10/ 10-16-89 JS 1

UCID- 21770

A STRUCTURED HISTORY FOR COMMAND RECALL

Meera M. Blattner
E. Eugene Schultz
Jeremy Y. Uejio

University of California, Davis, and
Lawrence Livermore National Laboratory
Livermore, California

Masumi Ishikawa

Electrotechnical Laboratory
Tsukuba, Japan

Lawrence Livermore National Laboratory

A Structured History for Command Recall

Jeremy Y. Uejio, Meera M. Blattner, E. Eugene Schultz

University of California, Davis, and
Lawrence Livermore National Laboratory
Livermore, California

and

Masumi Ishikawa

Electrotechnical Laboratory
Tsukuba, Japan

Topics:
UNIX support systems, user interfaces, PDP models, artificial intelligence, history

ABSTRACT

The purpose of a history is to provide access to previously used commands. A structured history is a database of commands that were previously executed by a user, together with methods for storing and recalling these commands. Two models and their implementations are presented here. The first model stores all commands eliminating duplicates and short commands. Retrieval is by pattern matching of several different types, frequency, and time. The second model is a parallel distributed processing model that presents the user with a list of likely candidates for good matches to an input string. The PDP history allows for misspellings and retrieves a list of related commands. The implementations are compared.

Please send correspondence to:

Meera M. Blattner, L-540
Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, CA 94550

Phone: (415) 422-3505
Netmail: blattner@crg.llnl.gov

A Structured History for Command Recall

Jeremy Y. Uejio, Meera M. Blattner, E. Eugene Schultz

University of California, Davis, and
Lawrence Livermore National Laboratory
Livermore, California

and

Masumi Ishikawa

Electrotechnical Laboratory
Tsukuba, Japan

UCID--21770

DE90 000904

Abstract

The purpose of a history is to provide access to previously used commands. A structured history is a database of commands that were previously executed by a user, together with methods for storing and recalling these commands. Two models and their implementations are presented here. The first model stores all commands eliminating duplicates and short commands. Retrieval is by pattern matching of several different types, frequency, and time. The second model is a parallel distributed processing model that presents the user with a list of likely candidates for good matches to an input string. The PDP history allows for misspellings and retrieves a list of related commands. The implementations are compared.

Please send correspondence to:

Meera M. Blattner, L-540
Lawrence Livermore National Laboratory
P.O. Box 808
Livermore, CA 94550

Phone: (415) 422-3505
Netmail: blattner@crg.llnl.gov

Topics: UNIX support systems, user interfaces, PDP models, artificial intelligence, history

[illegible]

A Structured History for Command Recall

Jeremy Y. Uejio[1], Meera M. Blattner[2], E. Eugene Schultz[3]

University of California, Davis, and Lawrence Livermore National Laboratory

and

Masumi Ishikawa

Electrotechnical Laboratory

ABSTRACT

The purpose of a *history* is to provide access to previously used commands. A *structured history* is a database of commands that were previously executed by a user, together with methods for storing and recalling these commands. Two models and their implementations are presented here. The first model stores all commands eliminating duplicates and short commands. Retrieval is by pattern matching of several different types, frequency, and time. The second model is a parallel distributed processing model that presents the user with a list of likely candidates for good matches to an input string. The PDP history allows for misspellings and retrieves a list of related commands. The implementations are compared.

1. INTRODUCTION

1.1. Overview

The role of memory in computer usage has traditionally been limited to storing either partial results of computations within programs or large banks of data. In some cases, a simple log either of commands or data is available. Imagine the difficulty people would have in attending to daily tasks if their memory was so limited! Yet, in performing a task on the computer, the user largely relies on his or her own memory to recall system interactions. To increase the usability of complex

[1,2,3]Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.
Meera Blattner is also with the Department of Biomathematics, M.D. Anderson Cancer Research Center, Univerisity of Texas Medical Center, Houston, Texas.

interfaces, software that stores and recalls our interactions with the computer is required. Command recall is not of much assistance to the user if there are an insufficient number of command attributes for retrieval or if the list does not extend far back into time.

This article describes the use of a *structured history* to assist users in recalling previously entered complex UNIX commands. The UNIX command language can be difficult to use due to its inconsistent syntax, its non-mnemonic command names, and its sometimes lengthy commands (Norman, 1981). It is often difficult to recall and re-execute previously entered commands. To alleviate this problem, an enhancement of the existing history mechanism, a structured history, was undertaken. A structured history is a database of commands that were previously executed as well as a method of saving and recalling these commands. Two models are examined: the first model uses a conventional database and the second model uses a parallel distributed processing system. The first was created in the Emacs environment and saves only complex commands that can be recalled by pattern matching on command name, pattern matching on command options, frequency of use, and relative time. The parallel distributed processing system (the PDP history) stores complex UNIX commands by decomposing them into two letter sequences called bigrams. A two-layer network, in which each unit represents a bigram, was used. A prototype implementation with features such as spelling correction and associated command rec[illegible]ated.

1.2. The UNIX user interface

The UNIX operating system, developed by Bell Laboratories nearly 20 years ago (Franklin, 1987), has become extremely popular in the academic and business world. The modular design and tool-box philosophy make UNIX an ideal programming environment for the expert user. However, there are a number of features that make UNIX difficult to use for both novice and expert users.

The bulk of user input consists of a series of command lines. Each line executes a function such as editing a file, compiling a program, or reading electronic mail. For example, a simple session could be the following:

```
mail              - execute the electronic mail program
ls  -l            - display a listing of files in long format
rm dead.letter    - remove the file called "dead.letter"
logout            - exit UNIX.
```

A command line can be divided into two parts: a command name, which is the first word and is the name of the program executing the function, and the command options, which are the rest of the command. The options consist of arguments for the command name and filenames. By using pipes a user can execute several commands on a single command line.

