , for a search string based on the user's predefined probability of success, P_s . All variables of interest to search string selection identified in the literature review were included. A series of character position coefficients for $S(i)$ within a programming language was estimated through linear regression, modeled after the $H(c,i)$ curve suggested by Shannon (1951).

CHAPTER IV

AN EMPIRICAL STUDY OF SEARCH STRING SELECTION

An empirical study was conducted to evaluate the effects of text familiarity, distance information, and problem type on user search string selection behavior. In addition to the performance data collected, search strings issued by subjects were used to evaluate model performance, which is described in Chapter 5.

Introduction

Two separate experiments were conducted in this validation study. The first was a text searching experiment within a text editing environment. The second experiment was an on-line string frequency estimation task designed to elicit frequency estimates of particular character strings in each file. This data was used as a general estimate of a subject's knowledge of the file contents.

The chapter begins with identification, description, and justification of the variables used in the searching experiment. Experimental and statistical design is reviewed next, and then the actual experimental procedure is discussed. Following the results and

discussion of the searching experiment, the estimation experiment is reviewed in a similar manner.

Text Searching Experiment

Independent Variables

Based on current literature and experimenter hypotheses about human text searching behavior, variables were selected for the text searching experiment. Each variable is discussed in a subsection. The operational definition, rationale for inclusion, and types of stimuli within each variable range are included in each description.

Familiarity. (F) - Familiarity is operationally defined as whether the subject had experience with the contents of the file being searched. Following Kintsch and van Dijk (1978), the subject should have approximate knowledge of the frequency of characters and location of particular strings within the file. This familiarity with the text file should have a plausible effect on search string selection.

Two FORTRAN 77 source code files were used: one was the file that subjects had studied before the second experimental session (familiar environment); the second was a similar (in structure and length) FORTRAN 77 file that subjects had never seen before (unfamiliar environment). All subjects studied the same familiar program text. The training method

used to induce studying is described later.

Familiarity is the major variable of interest in this experiment. The user's success rate, as defined by the number of target strings found on the first searching attempt, should be greater in the familiar condition than the unfamiliar condition. The higher success rate is primarily due to level of familiarity with the text.

Distance Information. (D) - After Card, Moran, and Newell (1983), distance information cues about the searching target location were supplied to subjects in two forms: (1) the page number of the printed text that the target (desired text to be located) could be found, and (2) the line number, measured from the top of the file, on which the target could be found. Distance to the desired editor position would seem to be an obvious influence on string selection since the intervening text between the current cursor location and the target decreases the probability of locating the target. Distance was also one of the inputs to the model, further identifying it as an important factor in this experiment.

Problem Type. (T) - Problem type is defined as either familiar or unfamiliar targets to be located by the user. Ehrlich, Damon, and Cooper have established that a user's knowledge of word frequencies (or character strings) in a file is very accurate as long as the frequency did not exceed 10.

In the experiment, subjects searched for two types of targets. The basic content of the target was either: familiar (e.g., a subroutine

name, language keyword, etc.); or unfamiliar (e.g., variable name, parameter of a subroutine, etc.). The subject was expected to recognize when a target was familiar. This recognition should also have influenced the search string length.

Targets were evenly distributed in each text file to ensure target stimuli equivalence. Basically, three distance ranges were used: from 100 to 9000 characters, from 9500 to 20,000 characters, and 20,000 to 30,000 characters. Both text files contained approximately 31,000 characters. The stimuli, equally divided among the three distance ranges, were also equally distributed (as closely as possible) within the distance ranges (± 50 characters). Targets were also matched according to character distance, uniqueness of the character strings, and number of each type in each familiarity file. There were 30 problems of each type in each file.

Dependent Variables

Successes. - The number of successes was used as the dependent variable in the experiment. This variable is defined as the number of searching problems solved on the first searching attempt (i.e., the desired target was found on the first search) within a particular combination of the independent variables.

Search Strings. - The actual search strings issued by the subjects were also recorded. The average string length for each problem was used

as input to the search string selection model. The search string data was not used in the present experiment.

Blocking Factors

The variables subject and presentation order of familiarity files were used as blocking factors in the experiment.

Statistical Design

A split-plot factorial statistical design was used in the experimental analysis. This particular design was selected for its ability to isolate all variables of interest.

Plots were split over the distance information (D) condition. The split-plot design is divided over the levels of familiarity; a subject's initial exposure was assumed to increase statistical knowledge of text contained in the file. If this were so, a within-subjects design would increase learning about the string frequencies and content of the files and therefore bias the number of successes in searching problems.

The factors familiarity (F), distance information (D), problem type (T), presentation order (O), and subjects (S) were considered according to Figure 4-1. Distance information (D) is a between subjects variable, the remaining variables are within-subjects with presentation order (O) counterbalanced in the design.

The linear model in Figure 4-1 contains only those variables and interactions important to the analysis. In the experimental design, order (O) and all interactions involving O are excluded from the model.

Initially, it was desired that presentation order effects should be considered. Because presentation order was balanced within plots (see Figure 4-1), presentation order was aliased with Distance information (D). In the experiment, it was assumed that D and O would both give a positive effect to the ANOVA from both unfamiliar to familiar condition and from distance information not given to distance information given conditions. A one-way ANOVA using only O in the linear model revealed that the presentation order effect was not significant. Since D and O were aliased in the design and both were expected positive effects, the non-significance of the ANOVA indicated that the experimental design was adequate to accurately analyze the data. The presentation order (O) variable was therefore deleted from the model to make all other effects of interest estimable by the least squares method.

Also omitted from the model were interactions FT_{ij} , $FTS_{ijl(k)}$, and $FTDS_{ijkl(k)}$. The FT_{ij} interaction was not expected because, as given in the Problem Type (T) description, problems were matched according to distance, content, and distribution within each level of familiarity. $FTS_{ijl(k)}$ and $FTDS_{ijkl(k)}$ were omitted because individual subject differences were not of interest to this study.

Random subject effects nullify standard F-statistic assumptions. Due to this condition, approximate F-statistics were formulated for experimental effects according to Satterthwaite's method for

		FAMILIAR				UNFAMILIAR				(F)
		ORDER 1		ORDER 2		ORDER 1		ORDER 2		(O)
(D)		NAME	VAR	NAME	VAR	NAME	VAR	NAME	VAR	(T)
DISTANCE_INFO_N	S ₁	X	X					X	X	
	S ₂			X	X	X	X			
	S ₃	X	X					X	X	
	S ₄			X	X	X	X			
	S ₅	X	X					X	X	
	S ₆			X	X	X	X			
	S ₇	X	X					X	X	
	S ₈			X	X	X	X			
DISTANCE_INFO_Y	S ₉			X	X	X	X			
	S ₁₀	X	X					X	X	
	S ₁₁			X	X	X	X			
	S ₁₂	X	X					X	X	
	S ₁₃			X	X	X	X			
	S ₁₄	X	X					X	X	
	S ₁₅			X	X	X	X			
	S ₁₆	X	X					X	X	

EXPERIMENTAL DESIGN

X = DATA COLLECTED IN CELL

= DATA NOT COLLECTED IN CELL

Linear Model:

$$Y_{ijk1} = \mu + F_i + T_j + D_k + S_{1(k)} + FD_{ik} + FS_{i1(k)} + TS_{jk} + TD_{jk} + FTD_{ijk} + \epsilon$$

where:

F = familiarity level (i = 1,2)

T = problem type (j = 1,2)

D = distance_info level (k = 1,2)

S = subject (1 = 1,2,...,16)

O = presentation order (omitted from linear model)

Figure 4-1 - Experimental Design and Linear Model

Approximating F-statistics (from Montgomery, 1984). When exact tests do not exist, Satterthwaite has demonstrated a technique for calculating an approximate F-statistic using linear combinations of mean squares to isolate the mean square of interest. The technique used to calculate the approximate F-statistics for all conditions were performed according to the procedure outlined in Appendix B. Also in Appendix B are the actual equations used to calculate the F-statistics.

Subjects

Sixteen junior and senior undergraduate students (mean age = 21.2 years) from an introductory Industrial and Systems Engineering Man-Machine Systems course at the Georgia Institute of Technology served as subjects for the experiment. Subjects were required to have a working knowledge of FORTRAN 77 (i.e., at least one formal course and 1 year of programming experience with the language) and a general familiarity with computer text editors and various editor commands (e.g., search commands and cursor control keys). Subjects received class credit for experiment participation.

Training

There were three experimental sessions. The first session was an initial briefing to become familiar with the task and instructions to be followed during individual study; the second was an individual learning session where the subjects became familiar with the experimental

material; and the third was the actual on-line text editing session. All instructions, questionnaires, and training materials are contained in Appendix C.

Subjects were given a printed copy of an approximately 1000 line FORTRAN 77 program, along with a one page questionnaire (see Appendix C). During the second experimental session subjects answered various questions about the program content and functionality. For example, they were asked to identify various aspects of the program, such as the particular functions called in a subroutine, the order that function code appeared in the printed source code, etc.

Additionally, subjects were asked to do walk-throughs of code sections to instill structural as well as functional knowledge of the source. There was an oral review of the questionnaire immediately prior to the on-line editing session (session two) to ensure that the subject had understood the questions.

Procedure

The sixteen subjects were randomly assigned to one of two groups: Target distance information given, where target distance information was given for searching stimuli, or target distance information not given, where target distance information was provided for searching stimuli. This condition held for the entire experiment for the particular subject. Experimental sessions were conducted on an individual basis

and ran approximately 2.3 hours in length.

The question sheet was reviewed with the subject before the searching task began to ensure that the learning session was effective. In particular, questions pertaining to structural and functional knowledge of the familiar condition file were asked. Seventy-five percent correct answers on the oral examination was considered adequate for the subjects to continue with the experiment. If the subject did not attain this, an additional learning session was provided with experimenter help available. Only one subject did not attain the desired score on the first test. An additional learning session enabled the subject to pass the oral review and continue with the experiment.

The subject was then seated at a computer terminal. An experimental text editor with only three editing commands: home (return to the top of the file), search (search for the indicated string), and Control-Z (quit the editor) was used to search the files. The editor had a keystroke monitor which recorded keystrokes in serial order and cursor movement during editing.

After final instructions were given, subjects were given sixty hard copy sheets (thirty of each Problem Type in random order) from one of the familiarity conditions. On each stimulus page, containing a full page of program text, a character in a text string was highlighted with a yellow marker. The highlighted character was the target for the problem. The first ten problems were declared practice and not scored.

