SCHOOL OF INFORMATION SCIENCES AND TECHNOLOGY
THE PENNSYLVANIA STATE UNIVERSITY

Dismal: A spreadsheet for sequential data analysis and HCI experimentation

Frank E. Ritter and Alexander B. Wood

Technical Report No. ACS 2002-1
24 January 2002

Ritter@ist.psu.edu
ritter.ist.psu.edu

Phone +1 (814) 865-4453 Fax +1 (814) 865-5604

School of IST, 007 Thomas Building, University Park, PA 16802

Dismal: A spreadsheet for sequential data analysis and HCI experimentation

Frank E. Ritter and Alexander B. Wood
ritter@ist.psu.edu abw136@psu.edu
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16801-3857

ACS #2002-1
24 January 2002

Abstract

Dismal is a spreadsheet that works within the GNU Emacs editor, a widely available programmable editor. Dismal has three particular features of interest to those interested in studying behavior: (a) the ability to manipulate and align sequential data, (b) an open architecture that allows users to expand it to meet their particular needs, and (c) an instrumented and accessible interface for studies of human-computer interaction (HCI). Example uses of these capabilities are provided including two cognitive models that have had their behavior aligned with protocols, extensions useful for teaching and doing HCI design, and studies using keystroke logs from the timing package in Dismal. Dismal is distributed with the help of the Free Software Foundation.

Acknowledgements

The initial development work was done jointly with David Fox when he was at NYU and was sponsored in part by a training grant from the Air Force Office of Scientific Research, Bolling AFB, DC; and in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB under Contract F33615-90-C-1465. Dismal's later development was partially supported by the Economic and Social Science Research Council's Centre for Research in Development, Instruction and Training, by a grant from the Joint Council Initiative in HCI and Cognitive Science, number SPG 9018736, by a grant from DERA, contract number 2024/004, and by Lockheed Martin. Numerous users have contributed bug reports, and Erik Altmann, Stephen Eglen, David Lamkins, Nigel Jenkins, Sarah Nichols, and Roberto Ong have contributed extensive additions. Angela Barnhill has provided useful comments on this report.

Table of Contents

1. Overview of Report

Dismal is a spreadsheet that was designed to gather and manipulate behavioral data. Figure 1 provides an example display. Dismal has three features of particular interest to those studying behavior: (a) the ability to manipulate and align sequential data, (b) an open architecture that allows users to expand it to meet their particular needs, and (c) an instrumented and accessible interface for studies of human-computer interaction (HCI).

Dismal extends the GNU Emacs editor (Free Software Foundation, 1988) using Emacs' extension language, Emacs Lisp. There is a large potential user community for Dismal. Users of GNU Emacs, a popular text editor installed on most UNIX and Linux systems and on many PCs, can use Dismal. This community is large indeed, for Emacs is installed on most UNIX machines by system administrators or vendors, and on most of the 8-10 million Linux systems by their owners. Dismal has been accepted for inclusion as part of GNU Emacs distribution, so it will included on many machines automatically in the future. Dismal also works with the latest version of Emacs for the Macintosh.

Example uses of these capabilities are provided illustrating its use in sequential data analysis, extensions useful for teaching and doing HCI design, and studies using keystroke logs from the timing package in Dismal. In addition to providing a tool, Dismal illustrates some new features that designers of other spreadsheets and related tools may wish to include in their systems.

2. General Capabilities

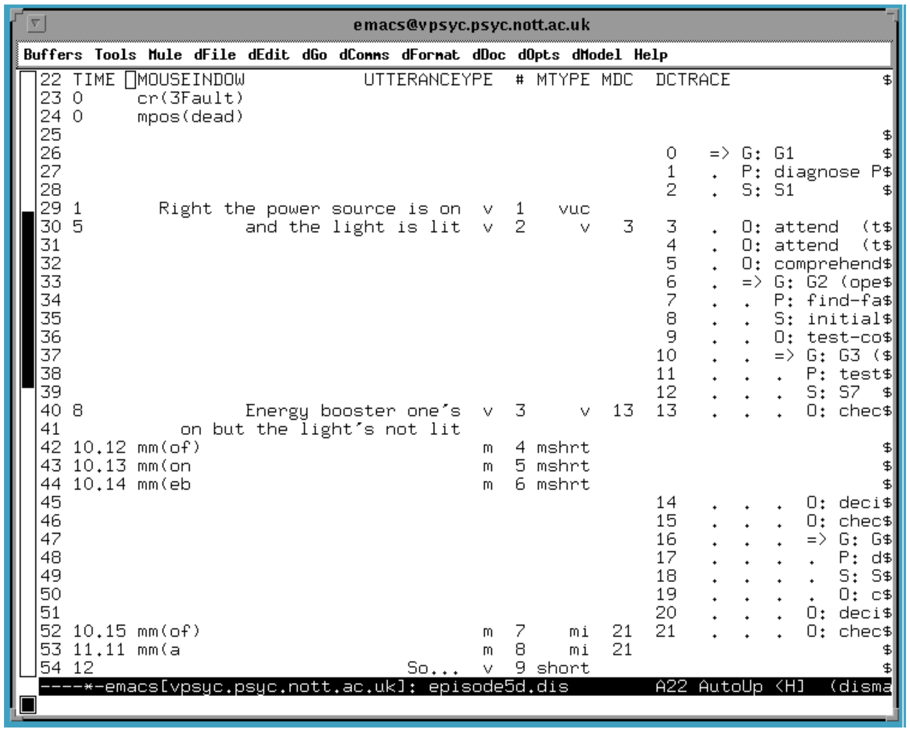
Emacs supports editing different types of files called 'major modes'. These modes, implemented as a set of Elisp commands, adapt the basic editing commands to suit what is being edited. For example, forward-paragraph might mean different things when editing text and when editing a C program. This functionality is implemented in and can be extended through a powerful language called Emacs Lisp, or Elisp. Elisp is available as an interpreted language, but also includes a powerful compiler.