The user interface is implemented by a special program called the "shell." The shell interprets the user's input, controls input and output of user's processes, and provides various programming features. There are three common UNIX shells—the standard AT&T Bourne shell, the Berkeley C shell, and an extension of the Bourne shell, the Korn shell (Kochan & Wood, 1985). These and other shells provide various functions to assist the user in entering commands. Among such tools are : a command history, command aliases, on-line help (man pages), command name and file name completion, command line editing, which allows the user to edit a command line before executing it, and spelling correction (Sebes, 1987).

One of these tools, the command history, is designed to assist in command recall. One of the shells, the C shell, saves all the commands chronologically in the history list as they are entered by the user. The user can display the history list and choose commands to combine or re-execute. Examples of some history commands are:

```
!!        -   execute the last command
!10       -   execute the 10th command in the history
!9:0-1    -   execute the command name and the first word of the ninth command
```

Only a selected number of the most recent commands in the history list can be saved between login sessions. Typically this number is small (less than 100). The UNIX history was probably based on the assumption that users would only want to recall recently executed commands and would not need to save old commands.

Norman (1981) observes that the command language for UNIX has an inconsistent syntax and many non-mnemonic command names. Some command names may have no relation to the function that they execute. For example "grep" is a program to search a file for a pattern, "awk" is a pattern matching and scanning language (named after the authors—Aho, Weinberger, and Kernighan), and "troff" is a program which formats files for printing.

In addition, UNIX commands tend to be lengthy because they contain many options and sometimes several commands. (Unfortunately this tendency is inherent in the toolbox philosophy of UNIX.) Some examples from sample sessions are shown below:

```
cat p313.data | graph -g1 -c o -l 'Problem 3.15' | psplot | lpr -h
find /u0 -name latex -print > latex.psn &
awk '{print substr ($0,8)}' oldhistory | sort | awk -f freq.awk
cc -O -Dstrrchr=rindex -s -o hier hier.c sftw.o
```

Note the inconsistent arguments such as "-c," "-print," and "-Dstrrchr=rindex." The syntax for each command name can be quite demanding and can make it difficult for the user to compose and remember complex and infrequently used UNIX commands.

The following is a detailed example of a user executing a typical, lengthy, difficult command. Suppose the user wants to print part of a file to a PostScript Laser printer. A possible UNIX command to do this is:

```
ptroff -me -PT4387 -o1-2 introduction.tx
```

in which introduction.txt is the file to be printed. If, a few weeks later, the user wishes to format another file, but has completely forgotten the above command, the existing UNIX tools for assisting the user are not helpful. The command history could not be used because: 1) the user does not remember exactly what command was used, 2) when the command was last executed, and 3) the command is no longer in the history list (due to the finite size of the list). Even if the command were in the list, it would be difficult to find, due to both the length of the list and the simple methods of recalling events. Other tools such as command aliases would not be useful since the user did not explicitly create an alias for the command after it was first executed. The user would have to revert back to the man pages and once again spend time determining the requirements and syntax for this complex command.

1.3. Related work to improve the history mechanism

Greenberg and Witten (1988) conducted a study to determine principles for the design of history mechanisms. They recorded the commands entered by 168 users with various experience levels using the C shell over a period of four months. They found that there is a 43% chance that the next command occurred within the previous seven commands, a 31% chance that it occurred further back in the history, and a 26% chance that the next command line has not appeared before. (The investigators also found that just over half of the subjects used the C shell history to recall commands, but for only 4% percent of their total commands. Users found the history syntax difficult and not worth the trouble.) Greenberg and Witten found that even extending the history to 25 items would only increase the chance of the next command occuring in the history to a little under 60%. There is still a 15% chance that the next command was previously entered, but no longer in the history of recent commands.

There is also the likelihood that many new commands are variations of old commands and should be considered as previously entered. For example, in the previous "ptroff" command, the user might enter:

```
ptroff -ms -PT4387 -o1-2 introduction.txt —use the "ms" macros
ptroff -me -Plw1 -o1-2 introduction.txt —use a different printer
ptroff -me -PT4387 introduction.txt     —print the entire file
```

All these commands would be counted as new commands in the Greenberg and Witten study. However, the user is more likely to think of these as variation of the same command. In spite of these difficulties, from their study they conclude that history mechanisms are useful and, with careful design, can recall the majority of commands entered by the user.

Another study designed to extract features of UNIX commands was made by Ishikawa (1987, 1989b). Although restricted to command names, a PDP model was developed to obtain frequencies of not only commands, but also command subsequences to assist users by locating candidates for "macro" commands. There are several studies on general command reuse not related to UNIX. For example, Yang (1988) describes a conceptual model for user recovery and command reuse, Vitter (1984) has applied his US&R package for undo, skip, and redo to a graphics layout system, and Common Lisp defines a simple mechanism to recall forms (Steele, 1984). However, these studies discuss only simple methods of recalling commands, such as chronological lists, and do not consider a more general approach.

2.0. THE ORGANIZATION OF THE STRUCTURED HISTORY

2.1. Definition of a structured history

The purpose of a structured history, like the existing UNIX history, is to provide quick easy access to UNIX commands that were entered by the user. The existing UNIX history is a very simple database. Each command line is a data item, all commands are saved, and there are only two ways to access commands: by relative time (position in the history list) or by pattern matching with the first few characters. However, unlike the existing history, a structured history has a more complicated database and saves commands in an organized manner, and not as a simple list, therefore providing a variety of strategies for recalling commands. It also saves most of the commands entered by the user, not just the recent commands. In the section below, there is a description of a structured history we call "conventional structured," or the CS history, to distinguish it from the PDP structured history.

2.2. Design of the CS history

In the design of the CS history each command is broken into parts and attributes. The two parts of a command are the command name and the options. The attributes of the command can be frequency of use, files accessed, time executed, keyword descriptions, etc. This allows the user to be more specific about the command he wishes to recall. For example, the user may ask for "the most frequent command which accessed the file temp.dat." The design of the CS history consists of a database for commands, a saving method, and a recalling method. The database for previously entered UNIX commands has the following structure:

- Each command is composed of two parts: the command name and the command options (arguments, filenames, etc.).
- Each command contains two attributes: the number of times the command was previously executed (command frequency) and the relative time that the command was executed (similar to the existing C shell history).