Subjects were instructed to find the highlighted text on each stimulus page by issuing a search beginning with the highlighted

character and adding characters after it until they subjectively judged the string unique. A unique search string was defined in the instructions to be a string of sufficient length to enable the editor to locate it on the first try. If the search was a success, they were to home the cursor (i.e., return the cursor to the top of the file) and go on to the next problem. If the initial search failed, they were instructed to home the cursor and try once more. The second searching attempt was used to identify typographical errors. For example, should the first search fail due to a typographical error, and the subject entered the second search string with the typing error corrected, it was scored as a success. This study was concerned with text searching behavior, not typing accuracy. The process was repeated for all sixty problems. A short break was given (approximately 10 minutes), then the editor was initialized with the remaining condition (familiar or unfamiliar) and the previous process was repeated.

After subjects were run on both conditions, a short debriefing questionnaire was given to solicit subjective information about the experiment.

Experimental Results and Analysis

Data from the experiment was analyzed with SAS statistical software procedure GLM using a split-plot, factorial analysis scheme over all variables. The analysis was conducted using the linear model given in

Figure 4-1.

Table 4-1 shows the mean performance results for the text searching experiment. The table reflects subject means over a condition. A significant difference was found due to levels of familiarity using Duncan's Multiple Range test. Distance information levels did not exhibit a significant performance difference.

The ANOVA results are given in Table 4-2. Familiarity had a marked effect on a subject's ability to locate specific text on the first attempt ($F(1,14) = 14.36, p < 0.0026$). Distance information had no effect on subject performance ($F(1,14) = 0.0625, p < 0.9371$). Additionally, Problem Type had no effect on performance ($F(1,14) = 1.92, p < 0.1880$). No significant interactions between factors were observed.

Discussion

Performance within levels of familiarity (F), the only significant factor, varied according to the experimental hypothesis: Subjects

Variable	Mean # correct responses (25 possible responses)
Familiar	21.88
Unfamiliar	19.75
Distance Information Y	20.84
Distance Information N	20.78

Table 4-1 Mean Number of Correct Responses By
Problem Type Within Familiarity and
Distance Information Condition

Source	DF	Sum of Squares	F	Pr > F
Familiarity	1	72.25	12.31	0.0035
Distance	1	0.0625	0.01	0.9371
Information				
Problem Type	1	10.53	1.92	0.1880
Subject	14	135.69	155.07	0.0629
Problem Type x	1	0.25	0.05	0.8344
Distance Inf.				
Familiarity x	2	16.31	1.95	0.1796
Problem Type x				
Distance Inf.				

Table 4-2 ANOVA Results.

located ten percent more target strings on the first attempt in the familiar environment vs. the unfamiliar environment.

The distance cues supplied to the subjects in the distance information given condition did not have any significant effect on searching performance. There are three possible reasons for this result. Either the distance information cues were: (1) not perceived by the subjects when searching, (2) the subjects already knew the relative distance to cover, indicating that the cues provided no additional information, or (3) distance information is not really used in search string selection. If humans are sensitive to distance information when formulating search strategies, they do not rely on the specific quantities given in the experiment.

It was expected that distance information would have a positive effect on searching performance, particularly in the form of a distance information x familiarity interaction. Subjects had knowledge of the

intervening text between cursor origin and desired cursor location in the familiar environment. In the unfamiliar environment, however, estimation of search string length was dependent on the amount of intervening text based on knowledge of the language, syntax, etc.

The quantity of distance (in characters) was obviously not reflected by the distance cues given in the experiment. It appears that a more distinct indicator of character distance should have been used, perhaps a table of conditional probabilities of character N-grams (after Shannon, 1951).

Comments solicited from subjects revealed that they recognized frequent text strings and increased search string length accordingly. Also, it appeared that subjects sought closure for strings that spelled out words. For example, even though the entire word was not necessary to locate the target on the first searching attempt, subjects tended to finish the entire word (e.g., "putstring", instead of "putst").

Upon reevaluation of the stimuli, it appeared that there was not a qualitative difference between the two problem types. Originally, it was expected that users of a well known programming language would identify language keywords as occurring more often in a large program text. Subjects reported no information that would indicate the sensitivity to keywords. Instead, frequently occurring strings were recognized according to perceived number of reinforcing occurrences. Individual subject difference data were not analyzed.

In summary, a human's level of familiarity with a file being edited does indeed affect searching performance. Based on the results of this experiment, the other factors considered (distance information, problem type, and order) do not influence searching performance. These results, along with search strings given by subjects in the searching experiment will be analyzed against model performance in Chapter 5.

A Reanalysis of the Data

Although subjects' success rate remained consistent over variant target character distances, the non-significant effect found for distance information in the ANOVA was questioned. Referring to previous discussion, if distance information was not utilized in subjects' search string length strategies, then string lengths should not vary according to character distance in the search problem. It would be reasonable to assume therefore, that if subjects' search success rate remains consistent over variant distances, search string length should increase with distance.

With the above heuristic in mind, the search experiment data was reanalyzed using search string lengths (i.e., in characters) as the dependent variable. The reanalysis revealed a significant effect for distance information in addition to the familiarity effect (familiarity: $F(1,14) = 24.33$, $p < 0.0002$; distance information: $F(1,14) = 20.33$, $p < 0.0005$). No other significant effects were identified in the reanalysis.

It can therefore be concluded that distance information was utilized by subjects in determining the length of the search string. Distance information did not affect overall success rate. However, it did affect search string length selection because string lengths were increased monotonically with increasing character distances.

Frequency Estimation Experiment

An on-line text string frequency estimation task was given immediately after the text searching task. This task was designed to allow the experimenter to compile subjective frequency estimates of specific text strings in the files (after Ehrlich, Damon, and Cooper). The hypothesis in this experiment pertained to a subject's knowledge of text content. Even though learning occurred in the unfamiliar environment, the subjects' knowledge of character string frequencies in the file would afford better performance in the familiar environment.

Subjects

The same subjects were used in this experiment.

Procedure

A C-program was used to present lines of text from each file (one file at a time). A stimulus presentation rate of 4 seconds was chosen to limit subjects' responses to recognition memory for each text string.

During this task, part of the text presented on the screen was highlighted in reverse video. Subjects were asked to estimate how many times they thought the highlighted text appeared in the file, then enter an integer estimate at the keyboard. Results were averaged over subjects and the mean estimate was regressed against the actual frequency value for both levels of familiarity using SAS statistical software procedure REG. The subject mean was used as regressor since estimation accuracy for the average subject was desired.

Estimation Task Results

Results from the estimation task are shown in Table 4-3.

Preliminary analysis of the R-squared (adj.) values indicated that subjects were much better at estimating the number of occurrences of a particular string in the familiar condition. A correlation analysis of

Familiarity	F-value	Prob	R ² (adj.)
Familiar	44.58	p < 0.0001	0.425
Unfamiliar	12.10	p < 0.001	0.160

Table 4-3 Regression results of estimation task.
by familiarity type

actual versus estimated string frequencies was conducted using the bivariate normal population correlation analysis technique given by Ostle (1963). The results of this analysis showed that the hypothesis that the two correlations were equal could not be rejected.

In Ehrlich, Damon, and Cooper (1983) it was shown that subjects were insensitive to word occurrence frequencies above 10 occurrences. Considering that some of the experimental stimuli occurred over 30 times, a non-linear relationship between actual and estimated number of string occurrences was suspected. The mean subject estimate was therefore normalized using a percent error estimator and the regression analysis was rerun. Although the R-squared (adj.) values increased for both levels of familiarity (R-squared (familiar) = .645, R-squared (unfamiliar) = .463), recalculation of the bivariate normal population correlation analysis technique (Ostle, 1963) again indicated that the null hypothesis could not be rejected.

Discussion

Given the correlation analysis results, it cannot be concluded that subjects possessed a higher level of string frequency knowledge about the familiar file. However, a trend is indicated in that direction. Upon reanalysis of the data collected in this experiment, sample size and number of problems appeared to contribute to the non-significance of the correlation differences. Further research is needed with larger samples to discern this quantity.

One interesting result was noted during comparison of actual frequency of occurrence and subjects' estimated frequency of occurrence of a text string. Except for a few outliers, subjects' estimates did not exceed 20 even though some of the more common strings occurred in excess of 60 times in a file. This result suggests a bias toward low frequency estimation possibly due to anchoring effects in the estimation process.

CHAPTER V

COMPARATIVE ANALYSIS OF MODEL VERSUS HUMAN PERFORMANCE

Introduction

A comparative performance analysis between the search string selection model developed in Chapter 3 and the empirical study of Chapter 4 is described in this chapter. The same search problems from the empirical study were used in the model validation runs. The search string selection model was run within both levels of familiarity. Search strings issued by subjects in the text searching experiment were then compared to the search strings selected by the model on various dimensions.

Performance was compared in terms of: (1) The number of search successes for the model and subjects in both levels of familiarity, and (2) Search string lengths used by the model and mean string length issued by the subjects.

Performance Analysis

Overall text searching performance was analyzed based on percentage of search successes by level of familiarity. It was found that the search string selection model performance was similar to mean subject performance within both levels of familiarity.

Model performance was equal to mean subject performance in the familiar condition. Both the model and subjects (average) exhibited an 86% success rate (based on 50 searching problems). The model failed on 3 (of 7) of the same problems as the subject mean.

The unfamiliar editing condition had even more interesting results. The model exhibited a 74% success rate, while subject mean was 80%. The 13 model failures encompassed 10 of the same failures as the subject mean.

A chi-squared analysis was conducted on the performance results. The chi-squared hypothesis: " H_0 : successes versus failures are independent for the average subject versus model performance" was rejected in both cases (familiar condition: chi-square = 12.58, unfamiliar condition: chi-squared = 35.58, where chi-square = .0001 10.8). The chi-squared test results indicate that the model tended to exhibit the same behavior as the average subject on the same experimental problems.

Model performance within the familiar condition was always better than the unfamiliar condition, further supporting the experimental hypothesis espoused in the empirical study: greater knowledge of the text content and structure will aid in text searching performance.

Search String Length Comparison

Search string lengths for the model and the mean string length for subjects were analyzed by search problem. Results of this analysis showed that although the number of successes were equivalent in the performance analysis, the search string lengths chosen by the model did not correspond to those chosen by subjects.