Dismal runs wherever GNU Emacs (version 20 or later) is available. Currently this is all UNIX platforms and Linux. Dismal also works with OS2 and with X Emacs, an offshoot of GNU Emacs.

Dismal is implemented as a major mode[1]. It is a rather extensive major mode, and changes nearly all the commands to have Emacs work like a spreadsheet with a given file.

Dismal thus includes most of the major functions that one now expects from a spreadsheet, such as (a) the addition, deletion, clearing, and pasting of cells, rows, columns, and range; (b) formula entry and evaluation; (c) movement within the spreadsheet with keystrokes and mouse movements; and (d) the ability to format each cell's display. The more global layout of spreadsheets and their components are reconfigurable by adjusting the location, width, and alignment of columns. Users can interact with the spreadsheet through keystrokes, menus, and function calls. As with any spreadsheet, the ability to put formulas into cells and to create new functions allows many analyses to be performed directly. For example, several functions have been added to translate grades between letters and number equivalents.

[1] This gives rise to its name, DIS Mode Ain't Lotus, coined by David Fox, its inital developer.


emacs@vpsyc.psyc.nott.ac.uk
Buffers Tools Mule dFile dEdit dGo dComms dFormat dDoc dOpts dModel Help

```
22 TIME  MOUSEINDOW           UTTERANCEYPE  # MTYPE MDC  DCTRACE              $
23 0     cr(3Fault)
24 0     mpos(dead)
25                                                                            $
26                                                        0   => G: G1        $
27                                                        1   .  P: diagnose P$
28                                                        2   .  S: S1        $
29 1       Right the power source is on  v  1   vuc
30 5               and the light is lit  v  2     v   3   3   .  O: attend  (t$
31                                                        4   .  O: attend  (t$
32                                                        5   .  O: comprehend$
33                                                        6   .  => G: G2 (ope$
34                                                        7   .  .  P: find-fa$
35                                                        8   .  .  S: initial$
36                                                        9   .  .  O: test-co$
37                                                       10   .  .  => G: G3 ($
38                                                       11   .  .  .  P: test$
39                                                       12   .  .  .  S: S7  $
40 8               Energy booster one's  v  3     v  13  13   .  .  .  O: chec$
41           on but the light's not lit
42 10.12 mm(of)                          m  4 mshrt                           $
43 10.13 mm(on                           m  5 mshrt                           $
44 10.14 mm(eb                           m  6 mshrt                           $
45                                                       14   .  .  .  O: deci$
46                                                       15   .  .  .  O: chec$
47                                                       16   .  .  .  => G: G$
48                                                       17   .  .  .  .  P: d$
49                                                       18   .  .  .  .  S: S$
50                                                       19   .  .  .  .  O: c$
51                                                       20   .  .  .  O: deci$
52 10.15 mm(of)                          m  7    mi  21  21   .  .  .  O: chec$
53 11.11 mm(a                            m  8    mi  21                       $
54 12                              So... v  9 short                           $
----*-emacs[vpsyc.psyc.nott.ac.uk]: episode5d.dis        A22 AutoUp <H]  (disma
```

Figure 1. Dismal running within an X windows environment. A sample model/data alignment taken from Ritter and Bibby (2001, episode 5).

All of the commands are available on keystrokes, as well as on pull-down menus under X windows and a continuously available command line menu (Ritter & Ong, 1994). The commands are implemented as Elisp functions, so further extensions or customizations can be created, such as the Keystroke Level Model below, and a few users have done this.

Unlike all the other modes in Emacs, it does not primarily keep its information in a text buffer. Its primary data structure is a matrix kept as a variable associated with the buffer. Each cell in the matrix keeps information about its current value, its dependencies, and special information about how to print it (i.e., decimal width and alignment). Movement around the spreadsheet is thus implemented as movement in the textual expression of the matrix (the buffer) and in the matrix itself.

Additional variables are kept for each buffer noting dependencies between cells, how to draw columns, and so on. Saving a spreadsheet buffer is also different from saving a typical buffer. The buffer contents are written out (which saves a lot of time when loading), but the variables also must be written out. When a file is opened, Dismal specifies that the variables get reset and the textual portion gets displayed.