Two principles are followed when saving a command.

1) Only complex commands are saved.

There are many different measures of command complexity. The number of words, the number of options, the length of the command line, or a combination of the above could be used. Some commands, such as "ls" or "cd," are simple enough to remember and re-enter without the help of a history mechanism, thus do not need to be saved. Other commands, such as the "ptroff" example in chapter one, are difficult to remember and should be saved in the history. As a first approximation, the length of the command line was used to measure complexity—only lines longer than five characters were saved in the CS history.

2) Only one copy of a command is saved.

The existing UNIX history saves a command, even if it was executed many times before. Duplicate commands cause the history to grow in length but not in information content. For example, suppose a user enters the "ptroff" command mentioned earlier. Then the user debugs a C program by repeatedly editing the program, running the compiler, then executing the program (i.e. executing the three commands "vi test.c," "cc test.c," and "a.out"). The history list will now be filled with many copies of these three commands. Since the list is of finite (and often of short) length, the "ptroff" command will soon be removed from the list and the user will no longer be able to recall that command. The CS history, however, will save a command only once, regardless of the number of times it has been executed. (The frequency of use will be updated each time). The command will be saved chronologically at the latest time it was executed. In the above example, the history list will contain the four commands, "vi test.c," "cc test.c," "a.out," and the "ptroff" command.

The above two principles were applied to a sampling of commands taken over a period of several months from the history logs of three experienced UNIX users. These history logs were reduced by at least 80%! The users together executed over 20,000 command lines, but only a little over 3,000 were both unique and over five characters long.

There are two ways a user can recall these saved commands: as a single command or as a list of commands. The single command method is used to recall the last command containing a specified string or, if no string was specified, to recall the last command executed. The list of commands method is used to recall a set of commands which meet a given criterion or are sorted in a given manner. The different methods are listed below.

- Pattern matching using the command name. The user enters a sequence of letters which might be a command name or part of a command name and the system displays all previously entered commands in which the name contains those letters.
- Pattern matching using the command options. The user enters a sequence of letters which might be contained in the command options. Usually, these letters are part of a filename, but they can also be part of the arguments to the command. The system then displays all previously entered commands in which the options contain the specified letters.
- Relative time of last execution. The system displays a list of commands sorted chronologically.
- Frequency of execution. The system displays a list of commands sorted by the number of times the command was used.
- Combination of the above. The user can combine the different methods. For example, the user can display all commands that have options containing "temp.dat," sorted by frequency.

The single command method is useful when the user knows exactly what command to execute, but does not want to re-type the command. (The existing UNIX history provides this method and was probably designed for this purpose.) The list of commands method is useful when the user does not know exactly what command he wants, but can recognize it among similar commands. (The existing UNIX history does not provide this method.)

2.3. Implementation of the CS history

A simple prototype of the CS history was written using Lisp in the GNU Emacs environment. This prototype used conventional data structures and algorithms and allows a user to recall commands by frequency, time, and pattern matching. It was found to be very useful by several of our colleagues. GNU Emacs (Stallman, 1986) provides an easy to use powerful editor and a Lisp programming environment. Emacs includes a shell mode that calls the C shell, thus making it convenient and relatively easy to add a history to the C shell without actually modifying the C shell. Emacs provides functions to call other processes and redirect both input and output for that process. (This feature was used in the PDP implementation to run SunNet from Emacs.) In addition, the Emacs Lisp environment allows incremental coding of Lisp programs and contains an interactive debugger. Emacs facilitates the use of the conventional history by allowing the user to easily scroll and edit.

The basic design of the implementation is shown in Figure 1. The flow of processing is as follows. First, the user executes the "shell" function of Emacs. This function reads the user's history of commands from a predefined file and executes a C shell subprocess for UNIX. All input and output to and from the shell is done thru an Emacs buffer called "*shell*."

Now, the user can: 1) execute UNIX commands by entering them and pressing the return key (<CR>), or 2) execute history commands with the prefix <ctrl>-C and a letter (the "control" key and the "C" key are held down simultaneously, then another letter is pressed) . In the first case, two things happen: the command is added to the history list, and then the command is sent to the shell subprocess and executed. If the command is "exit," then the shell is terminated and the history file is saved.