Search string data was plotted for search problems on which both the model and subject mean string lengths were successful. Subject mean string lengths were used to estimate the average subject response; data are plotted by familiarity condition. Figure 5-1 shows the mean subject search string lengths versus model search string lengths. The

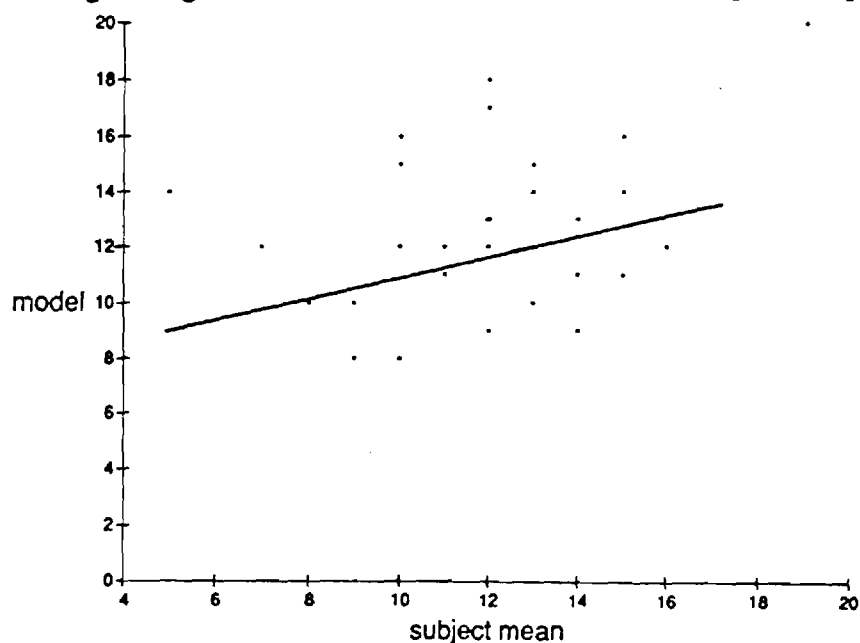


Figure 5-1 Mean Subject vs. Model Search String Lengths.
Search Successes Under Familiar Condition.

correlation between mean subject length and model length was 0.23.

The unfamiliar condition data yielded better results, as shown in Figure 5-2. Correlation of model search string lengths and mean subject search string length was 0.54.

Discussion

The model of human search string selection reflects overall human performance in both familiar and unfamiliar programming language text environments. Model adequacy based on number of search successes is quite acceptable as shown in the overall performance analysis.

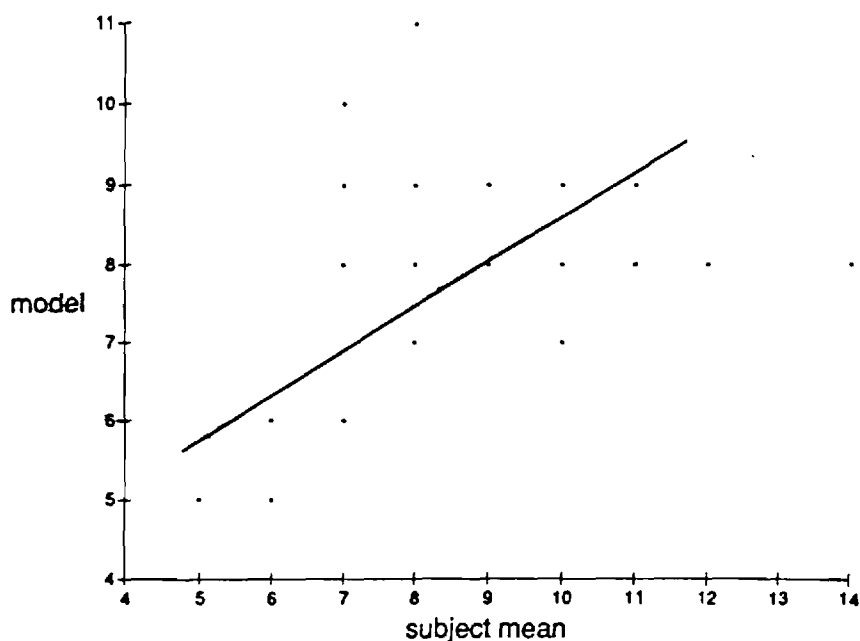


Figure 5-2 Mean Subject vs. Model Search String Lengths
for Successful Searches Under Unfamiliar Condi-
tion.

Upon finer-grained analysis, however, it was discovered the model did not correspond to actual human performance on individual searching problems. This can be attributed to the algorithm used in determining the information-theoretic bits threshold for a searching problem. The two-state Markov process, described in Chapter 3, used an estimate of the overall desired probability of success, not an individual probability based on the particular problem of interest.

An interesting trend was noticed in Figures 5-1 and 5-2. In Figure 5-1, note that the model tended to issue longer search strings than the subject mean, even though a shorter search string would suffice. Conversely, the search strings issued by the model in the unfamiliar condition (Figure 5-2) were shorter than the mean subject search string on the same problem.

A possible explanation for this observed trend is based on information theory. The search string selection model used B_c , or bits per character c in the file being searched, as the bits value used to determine the search string length. Although adequate for the observed overall searching success ratio, B_c gives the model no increased knowledge of the file's structure or function in the familiar condition. There was no input to the model reflecting this increased knowledge.

The shorter search strings issued by subjects in the familiar condition indicates that an additional input to subject search strategy may be present. This input may be similar to the M_N , or text transition, matrix of the alternate model given in Chapter 6. It is hypothesized that the search string selection model algorithm combined

with the text transition matrix scheme of the Chapter 6 model would yield higher correlation values. In order to do this, B_c would be calculated based on the transition probability from the previously known characters in the string to the character of interest. The level of N in M_N would vary over level of familiarity, thus providing the string selection model with varying knowledge of the text.

CHAPTER VI

AN ALTERNATE MODEL OF SEARCH STRING SELECTION

Background On Alternate Model

During the thesis research, an alternate model was developed simulating the entire search string selection process within the editing environment. Instead of simulating only the human's probabilistic search process, this model simulates the editor searching through the editing environment. The human's knowledge of the text is represented by varying levels of transition matrices.

The level of transition is defined as the number of previously known characters (from Shannon, 1951) in the model. The probability of a particular transition entry in the matrix is the probability of a specific character occurring after a character string of length equal to level of transition. At transition level 3, for example, there are several possible characters (e.g., "t", "m", "f", etc.) following the character string "roo". The transition probability of the character "m" occurring after the string "roo" is equal to the number of occurrences of the character "m" after "roo" divided by the total number of occurrences of the level 3 transition string "roo".

This model uses the same theoretical basis as the model described in Chapter 3, but was not successful. It was abandoned during the thesis research because:

- 1.) Model runs required too much computer time and,
- 2.) The model still had not converged on a reasonable solution by the time it had been abandoned.

Basically, modeling of individual searching problems had reached level 11, and convergence acceleration predicted a transition matrix of level 20 or higher to solve it. At level 11, each searching problem required approximately 83 minutes to solve a problem for one hundred iterations. Multiplied by 50 problems in each of two searching environments, model execution time exceeded 2.8 days per environment. This execution time made the model prohibitive for this type analysis.

The alternate model is functionally described in this chapter. Following the model description and implementation sections, a comparison between the two models is conducted.

Alternate Model Overview

In the model, the probability estimate of search failure is modeled as a function P of three variables: the search string S , the distance D to the desired position, and the text transition matrix M_N . A transition matrix entry contains the probability that a given N -character sequence is immediately followed by another character in the file being edited (see Shannon, 1951). As N increases, the matrix contains an increasing

amount of the file's structure. Further, at some value of N , the matrix M_N uniquely reproduces the file.

Using the transition matrix, the simulation estimates the probability of a given string occurring at least once in D transitions. The accuracy of the probability estimate increases as N increases. This is due to the larger number of previously known characters. The same increase will cause the model to produce search strings of decreasing length if the other variables P_g (i.e., the user's desired probability of success) and D are held constant. The user's understanding of the text is modeled by the value of N . This value causes the model to issue search strings with lengths closest to those of the user. Conservative behavior (e.g., issuing a search string much longer than necessary to be unique) should be reflected in a smaller value of N . Thus, the model minimizes $|S|$, or the length of the search strings issued, subject to the constraint:

$$P_g \leq P(S, D, M_N)$$

where:

P_g = Desired probability of success,

S = search string,

D = distance in characters.

M_N = text transition matrix

and the model is fit to the user by choosing the best value of N . The goal of this model was to show that different values of N resulted from different experimental conditions.

Model Implementation

Search string data was gathered empirically. The search strings were recorded along with the distance (in terms of the number of characters) to the desired point in the file as a human conducted predetermined search tasks. Once the search string data was collected, a data file consisting of all character transitions in the file being edited was constructed. There was one data file for each level of N (see previous section).

When all of the necessary data was available, the model was run with both the search string data file and the transition matrices (one for each value of N) of the file being edited. M_N was used to simulate the text between the current and desired position. A random number generator was used to make choices when the transitions were not deterministic. The matrix was then searched repeatedly for the search key. The output of the searching processes was the number of times the key was found in each creation of the intervening text. When the predetermined number of runs was complete, the result was converted to P_s . If P_s was lower than the desired probability, then N or $|S|$ is adjusted to fit the user more exactly.

Analysis of Model Components

The computer program, herein known as "SPOCK", receives input from two data files. One file contains the search strings given to the text editor by the human to locate specific text. Along with the search string, character distances traversed by the cursor from beginning point in the file to the first matching character pattern in the file are also supplied to SPOCK.

Text File Transition Matrix Process - The second data file supplied to SPOCK is the text character transition matrix. The transition matrix, in the form of a data structure, allows SPOCK to possess complete knowledge of the contents of the file. This knowledge is known as "familiarity".

To produce the second data file, the text to be edited is run through a LISP program which constructs the M_N matrix. For example, at a level of $N = 6$, the string "abcde" is noted by the LISP program to occur 15 times in the file. This string is followed by the character "f" 10 times, and the character "g" 5 times. The LISP program compiles these relationships in the entire file in a format that can be read by SPOCK when it simulates the editing environment. By using these relationships at varying levels of N , SPOCK can probabilistically step through the file to search for the indicated segment of text. Then the program can be assumed to have complete statistical knowledge at the level of N for the character transition probabilities in the file.

Computer Model of Text Editing Environment

The computer model, or "SPOCK", was written in the "C" programming language and follows Shannon's (1951) theory of conditional probabilities of letter frequencies. The human's implicit statistical knowledge of a language was modeled for predictive purposes and encoded in the program's data structure. Level of "knowledge", or N, was varied with the degree of in-depth understanding of the text being edited. For example, a higher level of familiarity with the text is reflected in a higher level of N in the simulation.