3. Facilities for Sequential Data Analysis

Sanderson and Fischer (1994) note that there are several types of exploratory sequential data analysis (ESDA). Dismal provides support for simple and complex ESDA analysis, ranging from the simple analyses of computing word counts, to searching for lines matching given patterns, to semi-automatically assigning codes to segments, to the more advanced analyses of automatically aligning protocols with traces taken from running cognitive models.

3.1 Simple measures and simple analyses

Dismal's tabular display supports simple, initial visual inspection of sequential data. The tabular display of the information sequences, their arrangement side-by-side, and the ability to print out the correspondences, is necessary for ESDA. The spreadsheet functions also support the preliminary step of resegmenting by adding additional cells and breaking a segment into several segments.

Most tools for manipulating protocols include the ability to assign codes to protocol segments as a step in model formation (Ritter & Larkin, 1994). Through a menu or keystroke command, a segment can be assigned one of a set of codes. Dismal includes this as well. These codes can be entered by the user, taken directly from a elated, but perhaps incomplete, cognitive model, or from a saved file of previously used codes. A protocol interpreted this way can be analyzed like any other, including the ability to count matched segments and to perform simple analyses. The underlying spreadsheet provides formulas for computing simple aggregate measures on the coding. A data-base facility (somewhat similar to the database facilities in Excel) is included in GNU-Emacs, including the ability to display all segments that do (or do not) match a given regular expression.

As a spreadsheet, Dismal provides a tabular display of these correspondences. The tabular display helps the analyst see how to line up the predictions and to understand the alignment by providing the context of each match, along with a visual operation (scanning a row) to identify the prediction and data that are paired. The tabular display also shows how much of the model is matched and unmatched, and it starts to show patterns in the alignment.

This tabular display, with the field names shown as column headings, also allows more data (context) to be displayed on the screen than a database approach can provide. The tabular display reflects the underlying matrix organization of the data into rows of segments that each include several fields displayed as columns. Automatic alignment programs and semi-automatic tools have a uniform and appropriate data structure holding the segments and protocols to manipulate.

3.2 Complex protocol coding: Model traces

An advanced form of sequence analysis is to use a process model to predict the actions in the sequence. The model's predicted sequence is tested by interpreting and aligning protocol data with respect to a model's sequential predictions. This can be an important component of cognitive model testing (Ritter & Larkin, 1994).

As part of this analysis Dismal introduces the idea of meta-columns. This is necessary when the model's predictions or the data span more than one simple spreadsheet column. For example, most analyses will use two columns, one to hold the model's predictions and one to hold the simulation cycle. These two simple columns would be placed in a meta-column. During alignment operations these two columns need to stay aligned with respect to each other. A meta-column ties these columns together so that cells within a meta-column remain aligned. In the examples included here the columns on the right are the data columns, and the columns on the left are associated with the model's trace, but this need not be the case.

3.3 Aligning segments by hand

Dismal provides two functions for aligning the meta-columns by hand. The most powerful function aligns two user selected lines. The analyst can also insert blank rows into each meta-column individually. This allows offsetting meta-columns while maintaining alignment within each meta-column.

3.4 Aligning unambiguous segments automatically

Aligning predictions to data by hand is tedious and error prone, so some assistance is warranted. Providing alignment automatically and completely is beyond current parsing technology when dealing with natural language. However, other data streams (e.g., mouse clicks) can be discrete and relatively unambiguous. These data can be automatically aligned with process models' predictions.

Dismal provides a simple alignment algorithm that can automatically align trace-elements with unambiguous protocol segments. The automatic alignment can be supplemented and corrected by the previously mentioned manual alignment commands.

The algorithm used by Dismal to compute the alignment is based on the Card algorithm (Card, Moran, & Newell, 1983, Appendix to Chapter 5). Card's algorithm solves the maximum common subsequence problem (Hirschberg, 1975; Wagner & Fisher, 1974), except the output will not just be the maximum length, but the actual matched subsequences that will be used to align the two meta columns in the spreadsheet.

Dismal's algorithm, Card2 (Ritter, 1992; Ritter & Larkin, 1994), displays the matches one-by-one to the user for verification. After viewing each proposed match, the user can accept it, decline it, or accept all the remaining matches. After the set of matches has been approved, the alignments are used to align the meta-columns in the spreadsheet. Card2 includes three further additions.

(a) Matches that start at the front of the protocol and model trace are preferred. There may be several possible "best" alignments. Because we are interested not just in how much could be aligned, but in using the alignment to understand how the alignment could be improved, which possible match set gets chosen is a real concern. Card2 satisfies the requirement of starting the match at the front by returning the subsequent that starts closest to the front of the two input sequences. Consider aligning the two simple strings DUC and DUDUDU. Card2 would return an edit list (referenced by position) that would call for aligning the first D's together. This results in aligning the initial predictions, which is a more stable alignment if changes to the strings tend to come at the rear: If additional D tokens were later added to the shorter list, the alignment will change less.