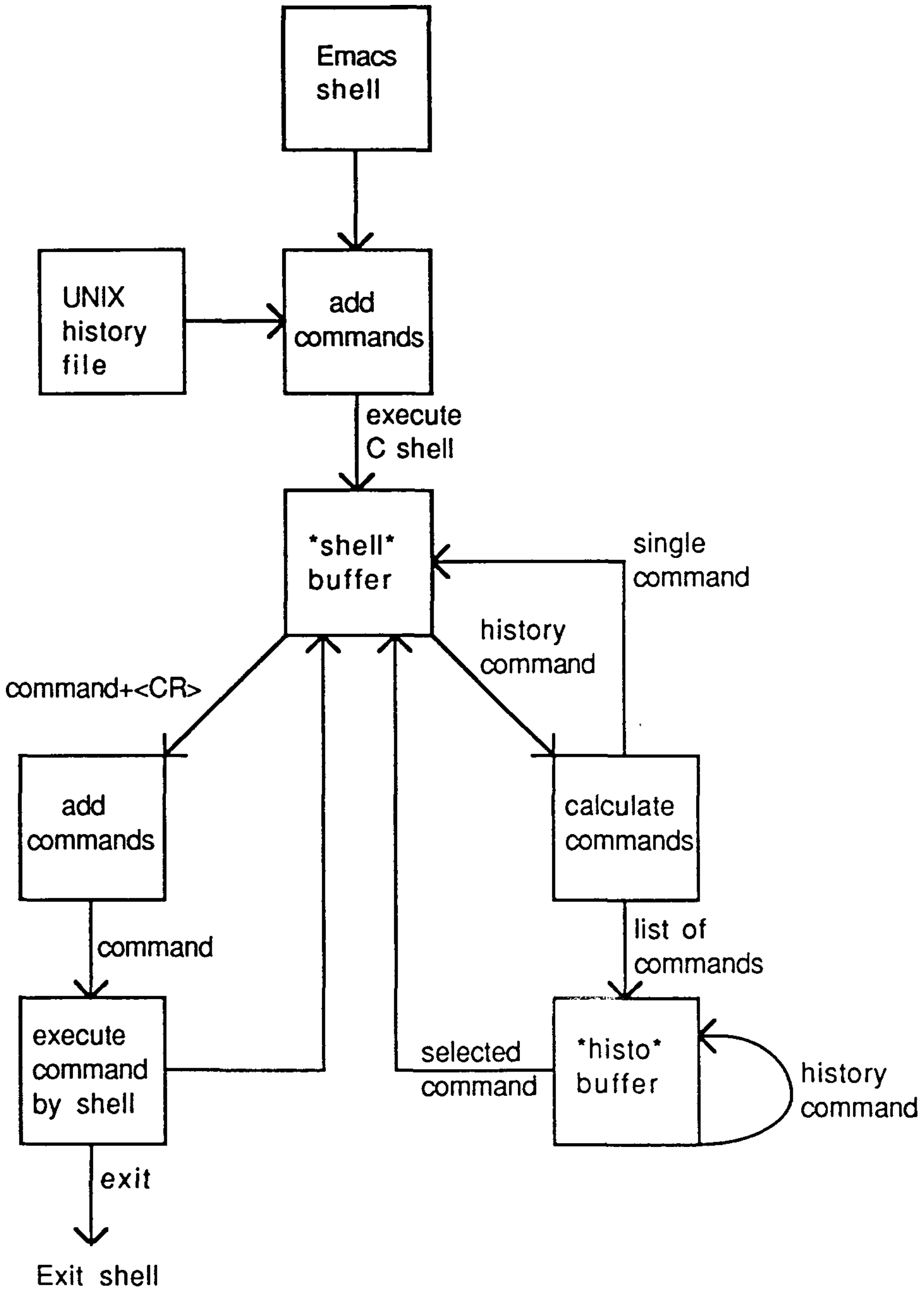

Figure 1: Schematic of the conventional structured history

In the second case, the CS history either displays a single command which the user can edit then execute, or the history calculates and displays a list of commands in another buffer called "*histo*." The user can scroll thru this list and select a command to be executed or apply a history command to display another list of commands. The <ctrl>-C prefix was used because the existing shell-mode of Emacs already uses several <ctrl>-C prefix commands. The CS prototype is described in more detail in Uejio (1989).

Two lists are used to store the previously entered UNIX commands: histo-list and histo-current-list. The histo-list contains all the previously entered commands in chronological order and is not viewable by the user. The histo-current-list, a subset of histo-list, is displayed to the user and contains a list of commands in a specific order. The two lists allow the user to apply various recall methods to histo-current-list without modifying histo-list. For example, suppose the user displays all commands that have options containing the file "temp.dat" ordered by frequency of use. First, all commands in histo-list that contain the string "temp.dat" are copied to histo-current-list. Then histo-current-list is sorted by the frequency property. Finally, the histo-current-list is displayed for the user. The histo-list remains unchanged.

The following history recalling features have been implemented (Note: the current string refers to any characters that the user has entered in the *shell* buffer before pressing the prefix command):

- "<ctrl>-C a"—List all commands in the history chronologically.
- "<ctrl>-C c"—List all commands in which the name contains the current string.
- "<ctrl>-C s"—List all commands in which the options contain a given string.
- "<ctrl>-C f"—List all commands by frequency of use.
- "<ctrl>-C l"—Insert the last command (recall a single command) containing the current string or the last command executed if no current string exists.

In the *histo* buffer, the user may combine some of these selection strategies by re-applying the history commands. For example, if the user wanted all the commands containing the string "temp.dat" arranged by frequency, the user simply enters "<ctrl>-C s temp.dat" from the *shell* buffer and then enters "<ctrl>-C f" from the *histo* buffer.

Also in the *histo* buffer, the user can delete commands from the history. Often when composing a new command, a user will enter several incorrect commands before entering the correct command. The delete function allows the user to remove these unnecessary commands. The features listed above were implemented using the standard list functions of Lisp and special search functions of Emacs.

2.4. A shell-script implementation

A very simple shell-script history using aliases, awk, and sed, was also implemented. This history, like the CS history, saves only one copy of the complex commands entered by the user. The shell history allows the user to recall commands only by pattern matching on the entire command line (rather than on specific parts of the command line). A list of commands is displayed and the user can select a command to execute. No editing is allowed, however. Although this implementation is more limited than the CS history, it does have the advantage that it can be run by any user of the C shell and does not need the Emacs environment.

3.0. THE PDP HISTORY

3.1. Some properties of PDP systems

Those unable to recall a command may remember it approximately, but not in exact detail. For example, the copy command is "cp" in UNIX, however, the user may try "cy" or "copy" instead. Similarly with command options. Some arguments begin with a dash while others do not; some command names are formed from the first two consonants of their respective function names while others use some arbitrary scheme, such as "cat." To provide the greatest possible assistance to the user for command recall, there must be a way of retrieving commands that are not clearly remembered. For this reason, a second implementation of a structured history was made using a parallel distributed processing (PDP) model. In the PDP model (also known as a neural networks or connectionist system) the user simply specifies a string of characters that may be in the command that the user wishes to recall. The PDP history "searches" thru the list of commands and presents the user with a list of likely candidates. For a basic description of PDP systems, we refer the reader to Rumelhart and McClelland (1986), who published the "bible" for PDP models, *Parallel Distributed Processing*.