SPOCK simulated the text editor searching through the given text. The model received the human input, text file transition matrix, and character distance from current cursor position to desired position. The simulation process is described below. SPOCK's source code is included in Appendix D.

The Simulation Process

SPOCK utilizes both environmental and human inputs to simulate the editing environment. In this section, the model is described in more detail.

Before SPOCK receives search string input, it establishes a simulated text editing environment. The first task is forming the data structure containing the desired text transition matrix based on the level of N. When working with large transition matrices, access to the

proper next transition string may be time consuming. A large hash table data structure is implemented. The hashing index is calculated from the decimal product of the previously known letters in the structure.

Hashing is important to the model. Without an efficient indexing system to locate transitions, the time to run a simulation would be inordinately long. For example, the M_n matrix is large (3,000 transitions) at level 3; the matrix increases to approximately 12,000 entries at level 8.

The second input is a known current cursor starting point. This input is a character string of length N preceding the cursor starting position.

Once the editing environment is formed, the human's search string information is input to SPOCK with the corresponding character distance from the top of the file. Each string is treated as one problem to solve in serial order.

Finally, given probabilistic knowledge of the text, distance information, a starting point (where the cursor is presently located in the file), and the search string selected in the experimental editing task, SPOCK will calculate the number of times the string was found in the simulated environment. Each problem is evaluated by SPOCK and a probability estimate (i.e., number of times found divided by the total number of simulation runs) is calculated based on the level of N, the search string length, and the character distance to traverse to the target string (problem).

Model Comparison

Although implemented first, this model did not attain high predictive performance compared with human and the former (Chapter 3) model's performance within the experimental editing environments. In the introduction to this chapter, the results of high transition level (level 11) simulations indicated individual solutions to problems were not converging. Given the high computation time for each problem, this model was abandoned in favor of a more suitable model.

An extensive comparison and contrast study was not conducted. However, a few significant results from the experimental validation study indicated a threshold model was more appropriate for modeling human search string selection behavior.

There were three basic motivations that led to the design and implementation of the search string selection model described in Chapter 3. First, there was a definite increase in probability of searching success within a distance D with an increase in N during the initial simulations with experimental data. The simulation process, as described in the chapter introduction and above, became computationally prohibitive above $N = 10$ (10 character strings in M_N). This observation indicated that the human's statistical knowledge was more in-depth, possibly to $N = 20$ or higher. This trend also indicated that the simulation did not contain enough environmental information to model the text searching process accurately. As a result, a smaller, more

constrained model was desired.

Second, simulation of individual searching problems showed the highest level of N in one particular searching problem was not always the best approximation of the human's overall searching success level. Since SPOCK used only one level of N for all problems during the simulation, a requirement for the smaller model was the ability to vary the level of knowledge within the model during simulation.

Third, analysis of low frequency strings (e.g., those text strings occurring less than three times in the text file) in experimental problems demonstrated that the sum of information bits in a low frequency string often equaled or exceeded the longer, less unique strings in other problems.

As a result of the three reasons given above, the search string selection model described in Chapter 3 was developed. It models the human search string generation process based on bits of information contained in both the text being searched and the statistical process conducted by the human while searching.

CHAPTER VII

CONCLUSIONS

In this thesis, the issue of how a user selects a search string in a text editing environment was analyzed. The problem space was constrained to editing of production sized software source code in which the user's task typically, is to locate a specific occurrence of a character string.

Empirical Study

Two experiments were conducted to determine effects of the important variables identified in literature review. The first experiment involved text searching problems. Subjects were given specific character strings to locate in two FORTRAN 77 source code files: one they had studied extensively to acquire structural and organizational knowledge of the program's content (familiar condition), the second they had never seen before (unfamiliar condition). Both files were equivalent in length and code complexity.

A split-plot experimental design was utilized in the experiment. The experimental group was supplied with distance information cues. These cues were designed to give some indication of the character distance between the current cursor position (the beginning of the file) and the desired cursor position. Two types of search problems were given: language keywords (e.g., SUBROUTINE), and variables (e.g., local variables defined by the programmer).

It was found that familiarity was the only experimental variable affecting search performance. Subjects found 10% more search targets on the first attempt in the familiar condition versus the unfamiliar condition. Distance cues were apparently either: a.) not used by subjects, b.) ineffectual because subjects already knew the relative distance to traverse, or c.) irrelevant to search string selection.

A reanalysis of the data using search string lengths as the dependent variable revealed a significant effect for distance information in addition to the familiarity effect. It was concluded that since subjects' hit rate remained consistent over variant target character distances; distance information was utilized by subjects to determine the length of the issued search string.

The second experiment was a character string frequency estimation task. This experiment was designed to provide an estimate of the subjects' knowledge of specific strings. Subjects were presented with lines of code where part of the line was highlighted in reverse video (one file at a time). Subjects were requested to estimate how many times the highlighted string occurred in the indicated file.

The mean subject frequency estimate for each estimation problem was regressed against the actual string frequency value. Although a trend was indicated towards more accurate estimation in the familiar environment, hypothesis testing revealed that the correlation coefficients were not statistically different. A reanalysis of the data with mean subject estimates normalized by a percent error estimator also yielded non-significant results.

Models

Two models were developed to study the text searching process. Both were based on Shannon's (1951) results which state that the human has implicit statistical knowledge of a familiar language (e.g., structural, syntactic, and semantic knowledge).

The unsuccessful model described in Chapter 6 attempted to simulate the editor searching through text using subject inputs. It was abandoned due to prohibitive run-time and non-converging solutions for a majority of the searching problems. No data were analyzed from the Chapter 6 model.

The model of human search string selection described in Chapter 3 used information-theoretic bits as a string length decision threshold. The user's searching performance was modeled in terms of the number of bits required to move the cursor from the current position to the desired position. It was hypothesized that the user would utilize the statistical knowledge of the programming language in determining the

optimal length of a search string based on four inputs: the character distance to traverse, a potential string of characters to choose from, the desired probability of search success, and the information-theoretic bits per character associated with a character from the file being edited.

A two-state Markov process was used to determine the user's probability of search failure. It simulated the editor searching through the text character by character with a small probability of finding the search key at every point in the text between current and desired cursor position.

Model runs conducted with data from the empirical study showed the model predicted human performance accurately. Also, the model's search successes and failures correspond well to those of the average subject. The model's search string lengths correlate significantly, but not strongly, with those of the average subject. There is much room for improvement in the model in this regard.

Conclusions and Suggestions for Future Research

Given the constrained environment of this thesis, the search string model performance corresponded well with human performance. The model of human search string selection was adequate for this thesis research. The model, however, had no means for including text familiarity in its search string length selection. In Chapter 5, a proposed improvement was to employ N-gram transitions to estimate the bits for additional

characters. If implemented, the value of N could represent familiarity in terms of previously known characters.

The model, although tested in a constrained environment, can be useful in research and practical applications. For example, the straightforward information-theoretic basis can provide a statistical model of textual knowledge within a larger model of human information processing. The textual knowledge component of the model would provide a parsimonious approach to the complex processes involved with text searching behavior.

A more general model of editor searching can be built from the foundation provided by the model described in the thesis. Recall that the experimental paradigm was rigid in that the user was forced to search using a specified string. Only the length could be chosen. A more realistic paradigm would be to allow the user to search for any string in the vicinity of the desired position, and subsequently issue a second search to position the editor exactly.

The interesting question in this paradigm is how does the user choose the first, approximate location search string. The user might be imagined to scan the text centered around the desired position for potential targets, using frequency or semantic salience as the criterion. The model developed in this thesis could be used to model how the user chooses one of the alternatives generated by scanning.

Research aimed at understanding the human information processes involved with text editing, particularly editor positioning, fall into two categories: intraline positioning, and interline positioning. The

Hammer and Rouse (1982) study is an example of intraline positioning research; this thesis is an example of interline positioning research. Several methods exist for locating text (e.g., search commands, screen by screen search, etc.). By investigating the methods of positioning available and the categories of positioning, insight into the human information processes responsible for these behaviors will continue to be refined.

APPENDIX A

SOURCE CODE FOR SEARCH STRING SELECTION MODEL

```
;This model represents a search string selection model based
; on four inputs:
;     a potential string of characters to choose from
;     a probability threshold that triggers the character
;         selection process to cease
;     an array of character frequencies from the file
;         being searched
;     the character distance to traverse
;
;     Written by: Rob Andes
;     Date:      24 MAR 87
;
```

```
(DEFUN CHOOSE-SEARCH-STRING (potential-string
                             prob-threshold
                             *char-freq-array*
                             distance)
  (let ( (work-string nil)
        (string-prob 1.0)
        (bits-per-char
         *char-freq-array*)
        achar
        (dist (read-from-string distance))
        )
    (setf *aref-overflow* nil)
    (do ( (inx 1 (1+ inx)) )
      ( (OR (< (find-in-distance string-prob dist)
                prob-threshold)
            (equal *aref-overflow* T)
            )
        )
      (if (< (1- inx) (1- (length potential-string)))
          (progn
            (setq achar
              (aref potential-string (1- inx)))
            (format t "achar = ~d~%" achar)
            (setq string-prob
              (* string-prob
                (shannon-prob achar bits-per-char inx)
                ))
            (setq work-string
              (append work-string `(~,achar)))
          )
        )
    )
```

```

    ) ;end progn
;ELSE
  (setf *aref-overflow* T)
  ) ;end if
  ) ;end do
  (if (equal *aref-overflow* NIL)
    (progn
      ; (format t "search string chosen = ")
      (dolist (char work-string)
        (format t "~A" (convert-to-char char))
      )
      (terpri t)
      (coerce work-string 'string)
    ) ;end progn
  ;ELSE
    "string aref overflow"
  ) ;end if
) ;end let
) ;end function

(DEFUN RUN-STRINGS (char-file dist-file out-file)
  (let ( (strings (make-array 53))
        (distances (make-array 53))
        predicted-string
      )
    (read-char-freq "jetchar.dat") ;char freq array
    (setf *string-ptr* (open char-file :direction :input))
;fill strings
    (do ( (line (read-line *string-ptr* nil 'EOF)
                (read-line *string-ptr* nil 'EOF))
      (i 0 (1+ i))
    )
    ( (or (equal line "EOF") (equal line 'EOF)) )

    (setf (aref strings i) line)
    ) ;end do
    (close *string-ptr*)

;fill distances
    (setf *dist-ptr* (open dist-file :direction :input))
    (do ( (line (read-line *dist-ptr* nil 'EOF)
                (read-line *dist-ptr* nil 'EOF))
      (i 0 (1+ i))
    )
    ( (or (equal line "EOF") (equal line 'EOF)) )
    (setf (aref distances i) line)
    )
    (close *dist-ptr*)
;open output file
    (setf *output-ptr* (open out-file :direction :output))
;run the model for each string
    ;string probf char-freq dist
    (dotimes (i (length strings))
      (setq predicted-string
        (choose-search-string