(b) The match process is based on regular expressions, not just constants. In the original maximum common subsequence problem, the comparison between the two information streams is that of strict equality, are the tokens the same tokens? Here, the two information streams will often use different names. The comparison step has been modified to do a more flexible comparison based on pairs of regular expressions, such as "mouse-move" matches "M(*)", where * represents "matches anything". The basic functionality of matching regular expressions is taken from this capability in GNU-Emacs.

The matching process also serves as the location to incorporate a natural language matching process. With regular expressions, the analyst can also start to compare in a simple way keywords expressions with verbal utterances. The model's predictions represent the actions and knowledge structures that one expects to find in the subject's actions and verbal utterances. Because there already exists a strong model of what will be said and done, the knowledge structures (the model's predictions) for a general parse are available -- this is a restricted case of natural language parsing.

The scope of the natural language parsing problem makes it clear that a simple keyword parser will not be adequate to interpret the data with respect to the predictions completely automatically even in this restricted form. Even if the parse is only approximate, however, it will make the alignment easier to perform.

<u>(c) The last duplicated item in a matched subsequence is preferred.</u> For example, when analyzing the Browser-Soar (Peck & John, 1992) data illustrated in Figure 2, we prefer to have the last possible item in a series of equal tokens to be used in the alignment. In that case, the earlier M's (mouse movements) represent mistaken applications of the M (mouse move) operator, while the final one tends to more closely matches the subject's behavior. For example, in matching XMDU to YMMMDU, Card2 would provide the sequence:

```
XM--DU
YMMMDU
```

One might prefer (and in analyzing Browser-Soar we do prefer) the following match:

```
X--MDU
YMMMDU
```

Doing this directly within Card2 would require a relatively large extension to the algorithm using either look-ahead search and access to the sequence values or else a new data structure. Manipulating the edit list returned by the Card2 algorithm is much simpler. A separate function is applied after the initial alignment that searches for multiple identical items in a row. If it finds such a set, it adjusts the lists to align the last element.

3.5 The types of segment alignments

Once the predictions have been aligned with the data, either automatically, semi-automatically, by hand, or through a combination of these approaches, they must be presented in an interpretable manner. Figure 2 shows another Dismal display with data and model predictions. The simple columns A through H are a meta-column of subject data, and the simple columns I and J are another meta-column used to represent the model's predictions. In this display, lines 0 through 10 are used as a header, but like any other spreadsheet, these rows are not fixed as header rows. Line 11 holds a ruler, which names the columns. This too can be adjusted to any row or omitted. When the user scrolls, the contents of the ruler row are redrawn as the top line of the display.

Formulas directly support many low level analyses of the comparison, including summary statistics of the comparison such as counting the matches and numbers of operators matched. All of the simple model/data measures presented in Ritter (1993), such as "goodness = hits - false-alarms", are directly supported. Additional measures, such as the number of model actions matched and number of words in a protocol have been added as formulas or special functions.

This example includes all of the ways Dismal can display how the model's predictions are matched by the data. The display should be capable of representing the types of correspondences noted in Table 1. The way Dismal represents each type of match is also noted in Table 1 and used in Figures 1 and 2 in the column labeled MTYPE.

```
    A     B      C          D      E  F   G   H  I                  J
  +--+--------+--------+-----------+--+--+----+--+---+--------------------------------
...
 17             1   SHORT is too short to code.
...
 20            10   V is verbal coded.
 21             6   MR is mouse required (these are movements).
 22             2   MI is mouse inferred.
 23            47   MBA is mouse button actions (necessary by default).
 24
 25             3   MUC is mouse uncoded.
 26             0   VUC is verbal uncoded.
...
 42 T MOUSE     WINDOW         VERBAL  ST # MTYPE MDC DC SOAR TRACE
 44 3  ...I have to declare variables   v 1    v  23
 46                                                  0  G: g1
 47                                                  1  P: top-space
 48                                                  2  S: s5
 49                                                  3  O: browse nil
 50                                                  4  =>G: g19 (operator no-change)
 51                                                  5  . P: browsing
...
 69                                                 23  ... O: generate-eval-criterion
...
 75                                                 29 ... O: find-criterion (keyword)
...
 80 5 M(+x) L of 4th keyword           m  2  mi  34 34  .... O: evaluate-current-window
 81 6                    Ahh! Uhm!...  v  3 short
 82 6 M(-x+y) below scroll bar elev.   m  4  muc
 83 7       Which I ... should go to   v  5  v   29
 84                                                 35  ....=>G: g258 (oper. no-change)
 85                                                 36  ..... P: evaluate-items-in-window
...
110                                                 61  ...... S: s486 (to-be-found vars)
111 7 M(+x-y) Left keyword D arrow     m  6  mr  62 62  ...... O: move-mouse (keyword down)
112    ---(+x) Right keyword D arrow
113    ---(-x) Right keyword D arrow
114 7 D  keyword menu scrolls D        b  7 mba  63 63  ...... O: press-button
115 8 U  stops                         b  8 mba  64 64  ...... O: release-button
116                                                 65  .... O: evaluate-current-window
...
143                                                 92  ...... S: s759 (to-be-found vars)
144 9 D  keyword menu scrolls D        b  9 mba  93 93  ...... O: press-button
145 9 U  stops                         b 10 mba  94 94  ...... O: release-button
146                                                 95  .... O: evaluate-current-window
...
```

Figure 2. Example display of a model trace aligned with data (taken from the Vars episode of Browser-Soar). Left-hand columns "T" (time of subject's actions) through "MDC" (matched decision cycle) are one meta-column (columns A-H), and columns "DC" and "Soar trace" on the right (I and J) are another meta-column.