PDP systems have the following properties:

- *Information is distributed in the connection strengths.* With conventional data structures, information is contained in records or specific slots. In a PDP model, however, information about items is distributed in the magnitude and types of connection strengths throughout the network.
- *Fault tolerant and error correcting.* Since the information is contained in many connection strengths, there is a redundant amount of information. A perfect copy of the input pattern is not needed because a PDP system can overlook certain defects.
- *Pattern completion.* A PDP system can be given an incomplete input pattern and will be able to make a "best guess" and give an appropriate output pattern.
- *Ability to generalize.* A PDP system can learn a series of patterns and implicitly group them by features.
- *Show graceful degradation.* In a hardware implementation, where units are represented by individual CPUs, if a few CPUs fail, then the network can still function. The network will gracefully degrade in performance as more CPUs fail.
- *Find rules and learn by examples.* Rules are learned by training over example patterns and not by explicitly stating parameters. This means that the same network can be used for a variety of applications without the programmer having to describe the details of each application.

3.2. Properties of the PDP history

The PDP history has three major properties:

- it allows for misspellings when users recall a command,

- it recalls several associated commands besides the command that was desired by the user, and
- it presents a sorted list of these commands to the user. The list is sorted by the number of activated bigrams.

The error correcting nature and the ability to learn by examples are desirable features for a UNIX command history. Error correction allows the user to misspell a command or to enter an incomplete command and have the system recall the correct command. The ability to learn by examples allows the system to be automatically configured to each user.

3.3. Design of the PDP history

Recall that a structured history has three parts: a database for commands, a saving method, and a recalling method. In a PDP history, the database is a PDP network, the saving method is a form of learning, and the recalling method is a form of testing the network. In a PDP history, each command is not broken into parts and attributes as in the CS history, but rather into component letters. Each command is composed of two character sequences called bigrams (Kohonen & Venta, 1988). For example, the command "ls -la" is composed of the bigrams: "ls," "s ," " -," "-l," and "la."

Bigrams were used because they hold some sequencing information and also compensate for minor spelling errors by activating several units for a word. For example, suppose the word "emacs" is encoded. Four units are used: "em," "ma," "ac," and "cs." If the user enters "emaxs" instead, then two of the bigrams, "em" and "ma," are still the same. There is still a good chance that "emacs" will be at least partially activated. If single letters are used, then "emacs" would require five units. If the user enters "emaxs" instead, then there is a good chance that "emacs" will be activated, however, since sequencing information is not stored, many other commands with the letters "e," "m," "a," "x," or "s" will also be activated.

Other encodings, such as trigrams (three letter sequences) or full words, would not allow for as many misspellings as bigrams. In the above example, "emacs" would be encoded to only three trigrams: "ema," "mac," and "acs." A simple misspelling such as "emaxs" would only match one trigram, "ema," probably not enough to activate "emacs." In addition, there are more trigrams than bigrams. For 26 letters, there are 26x26 or 676 bigrams, however, there are 26x26x26 or 17,576 trigrams. Therefore, there would be more units and the performance of the network would be slower.

A two layer network with equal numbers of units was used. Each unit in the input layer represents a unique bigram and each unit in the output layer represents a unique bigram. The activation level of the units in the network represents the likelihood that the respective bigram is present in the command. The PDP history decomposes each command in the history into a set of bigrams. A PDP system is then used to associate each of the bigrams, composing a command, to one another. In the recalling phase, the user enters a string of characters (a few bigrams) and the PDP history converts it to a command or list of commands of which the user can select to re-execute. The PDP network is used to "expand" the few bigrams that the user entered into a list of associated bigrams. Then a conventional system converts these associated bigrams into a command or a list of commands. The user can select the appropriate command and edit or execute it.

Saving commands (learning)

Only one rule, command complexity, is used to determine which commands are to be saved and learned by the history. All commands over five characters in length, including duplicates, are saved. The saving or learning phase consists of a converter to encode the list of UNIX commands to bigrams and the PDP network to store the bigrams. Each bigram activates its respective unit. These units then modify the connection strengths between themselves to learn the set of bigrams representing a particular command. This process is applied repeatedly to all commands in the history list until the difference between the presented bigrams and the output of the network is a minimum.

Recalling commands (testing)

The recalling phase consists of an interpreter to convert a user's input to a test pattern of bigrams for the network, the PDP network, another interpreter to convert the output of the network to a list of commands, and finally a routine to send the selected UNIX command to the shell. The user attempts to recall a command by entering a string of characters which might be contained in the command. The user's input is converted to a test pattern consisting of a list of bigrams. The test pattern is then presented to the network, and the network attempts to activate any associated bigrams and returns a list of bigrams.

Since the PDP network has learned sets of bigrams, the incomplete pattern of bigrams that the user entered is likely to activate a complete set (i.e. a specific command). However, the incomplete pattern may be ambiguous so the network will activate several sets of bigrams—in other words, several commands. The system then determines which commands contain a substantial number of these activated bigrams and presents the list of commands to the user. In this way, the user can select the appropriate command, and the system does not have to perfectly predict the user's intended command. This scheme gives a heavier weight to longer commands which contain more bigrams. This is desirable since long commands are generally more difficult to remember. This scheme also returns a list of commands rather than a single command. It is likely that the user's input will be ambiguous and several sets of bigrams will be activated.

3.4. Implementation

A prototype system was implemented using GNU Emacs Lisp and SunNet (a neural network simulator). Emacs was used to encode and decode the set of UNIX commands for the neural network and to provide a user interface to SunNet. Due to time limitations (and other problems discussed later), the PDP history was not actually connected to the UNIX shell. Only the learning phase and the recalling phase were implemented.

SunNet is a general purpose tool for constructing, running and examining a PDP network. It allows you to deal with a network at a fairly high-level of conceptualization, and yet provides the flexibility to construct networks of almost arbitrary structure and size (Miyata, 1987). SunNet allows the user to specify a full description of the network including the number of units, the method of connecting the units, the propagation rule, and the learning rule. SunNet requires two input files: a network file which describes the network, and a pattern file which contains the patterns to be learned.

Figure 2 shows a detailed schematic of the prototype. In the learning phase, a list of UNIX commands is learned. First the unique bigrams are found, then the commands are converted to bigrams and encoded to SunNet input format. Finally, SunNet cycles thru the input many times to learn the commands.