```

```

    (aref strings i) .210 character-frequency (aref distances i))
;      (format t "predicted-string = ~S~X" predicted-string)
(format *output-ptr*
  "~S ~S ~D~X" (aref strings i)
    predicted-string (length predicted-string))
  ) ;end dotimes
  ) ;end let
  (close *output-ptr*)
) ;end function

;char-file dist-file out-file
(run-strings "jetstim.dat" "jetdist.dat" "jetfir.out")
(setf char-file "c:\\gclisp\\strings.dat")

(DEFUN SHANNON-PROB (achar prob-of-char inx)
  (let (
    (shannon-scale '(1.0 .131 .249 .149 .162)) ; ned regression
    (shannon-scale '(1.0 .086 .427 .267 .267)) ; jet regression
    scale-factor
  )
    (if (>= inx (length shannon-scale))
      (setq scale-factor
        (first (last shannon-scale)))
      ;ELSE
      (setq scale-factor
        (nth (1- inx) shannon-scale))
    ) ;end if

    (expt 2.0 (- (* scale-factor (aref prob-of-char achar))))
  ) ;end let
) ;end shannon-prob

(DEFUN FIND-IN-DISTANCE (prob distance)
  (do ( (tm prob)
    (i 0 (1+ i))
  )
    (( > i distance) tm)
    (setq tm (+ tm (* (- 1.0 tm) prob)))
  ) ;end do
) ;end find-in-distance

(find-in-distance .00525 394)

;looking for .875 familiar
;looking for .790 unfamiliar

(DEFUN CALCULATE-CHAR-FREQ (file)
  (let ( (data-file (open file))
    (charcount 0)
  )
    (setf character-frequency (make-array 127))
    (dotimes (i 127)
      (setf (aref character-frequency i) 0)
    ) ;initialize array
  )

```

```

(do ( (c (readc data-file) (readc data-file)) )
  ( (equal c 'EOF) )
    (setf (aref character-frequency c)
      (1+ (aref character-frequency c))
    )
    (setf charcount (1+ charcount))
  ) ;end incrementing loop
  (setf total 0)
(dotimes (i 127)
  (if (equal (aref character-frequency i) 0)
    ;THEN
    (setf (aref character-frequency i) 0)
    ;ELSE
  (progn
    (setf char-prob
      (/ (aref character-frequency i) charcount))
    (setf total
      (+ total char-prob))
    (setf (aref character-frequency i)
      (coerce
        (log
          (/ 1 (/ (aref character-frequency i) charcount))
          2.0) 'short-float)
        ) ;end setf
    ) ;end progn
  ) ;end if
) ;end dotimes
; (format t "total character probability = ~D~%" total)
; (dotimes (i (length character-frequency))
;   (format t "~a ~d ~%"
;     (convert-to-char i) (aref character-frequency i)))
; (close data-file)
character-frequency ;return array
) ;end let
) ;end calculate-char-freq
'(calculate-char-freq "c:\\rca\\thesis\\ned.f")

(DEFUN PRODUCE-REGRESSION-DATA
  (freqfile datfile outfile probfile level)
  (let ( (bits-per-char
    (read-char-freq freqfile))
    (Hx-data
    (create-hx-array probfile))
    (null-val 0)
  )

    (setq *input* (open datfile :direction :input))
    (setq *output* (open outfile :direction :output))

    (do ( (line (read-line *input* nil 'EOF)
      (read-line *input* nil 'EOF))
      (i 0 (1+ i))
    )
      (or (equal line "EOF") (equal line 'EOF)) )
      ;; (pprint line)

```

```

(format *output* "~D "
  (aref Hx-data i) )
      ; print Y to regr file
      (dotimes (j level)
        (if (>= j (length line))
          (format *output* "~D " null-val)
          ;ELSE
            (format *output* "~D "
              (aref bits-per-char (aref line j)))
        ) ;end if
      ) ;do every character in line
      (terpri *output*) ;add newline
    ) ;end do
  (close *input*)
  (close *output*)
  *output* ;for now let's look at the data file
  ) ;end let
) ;end produce-regression-data

(produce-regression-data
 "c:\\rob\\thesis\\jetchar.dat" "c:\\rob\\thesis\\jetstim.dat"
 "c:\\rob\\thesis\\jet5.reg" "c:\\rob\\thesis\\jetprob.dat" 5)

(DEFUN CREATE-HX-ARRAY (prob-file)
  (let ( (data-ptr (open prob-file))
        (Hx-array
          (make-array 50))
        )
    (dotimes (i 50)
      (setf (aref Hx-array i) 0)
    ) ;initialize array

    (do ( (line (read-line data-ptr nil 'EOF)
                  (read-line data-ptr nil 'EOF))
          (i 0 (1+ i))
        )
      (or (equal line "EOF") (equal line 'EOF)) )

      (setf (aref Hx-array i)
        (coerce
          (convert-to-bits
            (read-from-string line))
          'short-float)
        )
      ) ;end do
    (close data-ptr)
    (dotimes (i 50)
      (format t "Array entry ~d = " i )
      (format t "~d ~Z" (aref Hx-array i) )
    ) ;print array
    Hx-array ;return array
  ) ;end let
) ;end create-hx-array

(create-hx-array "c:\\rob\\thesis\\jetprob.dat")

```

```

;*****
;utilities to read strings from files
;*****

(DEFUN SAVE-CHAR-FREQ (char-array dest-file)
  (setf data-file (open dest-file :direction :output))
  (dotimes (i (length char-array))
    (format data-file "~D~Z"
      (aref char-array i)
    )
  )
  (close data-file)
)

'(save-char-freq character-frequency "c:\\rca\\thesis\\test.dat")

(DEFUN READ-CHAR-FREQ (origin-file)
  (setf data-file (open origin-file :direction :input))
  (setf character-frequency (make-array 127))
  (do ( (line (read-line data-file nil 'EOF)
              (read-line data-file nil 'EOF))
    (i 0 (1+ i))
  )
    ( (or (equal line "EOF") (equal line 'EOF)) )

    (setf (aref character-frequency i)
      (read-from-string line)
    )
  );
) ;end do
character-frequency ;return the array
)

'(read-char-freq "c:\\rob\\thesis\\jetchar.dat")

(DEFUN READC (stream)
  (read-char stream nil 'EOF)
) ;end readc

(DEFUN CONVERT-TO-CHAR (int)
  (read-from-string (format nil "~C" int))
)

(DEFUN CONVERT-TO-CHAR (char-code)
  (setf *dummy-char* " ")
  (setf (aref *dummy-char* 0) char-code)
  *dummy-char*
)

(DEFUN CONVERT-TO-BITS (value)
  (coerce
    (log
      (/ 1 value) 2) 'short-float)
  )

```

```

(DEFUN CHAR-TO-INT (char)
  (aref char 0)
)

;;*****
;; LISP library routines*
;;*****

;; taylor - define a macro for a Taylor Series approximation
;;
(defmacro old-taylor (first-n last-n first-term next-term)
  `(do*
    ((n ,first-n)
     (term ,first-term ,next-term)
     (sum term (+ sum term)))
    ((or (< (abs term) 1d-8) (> n ,last-n)) sum)))

;; taylor - define a macro for a Taylor Series approximation
;; The macro computes two terms at a time to
;; avoid alternating term signs.
;;
(defmacro taylor (first-n last-n first-term next-term)
  `(do*
    ((n ,first-n)
     (term ,first-term ,next-term)
     (pair 1.0d0) ; don't trip end condition
     (sum 0.0d0))
    ((or (< (abs pair) 1d-10) (> n ,last-n)) sum)
    (setq pair (+ term (setq term ,next-term)))
    (incf sum pair)))

;; ln - returns the natural logarithm of number
;;
(defun ln (number)
  (unless (plusp number) ; error check
    (error "log: argument must be positive"))
  (let* ((y (/ (- number 1.0d0) (+ number 1.0d0)))
        (s (* y y))
        (z y)) ; term without coefficient
    (* 2.0d0 (taylor 1 400 y (/ (setq z (* z s)) (incf n 2)))))
  (ln 1.0))

;; exp - returns e raised to the power number,
;; where e is the base
;; of the natural logarithms
;;
(defun exp (number)
  (cond
    ((minusp number) (/ 1.0d0 (expl (- number)))))
    ((zerop number) 1.0d0)
    (t (expl number))))

(defun expl (number)
  (taylor 0 2000 1.0d0 (/ (* term number) (incf n))))

;; expt - returns base-number raised to the power power-number

```

```

;;
(defun expt (base-number power-number)
  (if (integerp power-number)
      (iexpt base-number power-number)
      (exp (* power-number (ln base-number)))))

;; iexpt - returns base-number raised to an
;; integer power (internal)
;;
(defun iexpt (base-number ipower-number)
  (cond
    ((plusp ipower-number)
     (do ((result base-number (* result base-number))
         (n (1- ipower-number) (1- n)))
       ((zerop n) result)))
    ((zerop ipower-number)
     (coerce 1 (type-of base-number))))
  (/ 1 (iexpt base-number (- ipower-number)))))

;; log - returns the logarithm of number in the
;; base base (default is e)
;;
(defun log (number &optional base)
  (if base (/ (ln number) (ln base)) (ln number)))

```

APPENDIX B

CALCULATION OF APPROXIMATE F-STATISTICS

When exact F-statistics cannot be calculated due to random effects in the linear model, Montgomery (1984) has suggested approximations using linear combinations of the expected means squares to isolate the desired quadratic effect of a factor. This allows the experimenter to estimate the F-statistic of interest.

SAS GLM statistical software provides the mean square estimators for an ANOVA with random effects. Approximate F-statistics were calculated for all effects in the text search experiment. The random effects were all related to the subject variable (S). Other variables in the linear model included: familiarity (F), distance (D), and problem type (T).

The SAS GLM output and method used to calculate the statistics are reviewed below. After the output, the approximate F-statistic equations are developed.