There is one type of correspondence that cannot currently be represented easily in Dismal, a subject action that is matched by multiple model actions. The current representation assumes that what matches a given data segment is a single action of the model. This lack of representation serves as a useful constraint -- segments that match more than one model action probably are not segments. Either the model is more fine grained than the data, or the segment should be considered for resegmenting.

Table 1. Types of correspondences that are used in model/data alignments in Dismal.

An uncodable subject action, one that cannot be interpreted with respect to the model, is shown on line 81 (as numbered on the left-hand side) -- a verbal utterance that is too short to code.

Uncoded model actions are shown in lines 46-51, 69, and so on. They are indicated by model traces without corresponding subject actions.

Hits are shown in lines 44, 80, 82, and so on. The match is indicated in columns E through H. Column E notes the type of the match of the segment (ST), which in this case is mouse movement inferred, or mi. The type of correspondence (column G, MTYPE) is of a matched mouse movement. The simulation cycle that is matched by the mouse movement is shown in Column H as the matched decision cycle (MDC).

A miss is shown on line 82. The subject has clicked the mouse, and there is no corresponding action in the model's trace.

A pair of actions that are crossed in time is shown in Lines 75 and 80. The corresponding behaviors cannot be directly aligned while keeping them in order, so the matched decision cycle column is used as a reference for the subject action matched.

3.6 Example Analyses

The autoalignment facility in Dismal has been used to align protocols with model traces. Figure 1 shows its application to Diag-Soar (Ritter & Bibby, 2001). Figure 2 shows (in a slightly modified form) its first application to the Browser-Soar model (Ritter & Larkin, 1994). While these two examples have used Soar models, any two sequences can be aligned.

The model in Figure 2 had a large number of external actions, of mouse moves and clicks, which allowed the automatically alignment algorithm to do most of the final alignment. The model in Figure 1 had far fewer external actions. While the alignment algorithm was useful for the analysis in Figure 1, the analyst here performed most of the alignment by pairing up segments.

4. Facilities for Extensions

Dismal itself is extendible because its source code is provided. The extensions are relatively easy to create and test because extensions are written in an interpreted language for which there is a primer (Chassell, 1997) and a manual (Free Software Foundation, 1998), both of which are available online. Users can write their own custom commands, for example, to count the occurrences of sequences in a column. Users could also change the way cells are displayed, the way numbers are represented, or how cells are updated.

Emacs itself includes an large set of existing extensions (about 300 packages totaling 13 meg) to serve as examples. These extensions include terminal emulators, shell (process) control programs, debuggers, and even games. Emacs' software architecture makes it a straightforward task to tie these packages together. This permits systems using Dismal to be written in the native Lisp code or called as an associated UNIX process.

As the source code is provided, Dismal can be used itself as a test bed for evaluating various interface designs, and comparisons can be based on actual user data. This can even be done as part of student projects. Two students in Lisp programming classes at the University of Nottingham (UK) have provided useful extensions to Dismal as class projects, and students at the University of New Brunswick (Canada) have done similar work. Dismal users have created their own packages to compute flight expenses, control machine tools, and to draw graphs from the spreadsheet data.

The largest extension has been to create a package for exploring HCI design, by computing aliases for a command set, and predicting the expected time savings. A useful approach towards improving interface design is to incorporate known HCI theory in design tools. As a step toward this, we have created a tool incorporating several known psychological results (e.g., alias generation rules and the keystroke level model). The tool, simple additions to Dismal, helps create theoretically motivated aliases for command line interfaces, and could be further extended to other interface types. It has been reported in detail elsewhere (Nichols & Ritter, 1995). We review it here to illustrate how Dismal is extendible.

4.1 An example extension to create an HCI design tool

The ideas behind the alias creation tool could in fact be applied to any command set, but the command set initially optimized with this tool was Soar (Newell, 1990), a unified theory of cognition realized as a cognitive modeling language (Laird, Newell, & Rosenbloom, 1987). It has previously been command line driven, and over 50 commands are available. While there is a graphical interface, the command line interface remains an important part of Soar, as some members of the Soar community prefer a command line. Also, some Soar users cannot use the graphic interface due to hardware limitations, or prefer not to because of program speed and size concerns.

There are a couple of reasons to create aliases for the Soar language (Congdon & Laird, 1995). It was (and is) used within our local environment and therefore it is worthwhile enhancing the system for both local users and the Soar community world-wide. There is also the potential to get command use frequency information and usability feedback from the local users. Some of the commands that exist within Soar are quite long -- this suggests that users could benefit greatly from the introduction of aliases.