In the recalling phase, the user enters a string of characters then the system replies with a list of possible commands. First, the users input is decomposed into bigrams. These are then encoded to SunNet input format. SunNet tests the network and outputs a pattern which contains the activation levels of the bigrams in the network. The UNIX commands that contain a substantial number of bigrams with a high activation level are displayed. Finally, the user can select the appropriate command from this list.

The pattern file contains a series of input-output training pairs. The input and output patterns are identical (this identifies the network as auto-associative). Patterns can either be the integer values from 0-9, with the letter "a" corresponding to a "10," or floating point numbers. For the PDP history, only a "0," for the minimum value, and an "a" for the maximum value was used. A more detailed description of the implementation can be found in Uejio (1989).

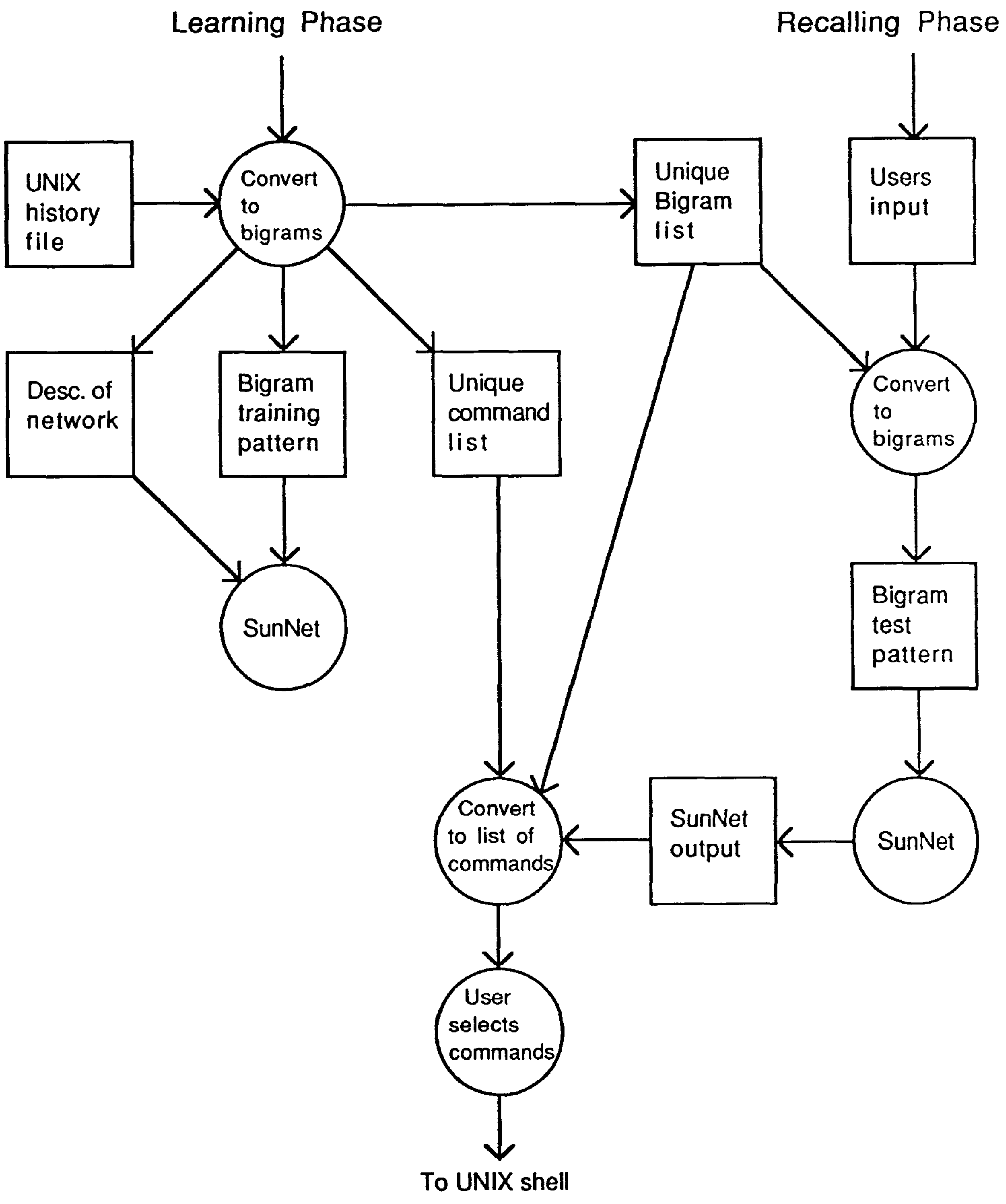

Figure 2: Schematic of the PDP history

4.0. A SAMPLE SESSION COMPARING MODELS

4.1. C shell history

An example comparing the usage of the existing UNIX history, the CS history, and the PDP history is shown in Figure 3. The first column is an excerpt from a log of a user's history taken over a period of several weeks. Roughly 600 command lines were executed, but only 37 of them are shown in this table. The commands are arranged chronologically from the most recent command at the bottom, to the least recent command at the top.

The second column of Figure 3 shows which commands are saved by the existing UNIX history. An "X" means that the command is saved and is accessible to the user. For this example, the C shell environment variable, history, is set to 20 so only the last 20 commands are saved. This variable can be larger, however it is generally set to the length of the screen because the history command itself will simply scroll the entire list on the terminal.

Suppose the user wishes to re-execute the most recent "nroff" command found near the bottom of the command list. Using the C shell history, the user would type "!nroff." However, suppose the user wishes to re-execute the "nroff" command in the middle of the list. Using the C shell history, the user would be unable to access this command since it was executed several weeks ago and is no longer in the history list. (Even if it were in the history list, entering "!nroff" would retrieve the "nroff" command just recently executed and not the desired command.)

4.2. CS history

The CS history allows the user to access commands entered both recently and a long time ago. The third column in Figure 3 shows the commands that are saved by the CS history. All commands, except short commands such as "ls" and duplicate commands such as the first two occurrences of the "logout" command, are saved.