F-statistics given in Table 4-1 were calculated using the following SAS output:

SOURCE	EXPECTED MEAN SQUARE
F	$\text{VAR}(\text{ERROR}) + 2*\text{VAR}(S(D)*F) + Q(F, D*F, D*F*T)$
D	$\text{VAR}(\text{ERROR}) + 2*\text{VAR}(T*S(D)) + 2*\text{VAR}(S(D)*F) + 4*\text{VAR}(S(D)) + Q(D, F*D, D*T, D*F*T)$
S(D)	$\text{VAR}(\text{ERROR}) + 2*\text{VAR}(S(D)*T) + 2*\text{VAR}(S(D)*F) +$

```

      4*VAR( S(D))
VAR( ERROR) + 2*VAR( S(D)*T) + Q( T,D*T,D*F*T)
VAR( ERROR) + 2*VAR( S(D)*F) + Q( D*F,D*F*T)
VAR( ERROR) + 2*VAR( S(D)*F)
VAR( ERROR) + 2*VAR( S(D)*T) + Q( D*T,D*F*T)
VAR( ERROR) + Q( D*F*T)
WAS UNESTIMABLE

```

Approximate F-statistics were formulated such that the linear of the two mean square estimators isolated the Q(COND) term of interest. Once these combinations were determined, the statistics are approximated by:

$$\begin{aligned}
 &= F / FS(D) \\
 &= \frac{\text{var}(\text{error}) + 2\text{var}(F \cdot S(D)) + Q(F, F \cdot D, F \cdot T \cdot D)}{\text{var}(\text{error}) + 2\text{var}(F \cdot S(D))} \\
 &= T / TS(D) \\
 &= \frac{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + Q(T, T \cdot D, F \cdot T \cdot D)}{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D))} \\
 &= D / S(D) \\
 &= \frac{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + 2\text{var}(F \cdot S(D)) + 4\text{var}(S(D)) + Q(D, F \cdot D, T \cdot D, F \cdot T \cdot D)}{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + 2\text{var}(F \cdot S(D)) + 4\text{var}(S(D))} \\
 &= S(D) / D \\
 &= \frac{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + 2\text{var}(F \cdot S(D)) + 4\text{var}(S(D)) + Q(D, F \cdot D, T \cdot D, F \cdot T \cdot D)}{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + 2\text{var}(F \cdot S(D)) + 4\text{var}(S(D)) + Q(D, F \cdot D, T \cdot D, F \cdot T \cdot D)}
 \end{aligned}$$

Problem Type

$$\begin{aligned}
 x \text{ Distance} = & \frac{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + Q(T, T \cdot D, F \cdot T \cdot D)}{\text{var}(\text{error}) + 2\text{var}(T \cdot S(D)) + 2\text{var}(F \cdot S(D)) + 4\text{var}(S(D)) +} \\
 & Q(D, F \cdot D, T \cdot D, F \cdot T \cdot D)
 \end{aligned}$$

The numerical results are given in Table 4-1

APPENDIX C

VALIDATION EXPERIMENT INSTRUCTIONS AND FORMS

EXTRA CREDIT FOR EXPERIMENT PARTICIPATION

Experiment - Analysis of Human Behavior in Text Editing

Principal Experimenter- Rob Andes, Center for Man-Machine
Systems Research, X3080 or X4318

Brief Description

The research will analyze human text editing behavior in a constrained environment. Participants will become familiar with some code and will then conduct editing exercises. There will be two sessions: the first will be a group meeting (~1.5 hr.), the second will be on an individual basis and will include some on-line editing (~2 hr.).

CLASS CREDIT

Class credit will be awarded for participating in this experiment. See your instructor for actual point values.

Participant Requirements

Participants should have experience with a FORTRAN dialogue and some programming experience (freshman FORTRAN is ok). "Experience" is defined as the ability to understand FORTRAN code that is already written, there is no programming involved in this experiment. Also, the participant should have experience with computer-based text editors, and the associated features (e.g.- insert commands, search commands).

Date and Time

The initial session will be held in one of the IC lecture rooms contingent on participant's schedules. Scheduling of the second session will be arranged individually.

INSTRUCTIONS

Please fill out the enclosed data sheet completely. If you have questions, use the specified space and I will get back to you. Do not ask your professor, he doesn't know.

Also, please enclose a copy of your class schedule so that I can schedule around everyone.

PARTICIPANT DATA SHEET

Name: _____

Class: _____ PO Box: _____

Phone No. (very important for scheduling): _____

Number of years experience with FORTRAN (estimate): _____

Number of years experience with text editors: _____

Other computer skills (explain): _____

Questions or Remarks: _____

REMEMBER TO ENCLOSE A COPY OF YOUR CLASS SCHEDULE!!!

PLEASE RETURN THIS FORM BY OCTOBER 20th SO THAT WE MAY GET
THE EXPERIMENT STARTED WITH YOU IN IT!!!

SUBJECT BRIEFING ON EXPERIMENT AND TASK OVERVIEW

In this experiment I am going to examine how a person conducts a text search for particular text in a text editor. There are 4 tasks that you will perform during the course of the experiment. These tasks will be done in order over <2> sessions, with a questionnaire given at the completion of the entire experiment.

The first session, this one, is the initial briefing and learning session. In the second session, you will get some hands-on-terminal exposure and do some text editing (in particular, some searching exercises). Any questions so far? Okay, let's begin.

As you will notice, I am handing out a hard copy of a program. This program, called ned (short for "new editor"), is a FORTRAN 77 based text editor designed to run on a vax 11-780. How about scanning some of the contents now.

Okay, now comes the next part of the experiment that requires you to know FORTRAN. Although you don't have to write any code, I am going to ask you to learn how the code works (e.g.-how subroutines function, the calling sequence of functions, where specific variables occur). For example, you will be asked to read over, learn the basic operation of the program and be responsible for its content (don't worry, the program is not difficult to understand). Basically, I want you to attain a level of familiarity with the code such as if YOU wrote it (but not totally). To do this, I'll give you a questionnaire (not too horrible yet, is it?), to fill out before you come in for the second session to allow for a

little learning by doing:-).

Most of the work done in this experiment is done on your own. By filling out the questionnaire, you will have to pay attention to things that I feel are important to your understanding of the program and its function. Should you have any questions, you can either call me at the Computer Coordinator's Office (x-3080), or come by to talk. I would like to stress the fact that mental effort will be the major component in understanding the program, however.

After I've given you some time to look over the code and answer the questionnaire, we'll set up a time for the second and final session. This session will involve some text editing on a terminal with some hard copies to help your memory. In addition to the text editing, I'm going to ask you some particular things about the program text online, so it would be a very good idea to pay attention to the stuff that I'm going to ask you to learn. There is one issue which needs to be addressed before we go any further with the experiment. The editor that you will be using has online data collection capabilities and keystroke monitors to aid the modeling effort. Your extra credit points are awarded on the contingency that you know the program. It is of utmost importance to the success of the experiment that you learn the concepts and quantities that you are asked to learn. Now, should you choose not to learn the program (look I'm not asking very difficult questions) it will be reflected in your extra credit. NO play, No pay, basically.

When all the "hands-on" tasks are completed I'll then ask you to complete a short questionnaire on your background. THEN, I'll tell you what all of this is about (thought I'd never get around to that, huh?).

Do you have any questions before we begin? OK, let's go.

EXPERIMENT QUESTIONNAIRE

Instructions: Please use separate sheets for answers to the questions. Put your name and second session date and time at the top of your answer sheets. There are 32 questions, please answer all of them to the best of your understanding of the program. Please do not consult anyone but me about questions or problems. I can be reached at 894-3080, 894-4318, or 874-7110.

General Questions

1. What routines use the variable "BUFC" ?
2. What routines use the variable "LOGGING" ?
3. Where is LOGGING defined?
4. What routines use the variable "SCRBOTY" ?
5. What routines use the variable "STAT" ?
6. What is a global input?
7. What are the global inputs to subroutine "CLEARSCREEN"?
8. Which function code appears first in the file, "PUNTSscreen"
or "FIXSCREEN"?
9. On what pages of the program can the previous functions
be found?
11. What does an "INCLUDE" statement do?
12. How many times does the variable "LETTERDIGIT" appear
in the program?
13. What is UNITTYPE?
14. What does UNITTYPE do?

15. How many times is "LOADUSER" used?
16. What routines call "KEYBOARD"?
17. What will happen if a CTRLN is input to ned?
18. How many subroutines are commented out?
19. How many times is "PUTSTR" called by "FIXSCREEN"?
20. What does "TEXTX,TEXTY" signify?
21. What is QUERYF"?

True or False

22. Subroutine "SETHOME" is called by subroutine "INITIALIZE"
23. The variable "CTRLZ" exits the editor.
24. There are 6 functions contained in ned.f.
25. There are 20 subroutines in ned.f
26. The variable "MYTEXTX" occurs a total of 10 times in
the program.

Editor function questions.

27. Generally describe how the ned editor functions (simulate the
computer).
28. Name the function that receives keyboard input and conducts
first processing.
29. Briefly describe how the function, named above, processes
input.
30. How many times is subroutine "GETCOMMAND" called?
31. Describe the function of "BINDEX".
32. What function does "SCAN" perform?

INSTRUCTIONS FOR TEXT EDITING TASK

In this experiment, you will be asked to do several text editing tasks on two separate files: one will be the file that I've asked you to learn, the other will be a file that you've never seen before.

Although ned is a powerful editor, several of the commands and features have been disabled for this experiment. During the course of the experiment, you will only be allowed to use three of the ned functions: "search"("/"), "home"("home" key), and End (Cntrl-Z). Backspace ("<-") will also be enabled so that you may correct your spelling, etc. Using two commands, home and search, you will locate indicated segments of text. Please only use ^Z at the end of the data file (I'll tell you when). The experiment procedure follows...

In this experiment, you will use your knowledge to locate certain sections of text in a file (e.g.- the program I've asked you to learn) using a search command sequence. Efficiency is the key in this experiment: I want you to use all the available information to find the desired text by issuing only one search command per test problem (two at most).

Each character string to be located is highlighted in yellow on a hard copy page. To find the desired location, do the following: Issue a "/" to ned; that will enable the search command. Beginning at the highlighted letter for each problem, type in a number of characters of the sequence that you feel would be unique enough to fix the cursor at the highlighted character on the first try (ned fixes the cursor at the

first character of the string given in the search command). A point to remember about searching strategy: The longer the search string given to ned (e.g.- number of characters in the line to be located), the greater the chance of finding the text on the first try. GIVE EXAMPLE. If you make a mistake, you may use the backspace key to erase characters. Also, please check your search string before hitting the return key so that typos won't hurt your performance.