The HCI theories were implemented by using two Lisp functions that were added to the Dismal spreadsheet. The first -- key-val -- took a command and calculated the number of mental operators and keystrokes used in the execution of the command. It then used the estimated typing speed to calculate a time prediction for the execution of the command according to the Keystroke-Level Model. The second function -- make-alias -- used the specific rules noted below to automatically generate commands aliases. However, human input was still needed to decide the best way to arbitrate clashes between aliases given the large command set examined here.

4.2 The alias generation function

An aliases generating function was created incorporating available alias generation guidelines (Ehrenreich & Porcu, 1982; John & Newell, 1987; Payne & Green, 1986; Watts, 1984), primarily using truncation and minimum-to-distinguish. These have been shown to the most efficient form of abbreviation (John & Newell, 1987), partly due to the ability of have consistency in the resulting command set (Ehrenreich & Porcu, 1982).

The characteristics of the command language itself were also considered -- particularly with reference to the fact that many of the commands consisted of several words. Therefore, we added the following additional guidelines to create a consistent alias set: (a) Include in the alias the first letter of each word in the case of multi-word commands. (b) If the length of the command is 5 letters or less, a one letter alias may be provided if clashes do not occur as a result of the new abbreviation. In general this means that already short, abbreviated commands (e.g., pwd), do not get shortened to one letter, but short one word commands (e.g., watch) do get abbreviated to a single letter.

This function can be placed in a cell, taking as an argument a pointer to another cell. It returns an alias. Clashes are kept on a global list that can be inspected. The automatic function generated 80% of the final aliases for the Soar command set, where the rules could be strictly adhered to.

4.3 The time predictor functions

The other functions created in Emacs Lisp and used in the spreadsheet are used to predict the time to execute the aliases and the original command set according to the Keystroke level model (Card et al., 1983). These functions could be further extended to apply to other types of interfaces.

These aliases reduce typing time by approximately 50%. Command frequency data, necessary for computing time savings and useful for arbitrating alias clashes, can be difficult to obtain. We found that expert users can quickly provide useful and reasonably consistent estimates, and that the time savings predictions were robust across their predictions and when compared with a uniform command frequency distribution.

The Keystroke-Level Model (Card et al., 1983) was used as a measure of design efficiency as it predicts the time taken for a set of commands or aliases to be executed based on sub-task speeds and frequency of tasks. It is a useful and practical simplification of GOMS (Goals, Operators, Methods and Selection) analysis at the level of individual keystrokes, when the user's interaction sequence can be specified in detail, as it could be in this case. The Keystroke-Level Model has been shown to predict user performance with between 5 and 20% error (Card et al., 1983; Haunold & Kuhn, 1994).

Our current model is adaptable for typing speed, but does not include the effect of different typing speeds for different letters. The inclusion of mental operators is governed by heuristics specified by Card et al. (1983), specifically rule 2 in their Figure 8.2. An important question to be considered is the increase in burden on memory and mental operators that the introduction of aliases might impose. The Keystroke-Level Model does make several suggestions about how the aliases should be generated consonant with the generation guidelines below. If a rule set is used to generate the aliases, once the initial (very small) rule set has been learned only a single mental operator is required to enter a command alias and that behavior can become automated more quickly -- rather than having to learn by rote each alias, a rule can be applied (Card et al., 1983).

4.4 Estimations of task frequency

The Keystroke-Level Model uses task frequency to balance the time it takes to do each task in a set of unit-tasks. An initial analysis of approximately two hours of actual subject data was performed to compute these frequencies. In this session, commands were used while a specific problem was solved. Perusal of additional transcripts suggested that command usage was highly dependent on the task, there were large individual differences, and in order to generate meaningful frequency data enormous amounts of keystroke logs would be required (we estimate on the order of hundreds of hours based on pilot data from Altmann).

Command frequency data, necessary for computing time savings and useful for arbitrating alias clashes, can be difficult to obtain. We found that expert users can quickly provide useful and reasonably consistent estimates, and that the time savings predictions were robust across their predictions and when compared with a uniform command frequency distribution.

In an attempt to generate useful frequencies more quickly, four expert Soar users, all with more than three years experience working with Soar, were asked to provide frequency estimates for their own use of the original command set. These were easy to provide, although they do not correlate very well with each other (mean pair-wise correlation of 0.49). In future work we expect to validate this approach as an approximation to complete logs.

4.5 Testing the predictions using Dismal

Figure 3 shows the estimated time savings based on the time to perform the original command set and the alias set balanced for each individual's frequency distribution. The time taken to execute the original command set is represented by 100%. This total time, based on the normalized command frequency distributions, varied between 245 s (flat frequency) to 458 s (User 3). When the keystroke values of the commands and aliases were calculated and weighted using a uniform command distribution (i.e., assuming all commands were used equally often), it was found that the aliases provided a 55% saving in time over the original command set.

Although each user had different preferences regarding which commands they used (or thought they used) to perform tasks, when savings were calculated with these frequency estimates, the time savings for individual users for the complete command set ranged between 38% and 53%. When the unchanged command names were omitted from the analysis, the average time savings across all the distributions increased to 62%.

This approach could be used within any programming language, but we prefer a spreadsheet to display the aliases. This visual presentation and the use of functions to compute and display the expected times makes the process easy to follow and provides updates automatically to the designer.