Suppose the user wishes to re-execute the most recent "nroff" command. In the CS history, the user simply types "nroff" and then presses "<ctrl>-C l" for the last command feature. Suppose

Commands	UNIX history	CS history	PDP history
mail			
cat oldhistory		X	X
more oldhistory		X	X
awk '{print substr($0,5)}' oldhistory		X	X
awk '{print substr($0,8)}' oldhistory		X	X
vi freq.awk		X	X
cd thesis		X	X
vi introduction.txt		X	X
ptroff -me -PT4387 -o1-2 introduction.txt		X	X
logout			X
... a few weeks later ...			
nroff -me stacks.p		X	X
nroff -ms stacks.p		X	X
... a few weeks later ...			
tail gif-l*		X	X
tail GIF-L*		X	X
rm GIF-* (		X	X
rm GIF-*		X	X
ls -la			
logout	X		X
mail	X		
ls	X		
cd temp	X	X	X
clear	X		
vi hier.c	X	X	X
cat Makefile.bsd	X	X	X
cc -O -Dstrrchr=rindex -s -o hier hier.c sftw.o	X	X	X
ls	X		
man getopt	X	X	X
cat Makefile.bsd	X	X	X
ls	X		
cd News/emacs	X	X	X
ls	X		
ls -la e*	X	X	X
nroff -o1-2 emacsdoc.nroff	X	X	X
emacs	X		
ps agu	X	X	X
mail	X		
logout	X	X	X

Figure 3: A comparison of the commands that are saved by the existing history, the CS history, and the PDP history.

the user wishes to re-execute the "nroff" command found in the middle of the history. In this case, the user once again types "nroff," but instead presses "<ctrl>-C c" for a list of commands whose name contains the string "nroff."

The user can directly scroll thru the list presented by the CS history and select a command to edit then execute. In the existing C shell history, the user is only presented with a chronologically ordered list of all the commands and must specify items by number or a simple string and not by a direct manipulation technique. If a pattern matching search is used, then only the most recent command is recalled. The CS history can use a pattern match search to display either a list of commands or return the single most recent command.

4.3. PDP history

The fourth column of Figure 3 shows the commands that are saved by the PDP history. This column is similar to the CS history column. However, duplicate commands, such as all three occurrences of the "logout" command, are saved.

Suppose the user again wishes to re-execute the "nroff" commands. In the PDP history, the user simply enters the string "nroff." The PDP system decomposes the string to its constituent bigrams—"nr," "ro," "of," and "ff"—and then propagates these bigrams through the network. Other bigrams are activated and the system returns the list of commands shown below:

```
ptroff -me -PT4387 -o1-2 introduction.txt
nroff -o1-2 emacsdoc.nroff
nroff -ms -o1-2 stacks.p
nroff -ms stacks.p
vi introduction.txt
nroff -me stacks.p
nroff stacks.p
psroff -ms -P219 -o1-2 stacks.p
psroff -me -P219 -o1-2 stacks.p
```

This list is ordered by the number of activated bigrams. (Note that this list includes some commands that are not shown in the history excerpt of Figure 3, however, they do occur in the entire history list of 600 commands.) Some commands, such as "vi introduction.txt," are listed but do not contain the string "nroff" and "nroff" actually activated many bigrams, among them the bigrams found in the "ptroff" command. The string "introduction.txt" occurs in both the "ptroff" command and the "vi" command, therefore the "vi" command was also returned. This ability to recall associated commands that are not directly requested is unique to the PDP nature, and is not found in the existing UNIX history nor the CS history.

The PDP history can also recall a command even if the user misspells the query for it. For example, suppose the user wishes to recall the "vi" command and enters the string "imtro" instead of "intro." The PDP history would recall the following commands:

```
vi introduction.txt
ptroff -me -PT4387 -o1-2 introduction.txt
mail -s 'Could you get the printout?' trj@trj
```

Although the "mail" command was also recalled, it contains fewer activated bigrams and is placed at the end of the list.

Both the CS history and the PDP history can recall a list of commands which contain a specific string. The PDP history, however, will also recall related commands and can account for user misspellings. The PDP history arranges the list of recalled commands according to the number of activated bigrams that the commands contain.

5.0 DISCUSSION

5.1. Advantages of a structured history

The existing UNIX history is very limited. It has only a few simple recalling methods and cannot be used to recall commands that were not recently executed. A structured history provides the following advantages over the existing UNIX history:

• *A list of possible commands is recalled rather than a single command.* The user can apply a variety of search strategies to recall a list of commands and can then directly select a command to edit or execute. In the CS history, the user has access to a wider range of search methods. The user can be more specific and ask for a command by both name and frequency, or the user can be more general and ask for all the commands containing a specific string. The user can say, "I want the commands which I've used more than a dozen times and which accesses the file 'temp.dat' ."

- *In the PDP history, associated commands are recalled.* The user can specify a fairly ambiguous string of characters and the PDP history will recall all commands that are associated with these characters.

- *More commands are saved.* Since duplicate commands and simple commands are not saved, a greater number of the more important, hard to recall, complex commands can be stored.

5.2. Problems and shortcomings

Both the CS history and the PDP history had several problems with GNU Emacs. GNU Emacs Lisp provides a powerful prototyping environment. However, it has the following drawbacks: The shell mode has several deficiencies involving the terminal emulation type and can't be used to completely replace the UNIX shell. There is no floating point number support. This was most apparent in converting the output of SunNet (floating point activation levels) to UNIX commands and sorting them by activation levels. It is not Common Lisp compatible and oriented towards editing and buffer manipulation. And it is slow.

The most significant problem with SunNet is that of any neural network simulator—a great deal of processing time is required to train the network. The example in Section 4.3 took many hours to learn using a Sun 3/60 workstation. A dedicated neural network, rather then a general purpose simulator, is required to make the PDP implementation efficient. Since the network used here contains only two layers and uses delta learning[4], a PDP history extension to a UNIX shell can be written.

Some shortcomings of the PDP history as implemented in the prototype were: There is no forgetting or unlearning. The user can delete the undesired commands, however, the PDP system would have to be reset and the entire history list of commands would have to be relearned. There is no incremental learning. This may be overcome if, instead of the backpropagation algorithm, we use a new algorithm with forgetting (Ishikawa, 1989b). Finally, the PDP history has not been applied to the shell. The creation of a special purpose PDP history would eliminate or improve these problems. The implementation of a structured history is practical if implemented as part of the C shell.