On the table in front of you are 2 items: a stack of hard copy sheets from the program to edit and the computer terminal. Once I've initialized ned and created a data file for you, please follow these instructions. If you have any questions, ask them after we read through them once:

Editing Experiment Instructions

- [a] The text character to be located is highlighted in yellow (one per page). Please do them in order. Remember to begin the search string with the highlighted character.
- [b] Please try not to spend more than approximately 15 seconds/problem.
- [c] After you have reset the editor to the top of the file (using the HOME command), use the search command to locate the indicated text. Your strategy should be such that the character string that you give to the editor will position the cursor at the highlighted letter on the issuance of the first search command. Remember, an efficient strategy will allow you to issue a string that is short but unique; this will help you to find the string on the first try.
- [d] If you have located the indicated string on the first try, you will notice a `***` on the far right of the indicated line. THAT'S IT!!!. Reset the cursor position to the top of the file and go on to the next character string to locate.
- [e] If you didn't find it on the first try, reset the cursor to the top of the file and try again. This time, be more careful to issue a string that is more exact (unique) so that the editor will be able to find the particular place you are looking for.
- [f] Continue this process until you reach the end of the stack of hard copies. Then call for assistance.

ALWAYS REMEMBER TO RESET THE CURSOR TO THE TOP AFTER EACH PROBLEM

!!! IF YOU FORGET, SEND A BOGUS SEARCH STRING AND THEN RESET. NOTE: If there are two occurrences of the string on the line, make sure that the cursor is positioned on the right instance of the string. You can check this by comparing the hardcopy to the screen. If you're unsure, ask.

NOTES:

1. ALWAYS remember to reset the cursor to the top of the file after every search attempt using the "HOME" command.
2. USE all of the information given on the hardcopy for each string to be located (e.g.-- if a page number is given, consider this information in the context of the text string you are looking for. Also, if a line number is given at the top of the page, this number will represent the relative line number of the highlighted text).
3. There are only 3 commands -- HOME("HOME"), SEARCH("/"), and QUIT(Ctrl-Z).
4. Use backspace to correct command line errors.
5. ONLY retry each problem ONCE, if you fail the second time, go on to the next problem. (no big deal).
6. All letter input is translated to upper case.
7. USE your knowledge of the program you've learned (e.g.-- ned.f) to locate the text you've seen before by considering where the text is in the file.

8. TRY not to take longer than 15 seconds per problem!

9. If you need help, ASK!!! Remember, you are very important to the success of this experiment.

10. Any observations you make during the experiment that you think I should know tell me after the experimental session. By the way, 0 and O are different characters to ned. A capital O has squared corners.

Thank You.

Rob.

INSTRUCTIONS FOR ESTIMATION TASK

This task is short and simple, it will test your memory for certain word (character sequence) frequencies in the program source code files that you have been editing (from last task). I will activate the program for you, and set up a data file. You will then be prompted to do the following: The computer will present you with a series of sentences (or sentence fragments) from both the program I've asked you to learn and an unknown program text (one program at a time). Based on your relative knowledge of the given sentence, estimate the number of times that you have seen the highlighted text in the indicated program. Each sentence will be presented by the computer for only <4> seconds, so don't think about it too long. Just give your best estimate.

- [1] "Hit any key to begin" This will begin the program.
- [2] If you entered a wrong number before 4 seconds is up, you can use the backspace key to correct your mistakes.
- [3] If you have not seen the text before, give your best estimate of how many times you think the string occurs in a program of the same size as the one I've asked you to learn.
- [4] Once all of the text lines from one program have been displayed, the program will switch input programs and prompt you to "Hit any key to begin" again. Once you hit a key, the process described above will be repeated with the other program that you've edited in the editing task.

REMEMBER: the sentences will only be displayed for a short period of time. ANSWER QUICKLY!!!

REMEMBER: You MUST hit RETURN at the end of your data input for the program to record your answer!!

When the computer signals the end of the task (e.g.- THE END.), you're finished! Please let me know. I should be in the next room. Thanks!!

APPENDIX D

SOURCE CODE FOR ALTERNATE MODEL

```

/*****
*   C-source code for SPOCK, the model simulating the entire   *
*   editing environment.  Written: Spring 1986                 *
*****/

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <assert.h>

#define TABLESIZE 16383
#define FALSE 0
#define TRUE 1
#define ARRSZ 50
#define MAXCHAR 50

struct namestruc { /* holds the N-gram */
    char name[16];
    struct namestruc *across;
    struct charstruc *down;
    long freq;
};

struct charstruc {
    char c;
    struct charstruc *next;
    long cfreq;
    long charused;
};

/* size hash table */
static struct namestruc *hashtable[TABLESIZE];

struct namestruc *lookup();
struct namestruc *addname();
struct charstruc *addchar();

long ourrand();

```

```

long simulate(),hash();
int mysend();

int    LEVEL;
char   buffer[ARRSZ];
int    keylen,bufferlen;
char   key[ARRSZ],begstr[ARRSZ],nubegstr[ARRSZ];
int    found,runs;
long   seedx;

main(argc,argv)
int argc;
char *argv[];
{
    int p,dist,i;
    char name[ARRSZ],inbuf[25],c;
    long freq,unistruc;
    FILE *fp,*fpd;
    struct namestruc *zorch;

    unistruc = 0;
    if(argc < 3){
        printf("Usage- spock <inputdata> <LEVEL> <#runs> <datafile>0);
        exit(-1);
    };
    if((fp = fopen(argv[1], "r")) == NULL) {
        printf("Cant open input file.0);
        exit (-1);
    };

    sscanf(argv[2],"%d", &LEVEL);
    sscanf(argv[3],"%d",&runs);

    while(!feof(fp)) {
        readstruct(fp,name,&c,&freq);
        if((zorch=lookup(name))== NULL) {
            unistruc++;
            addname(name,c,freq);
        }
        else {
            addchar(zorch,c,freq);
        }
    };
    printf("number of unique namestrucs = %d0, unistruc);
    printf("input file = %s0,argv[1]);

    printf("LEVEL=%d0,LEVEL);
    printf("runs=%d0,runs);
    if((tpd = fopen(argv[4], "r")) == NULL) {
        printf("Cant open data file.0);

```

```

        exit (-1);
    };
    fgets(inbuf,MAXCHAR,fpd);
    inbuf[strlen(inbuf)-1] = '\0';
    strcpy(begstr,inbuf);
    printf("beginning string =%s0,&begstr[0]);
    for(i=0;i < LEVEL;i++)
        strcpy(&nubegstr[i],&begstr[i]);
    fgets(inbuf,MAXCHAR,fpd);
    inbuf[strlen(inbuf)-1] = '\0';
    sscanf(inbuf,"%d",&seedx);
    printf("random seed = %d0,seedx);

for(;;) {
    found = 0;
    fgets(inbuf,MAXCHAR,fpd);
    inbuf[strlen(inbuf)-1] = '\0';
    sscanf(inbuf,"%d", &dist);
    if( feof(fpd)) break;
    printf("distance = %d0,dist);
    fgets(inbuf,MAXCHAR,fpd);
    inbuf[strlen(inbuf)-1] = '\0';
    strcpy(key,inbuf);
    printf("key =%s0,&key[0]);

    for(p=1; p <= runs; p++) {
        if(simulate(nubegstr,key,dist))
            found++;
    }
    outfile();
};
}

readstruct(input,name,c,freq)
char    name[], *c;
FILE    *input;
long    *freq;
{
    long    temp;
    int     nameinx;

    nameinx = 0;
    do {
        fscanf(input,"%ld",&temp);
        name[nameinx++] = temp;
    }
    while (temp != 0);
    fscanf(input,"%ld",&temp);
    *c = temp;

```

```

fscanf(input,"%ld 0,&temp);
*freq = temp;
}

struct namestruc *lookup(name)
char *name;
{
    struct namestruc *inx;

    for (inx = hashtable[hash(name)]; inx != NULL; inx = inx->across)
        if (strcmp( name, inx->name) == 0)
            return(inx);
    return(NULL);
}

long hash(name)
char *name;
{
    long tabval;
    int i;
    int len;

    /* for( i = 0;name[i];i++) {
        tabval += (int)name[i] * (int)name[i];
    }
    if(tabval < 0)
        tabval = -tabval;
    return(tabval % TABLESIZE); */

    len = strlen(name);
    for(tabval = (int)name[0] % TABLESIZE, i=1; i < len; ++i){
        tabval = (tabval * (int)name[i]) % TABLESIZE;
    }
    return(tabval);
}

struct namestruc *addname(name,c,tfreq)
char *name, c;
long tfreq;
{
    struct namestruc *ptr;
    char *malloc();
    int tabval;

    ptr = (struct namestruc *) malloc(sizeof( *ptr));
    if ( ptr == NULL)
        return (NULL);
    if ((strcpy(ptr->name, name)) == NULL)
        return(NULL);

    tabval = hash(ptr->name);

```

```

ptr->across = hashtable[tabval];
ptr->down = NULL;
addchar(ptr,c,tfreq);
hashtable[tabval] = ptr;
return(ptr);
}

```

```

struct charstruc *addchar(nptr,newc,cfreq)
struct namestruc *nptr;
char newc;
long cfreq;
{
char *malloc();
struct charstruc *cptr;

```

```

cptr = (struct charstruc *) malloc(sizeof(*cptr));
cptr->next = nptr->down;
nptr->down = cptr;
cptr->c = newc;
cptr->cfreq = cfreq;
/* cptr->charused = 0; /* initialize counter to zero */
nptr->freq += cptr->cfreq;
return(cptr);
}

```

```

long simulate(startstr,inkey,num)
char startstr[ARRSZ], inkey[ARRSZ];
int num;
{
struct charstruc *chptr;
struct namestruc *nm;
char worker[ARRSZ];
long pick;
int i,z;
long iter;

```

```

bufferlen=0;
for(z=0; inkey[z] != '\0'; ++z)
;
keylen = z;
for(i=0; i < ARRSZ; ++i)
    buffer[i] = '\0';

```

```

strcpy(worker,startstr);
worker[LEVEL] = '\0';
for(iter = 1; iter <= num; iter++) {
    if((nm = lookup(worker)) == NULL){
        printf("string with no successor =%s",worker);
        return(0); /* simulate EOF */
    }
    /*pick = (ourrand(&seedx) % nm->freq) + 1; */
}