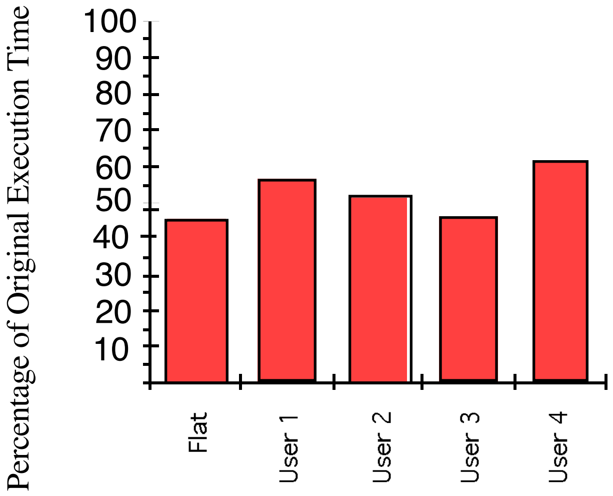


Figure 3. Comparison of predicted benefits of generated aliases for different command frequency distributions

4.6 The effect of aliases on time and errors

No users type perfectly, and decreasing command length, if it is consistent, may also decrease errors. The simplest prediction is that simple typing errors are related to the number of keystrokes, no matter what the users' typing accuracy or speed. This alias set reduced the number of keystrokes by 68%. In an unpublished study of 20 subjects we found that even while learning the interface through use, novices had a 62% reduction in typing time and a 48% reduction in errors (Ritter & Bishop, 1997).

4.7 Summary

Overall, this tool appears to be a robust and inexpensive way of applying simple HCI theories to design to reduce command execution time. Aliases can be constructed and tested very easily, and, with the use of an HCI tool, the principles behind the forms of command aliases can be applied routinely and uniformly. Savings estimates can also be documented directly and used to guide design when it is necessary or desirable. The use of a system using guidelines to generate aliases means that the alias forms are easy to learn and can generally be predicted without the particular alias being specified. The local Soar user group has found the alias set to be generally useful, and the aliases have also been distributed to the Soar community at large. Aliases improve the interface at quite modest cost, there appears to be no reason not to take this efficiency gain.

A further analysis using this package suggests that automatic command completion, while more useful than no support, would be inferior to this alias set. Command completion would not reduce the keystrokes in this set of commands as much as the aliases do, and would not decrease the mental operators at all.

5. Facilities for HCI Experimentation

Dismal is instrumented — it is possible to automatically record each user action and the time it occurred. We have found that undergraduates are able to use dismal in five week practical modules to gather realistic user data and test HCI theories (e.g., the keystroke model of Card, Moran & Newell, 1983).

The format is similar to other logging programs (e.g., Westerman, Hambly, Alder, Wyatt-Millington, Shrayane, Crawshaw, et al., 1996). Table 2 provides some example data. An automatically generated header comes first. The tab-delimited fields note for each keystroke the time in milliseconds, followed by the keystroke, and then the key binding. Analysis tools are included to find and aggregate the commands between carriage returns. The open architecture of Emacs would allow these logs to be played back, but we have not created such a facility.

Time data have been used to compare other interfaces that can be run within an Emacs shell. This facility has been used to gather data on using Emacs to program (Altmann & John, 1999).

Dismal is also useful for exploring HCI design, particularly as student projects. As the source code is provided, Dismal can be used itself as a test bed for evaluating various interface designs, and comparisons can be based on actual user data. Two students in Lisp programming classes at Nottingham have provided useful extensions to Dismal's interface as class projects.

Table 2. The keystroke logs generated by the timing package.

```
User: s12
Creation-date: Mon Feb 10 13:20:52 1997
System: upsyc.psyc.nott.ac.uk
Start-buffer: *scratch*

80855.017        ^M     minibuffer-complete-and-exit
80878.006        S      self-insert-command
80879.615        o      self-insert-command
80891.763        a      self-insert-command
80892.497        t      self-insert-command
80892.646        ^?     delete-backward-chars
80906.405        r      self-insert-command
80962.430        ^M     comint-send-input
80965.701        i      self-insert-command
80965.800        n      self-insert-command
...
```

6. Availability

Dismal and its source code is available at no cost. It is copylefted[2], which basically means that the source code is freely available, and that if you pass on Dismal you must include its source code. Dismal is available through a variety of sources. The most convenient is via FTP from one of the servers storing it. Its most recent version is stored at Penn State University (acs.ist.psu.edu/dismal) as well as through www.gnu.org/software/dismal/.

Older versions can be found at NYU and MIT. Other versions have been stored at The Ohio State Elisp archive, but these are not usually the latest versions It is also available by posting one 3.5-in high density diskette to Dr. Frank Ritter, School of Information Sciences and Technology, Penn State University, University Park, PA 16802. If you would like to be put on the dismal users' email list, send an email to ritter@ist.psu.edu .

A README file accompanies the tar, gzipped file providing installation instructions. The installer does not have to know GNU Emacs or Elisp, but they should be somewhat familiar with UNIX and the local file system. Users already familiar with Emacs will be the most productive.