Sometimes completely unrelated commands are recalled. This is both an advantage and a disadvantage. It is an advantage because associated commands are recalled. It is a disadvantage because the associations might be very weak. In the example of the previous chapter, the command "mail -s 'Could you get the printout?' trj@trj" was recalled given the string "imtro." This string activated too many bigrams, some totally unrelated. This excess activation is probably due to the limitations of delta learning and networks with no hidden layers (see Minsky, 1969 for a further discussion on the limitations of these networks). The network could not properly learn the distinction between the "mail" command and the "vi" command. Therefore, it simply grouped the two together.

5.3. Future work and other possible designs

The saving method of a structured history eventually determines the maximum number of commands that can be recalled. As a first approximation, command complexity was defined as command length. Only commands longer than five characters were saved. Other measures such

4 The basic notion of the delta rule is to compare a desired output pattern with the actual output pattern and then modify the weights to reduce the difference.

as the number of words, the number of arguments, or a combination of the two could also be used. An accurate complexity measure would require a study of how users remember and forget commands.

Only the command name, the command options, the relative time, and the frequency of use were used as search methods in the CS history. In an informal survey of UNIX users, users had the most difficulty in recalling the arguments for a command, but had little difficulty in recalling the command name. Therefore, recalling commands by name and options was considered the most important features of a structured history. Since the existing UNIX history can list all the commands by the relative time that they were executed, the CS history kept this feature and used the relative time as a recalling method. Frequency of use was used as an example for other search methods.

Three other methods could also be considered: categories, sequencing groups, and relations. Commands could be grouped into functional categories—such as file commands, text formatting commands, and administrative commands—or commands could be grouped into project categories such as thesis commands, homework commands, and personal commands. Some UNIX programs (command names) have an option that would classify them into one category and another option that classify them into another category. Since each command line is saved independently of other command lines, commands with the same name could be recalled separately using the category method, or recalled together using the command name pattern matching method.

Some commands are often used in a sequence. For example, the commands "vi test.c," "cc -o test test.c," and "test arg1 arg2" would be used together to debug a C program. A structured history could save these commands as a group, allowing the user to easily recall the next command in the sequence even months after the commands were first executed.

Finally, commands can be grouped by relations, i.e., a command in a structured history can be retrieved by its relationship to other commands. For example, a command name may be the result of a compilation of a C program and the command options may include the name of a file that must be created with a special program. A structured history would recall the command name, the command for compilation, and the command for creating the required file.

A combination of a PDP network and conventional algorithms would be the best system for a structured history. The CS history has the advantage of speed and the ability to delete commands. The PDP history, on the other hand, allows for minor spelling errors and can recall associated commands. Perhaps a system which uses a PDP model for pattern matching and a conventional model for other search methods would give the system all the advantages of the PDP nature and yet would allow the user to specifically ask for particular commands.

A structured history is a tool that can be applied to many other systems besides UNIX. A structured history could be applied to other operating systems, control systems, or any application system that uses a complex command language and requires the user to re-enter difficult to remember commands.

Acknowledgement: Some of the research in the article was done while Meera Blattner was visiting the Electrotechnical Laboratory (ETL) under the sponsorship of the Invitation Program of the Agency of Industrial Science and Technology (MITI). She would like to thank, Motoi Suwa, Chief of the Cognitive Science Section, and others at ETL, that made this joint work possible.

6.0. REFERENCES

Franklin, D. (1987). "UNIX: rights and wrongs." *UNIX papers*. Indianapolis, Indiana: Howard W. Sams and Co. 3-40.

Greenberg, S., and Witten, I. H. (1988). "How users repeat their actions on computers: principles for design of history mechanisms." *Proceedings ACM SIGCHI 1988,* 171-178.

Ishikawa, M. (1987). "Toward Interfaces which Learn User Behaviors." Report, ICS, UC San Diego.

Ishikawa, M. (1989a). "A Structural Learning Algorithm with Forgetting of Link Weights." *Preceedings of International Joint Conference on Neural Networks,* June 18-22.

Ishikawa, M. (1989b) "User Interface Models by Connectionist Approach," *Journal of The Society of Artificial Intelligence*, Vol. 4, No. 4 (In Japanese), 398-410.

Kochan, S. G., and Wood, P. H. (1985). *UNIX shell programming*. Indianapolis: Hayden Books.

Kohonen, T., and Ventä, O. (1988). "A content-addresssing software method for the emulation of neural networks." *IEEE international conference on neural networks, 1,* 191-198.

Miyata, Y. (1987). "SunNet." Ver. 5.2. Computer program. UC San Diego.

Norman, D. A. (1981). "The trouble with UNIX." *Datamation*, 27(12), 139-150.

Rumelhart, D. E., and McClelland, J. L. (1986). *Parallel distributed processing: explorations in the microstructure of cognition*. Vol 1 and 2. Cambridge, MA: MIT press.

Sebes, J. (1987). "Comparing UNIX shells." *UNIX papers*. Indianapolis, Indiana: Howard W. Sams and Co. 123-151.

Stallman, R. (1986). *GNU emacs manual*. (6th ed.) Vol 18. Free Software Foundation.

Steele, G. L. (1984). *Common Lisp: the language*. Digital Equipment Corp.

Uejio, Jeremy Y. (1989) A Structured Command History for UNIX using a Parallel Distributed Processing Model. (Master's Thesis, University of California, Davis) Lawrence Livermore Tech. Report, UCRL-53919.

UNIX User's Manual—Reference Guide (1984). Berkeley, CA: Computer Science Division. Ver. 4.2 Dist.

Vitter, J. S. (1984). "US&R: a new framework for redoing." IEEE Software, 1, 39-52.

Yang, Y. (1988). "A new conceptual model for interactive user recovery and command reuse facilities." *Proceedings ACM SIGCHI* 1988, 165-170.