```

```

        pick = ((ourrand(&seedx) / 2147483648.0) * nm->freq + 1.0);
        chptr = nm->down;
        while((pick == chptr->cfreq) > 0) {
            chptr = chptr->next;
        }
        strcpy(&worker[0],&worker[1]);
        worker[LEVEL-1] = chptr->c;
        chptr->charused++; /* increment character counter */

        if(mysend(chptr->c)) {
/*            printf("key found at iter = %d0,iter);  /**/
            return(iter);
        }
    }
    return(0);
}

mysend(c)
char c;
{
    if      (bufferlen < keylen)
        buffer[bufferlen++] = c;
    else    {
        strcpy(&buffer[0],&buffer[1]);
        buffer[keylen - 1] = c;
    }
    if(bufferlen >= keylen)
        return(!strcmp(key,buffer));
    else return(FALSE);
}

long ourrand(IX)
long *IX;
{
    long k1;

    k1 = (*IX) / 127773;
    *IX = 16807 * (*IX - k1 * 127773) - k1 * 2836;
    if (*IX < 0) *IX = *IX + 2147483647;
    return (*IX);
}

outfile()
{
    printf("0 found the string %s %d times out of %d 0,
        &key[0],found,runs);
    fflush(stdout);
}

```

REFERENCES

- Barnard III, G.A. "Statistical Calculation of Word Entropies for Four Western Languages", IRE Transactions - Information Theory, 1955, pp. 49-53.
- Card, S.K. "Visual Search of Computer Command Menus", in Attention and Performance X: Control of Language Processes. Hillsdale, NJ: Erlbaum Associates (eds- Bouma, H., and Bouwhuis, D.G.), 1982, pp. 97-108.
- Card, S.K., Moran, T.P., and Newell, A. "The GOMS Model of Manuscript Editing", in The Psychology of Human-Computer Interface, Hillsdale, NJ:Lawrence Erlbaum Associates, 1983, pp. 139-189.
- Card, S.K., Moran, T.P., and Newell, A. "The Keystroke Level Model", in The Psychology of Human-Computer Interface, Hillsdale, NJ:Lawrence Erlbaum Associates, 1983, pp. 259-297.
- Cover, T.M., and King, R.C. "A Convergent Gambling Estimate of the Entropy of English", IEEE Transactions on Information Theory, Vol IT-24 (4), July 1978, pp. 413-421.
- Edmundson, H.P. "Mathematical Models of Text", Symposium on Empirical Foundations-Information and Software Science, Atlanta, GA, Fall, 1982.
- Edwards, A.L. Experimental Design in Psychological Research, NY: Harper & Row, Publishers, 1985.
- Ehrlich, S.P., Damon, K., and Cooper, W.E. "Knowledge of Word Frequency as an Aid for Text Editing", in Cognitive Aspects of Skilled Typewriting, NY:Springer-Verlag, Inc., 1983, pp. 283-304.
- Hammer, J.M. Unpublished Working Paper, Spring, 1984.
- Hammer, J.M., and Rouse, W.B. "The Human as a Constrained Optimal Editor", in IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-12 (6), November/December, 1982.
- Mace, A.E. Sample Size Determination, NY:Reinhold Pub. Co., 1964.
- Markov, A.A. "Essai d'une recherche statistique sur le text du roman Evengi Onegin", Bull. Acad. Imper. Sci., St. Petersburg, 1913.

- McClelland, J.L., and Rumelhart, D.E. "An Interactive Activation Model of Context Effects in Letter Perception: Part 1. An Account of Basic Findings", Psychological Review, September, 1981, Vol 88(5), pp. 375-408.
- Montgomery, D.C. Design and Analysis of Experiments, New York, NY: John Wiley and Sons, 1984.
- Ostle, B. Statistics in Research, Ames, IA: The Iowa State University Press, 1963.
- Shannon, C.E. "Prediction and Entropy of Printed English", Bell Systems Technical Journal, 1951, Vol 30, pp. 50-65.
- Yavuz, D. "Zipf's Law and Entropy", IEEE Transactions on Information Theory, September, 1974, p. 650.
- Zipf, G.K. Human Behavior and the Principle of Least Effort. Reading, MA: Addison-Wesley, 1949.

NATIONAL SCIENCE FOUNDATION
Washington, D.C. 20550

FINAL PROJECT REPORT
NSF FORM 98A

PLEASE READ INSTRUCTIONS ON REVERSE BEFORE COMPLETING

PART I—PROJECT IDENTIFICATION INFORMATION

1. Institution and Address Georgia Tech Research Corporation Atlanta, GA 30332	2. NSF Program	3. NSF Award Number IST-8217440
	4. Award Period From 1/15/83 To 6/30/86	5. Cumulative Award Amount \$74,506

6. Project Title

"Models of Human Performance Using Text Editors (Information Science)"

PART II—SUMMARY OF COMPLETED PROJECT (FOR PUBLIC USE)

The results of this research increase our understanding of human-computer interaction and of experimentation with human subjects who are doing complex tasks. The human-computer interaction results include first, observations on the use of a text editor that kept the user in continuous control of the positioning process and second, experiments and models of user selection of search string keys during editor positioning. The results on experimentation include a study of the power of statistical tests from experiments on computer programming and a way of testing the significance of production rule models of human problem solving.

PART III—TECHNICAL INFORMATION (FOR PROGRAM MANAGEMENT USES)

1. ITEM (Check appropriate blocks)	NONE	ATTACHED	PREVIOUSLY FURNISHED	TO BE FURNISHED SEPARATELY TO PROGRAM	
				Check (✓)	Approx. Date
a. Abstracts of Theses		X			
b. Publication Citations		X			
c. Data on Scientific Collaborators	X				
d. Information on Inventions	X				
e. Technical Description of Project and Results		X			
f. Other (specify)					
2. Principal Investigator/Project Director Name (Typed) John M. Hammer	3. Principal Investigator/Project Director Signature			4. Date 1-15-88	

PART IV - SUMMARY DATA ON PROJECT PERSONNEL

NSF Division _____

The data requested below will be used to develop a statistical profile on the personnel supported through NSF grants. The information on this part is solicited under the authority of the National Science Foundation Act of 1950, as amended. All information provided will be treated as confidential and will be safeguarded in accordance with the provisions of the Privacy Act of 1974. NSF requires that a single copy of this part be submitted with each Final Project Report (NSF Form 98A); however, submission of the requested information is not mandatory and is not a precondition of future awards. If you do not wish to submit this information, please check this box ☐

Please enter the numbers of individuals supported under this NSF grant.
Do not enter information for individuals working less than 40 hours in any calendar year.

*U.S. Citizens/ Permanent Visa	PI's/PD's		Post- doctorals		Graduate Students		Under- graduates		Precollege Teachers		Others	
	Male	Fem.	Male	Fem.	Male	Fem.	Male	Fem.	Male	Fem.	Male	Fem.
American Indian or Alaskan Native												
Asian or Pacific Islander												
Black, Not of Hispanic Origin												
Hispanic												
White, Not of Hispanic Origin												
Total U.S. Citizens												
Non U.S. Citizens												
Total U.S. & Non- U.S. . .												
Number of individuals who have a handicap that limits a major life activity.												

*Use the category that best describes person's ethnic/racial status. (If more than one category applies, use the one category that most closely reflects the person's recognition in the community.)

AMERICAN INDIAN OR ALASKAN NATIVE: A person having origins in any of the original peoples of North America, and who maintains cultural identification through tribal affiliation or community recognition.

ASIAN OR PACIFIC ISLANDER: A person having origins in any of the original peoples of the Far East, Southeast Asia, the Indian subcontinent, or the Pacific Islands. This area includes, for example, China, India, Japan, Korea, the Philippine Islands and Samoa.

BLACK, NOT OF HISPANIC ORIGIN: A person having origins in any of the black racial groups of Africa.

HISPANIC: A person of Mexican, Puerto Rican, Cuban, Central or South American or other Spanish culture or origin, regardless of race.

WHITE, NOT OF HISPANIC ORIGIN: A person having origins in any of the original peoples of Europe, North Africa or the Middle East.

THIS PART WILL BE PHYSICALLY SEPARATED FROM THE FINAL PROJECT REPORT AND USED AS A COMPUTER SOURCE DOCUMENT. DO NOT DUPLICATE IT ON THE REVERSE OF ANY OTHER PART OF THE FINAL REPORT.

FINAL REPORT

NSF GRANT IST-82-17440

John M. Hammer

Principal Investigator

November 1987

This section of the final report is an overview of the four articles included as the substance of the report. These articles are:

Hammer, J.M., A display editor with random access and continuous control, *International Journal of Man-Machine Studies*, Vol. 21, 1984a.

Hammer, J.M., Statistical methodology in the literature on human factors in computer programming, in Human-Computer Interaction, G. Salvendy and M. Oshima, eds., Elsevier Publishing Co., 1984b.

Lewis, C.M. and Hammer, J.M., Significance testing of rules in rule-based models of human problem solving, IEEE Transactions on Systems, Man, and Cybernetics, 16(1), 1986.

Andes, R.C., Jr., An Information-Theoretic Model of Human Search String Selection in Text Editing, M.S. Thesis, Center for Man-Machine Systems Research, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia, 1987.

[Hammer 1984a] is a report on an editor designed to keep the user in continuous control of the positioning process. This work was begun out of frustration with a primitive display editor which could produce so much display output that it could lag the user's commands by a second or more, even at then high baud rates. While the description of this editor may make it at first appear similar to many others of its era, it did in fact take

advantage of or make allowances for many aspects of human performance that many display editors still do not accommodate today.

[Hammer 1984b] is the result of an experimental failure, which lead to an exploration of how experiments should be conducted with human subjects. One uncertainty in this new field was the degree of variability in human subjects. Since there were a number of articles on human performance in computer programming, this field was selected for an examination of successful and unsuccessful practices. Many researchers in this field had claimed that programming ability differed widely between individuals. The statistical results in the literature at that time do not support that claim. Instead, most of the results were found to be as large (in terms of variance explained) as were found in the "harder" areas of psychology (such as traditional experimental psychology and human factors psychology).

[Lewis and Hammer 1986] describes several methods for statistical significance testing of rule-based models. A typical evaluation of a production rule model of human problem solving was to point out the percentage agreement between model and subject actions. This article describes three methods (ANOVA, chi-square, and randomization tests) that can evaluate the significance of each rule. The conclusion of the article describes a problem which remains an research interest of both authors: identification of a rule-based model of cognition from

data. Significance testing, even as described in the article, does not guarantee identification. The article describes a paradigm through which this question could be studied.

[Andes 1987], a thesis directed by the principal investigator, describes a model of how the human chooses a search string to move an editor to a desired location. The model posits two processes. First, the human must estimate the number of bits of information in the text between the current and desired editor position. Second, the human must choose a search string with at least this many bits. The model was able to predict human success and failure with at least 90% accuracy.