7. Comparison with Similar Tools

Compared with Excel or other commercial tools, Dismal includes some new commands not supported in other spreadsheets, such as better movement commands (e.g., to the next filled cell, last filled cell in column). But a major difference between Dismal and similar tools is that Dismal exists within the GNU-Emacs environment. The GNU-Emacs environment provides the ability to cut and paste between files, an integrated text editor, and complete on-line help. Being part of GNU Emacs also provides access to simple commands to create temporary keyboard macros, and

[2] For more information, see www.gnu.org/copyleft/copyleft.html.

includes a full programming language for writing user functions, GNU Emacs-Lisp, which Dismal itself is written in. This Lisp can be run interpreted or compiled. More importantly, however, Dismal's source code is included in the distribution, so Dismal can be modified or integrated. Others have found developing macros and full extensions to Dismal straightforward, and it has been integrated with software for developing and testing cognitive models (Ritter & Larkin, 1994).

Some of Dismal's initial design and iconography for sequence alignment comes from Trace&Transcribe (John, 1994) and MacShapa (Sanderson, Scott, Johnston, Mainzer, Watanabe, & James, 1994). Dismal goes significantly further than Trace&Transcribe in its representations and in the tools it provides for interpreting and aligning the transcribed protocol data with respect to the predictions. It does not include as many exploratory sequential data analysis tools as MacShapa. Dismal's alignment algorithm has been added to the MacShapa analysis program.

Dismal includes a logging package that can time users, including users of other Emacs-based tools. Instrumentation of similar software is relatively unique. In most other ways, however, Dismal is a relatively modest spreadsheet. It is slower, less robust, and lacks some of the built-in features of other spreadsheets.

8. Summary

Dismal provides a useful environment for gathering, analyzing, and modeling data. User behavior in the Dismal spreadsheet and other behaviors in Emacs can be gathered as log files. The data can be analyzed with the spreadsheet functions provided, and students at Nottingham have done this.

Dismal also provides an algorithm for aligning the sequential predictions of cognitive models, either with protocol data or with another model's predictions. This alignment algorithm could have other uses as well, such as behavior concordances. When a complete description of correspondences can be provided such as keystrokes by the subject and the model, the two sequences can be automatically aligned. When the comparison is less clear, such as between a subject's verbal utterances and representations in the model, semi-automatic commands allow an analyst to align items by pointing. Together, these commands substantially reduce the work of testing cognitive models with protocols by up to a factor of five (Ritter & Larkin, 1994), allowing such analyses to be performed more often and with more insight.

As a very programmable spreadsheet, Dismal supports creating simulations and analysis programs. A set of functions for creating aliases and predicting their effect with the keystroke level model (Card et al., 1983) is included as an example. The example predictions could be tested by running subjects with the timing package available in Dismal.

In the future, Dismal will need to have its core features continued to be improved. These missing features include rather general features, such as the ability to split a buffer in two, better math functions, faster operation, and the ability to cut and paste ranges of cells with the mouse (in addition to being able to perform this with keystroke and menu commands). Dismal should also include more functionality specific to testing process models, such as a better way to choose sets of items to align and faster alignment of the cells through faster row and cell insertion and deletion.

Dismal has continued to develop over the last seven years because it serves several real needs in research, teaching, and administration. Further users and uses will strengthen it.

9. References

Altmann, E. M., & John, B. E. (1999). Episodic indexing: A model of memory for attention events. Cognitive Science, 23(2), 117-156.

Card, S., Moran, T., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Chassell, R. J. (1997). Programming in Emacs Lisp: An introduction. Cambridge, MA: Free Software Foundation. www.gnu.org/manual/emacs-lisp-intro/html_chapter/emacs-lisp-intro_1.html [checked 13 Dec 2001].

Congdon, C. B., & Laird, J. E. (1995). The Soar user's manual, version 7. Ann Arbor, MI: Electrical Engineering and Computer Science Department, U. of Michigan.

Ehrenreich, S. L., & Porcu, T. (1982). Abbreviations for automated systems: Teaching operators the rules. In A. Badre & B. Shneiderman (Eds.), Directions in *human/computer interaction*. Norwood, NJ: Ablex.

Free Software Foundation (1988). GNU Emacs. Cambridge, MA: Free Software Foundation.

Free Software Foundation (1998). *GNU Emacs Lisp Reference Manual*. Cambridge, MA: Free Software Foundation. www.gnu.org/manual/ [checked 13 Dec 2001].

Haunold, P., & Kuhn, W. (1994). A keystroke level analysis of a graphics application: Manual map digitizing. In *Proceedings of the CHI'94 Conference on Human Factors in Computer Systems*. 337-343. New York: ACM.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *CACM, 18*(6), 341-343.

John, B. E. (1994). A database for analyzing "Think-aloud" protocols and their associated cognitive models No. CMU-HCII-94-101). Carnegie-Mellon University Human-Computer Interaction Institute.

John, B. E., & Newell, A. (1987). Predicting the time to recall computer command abbreviations. In *Proceedings of the CHI'87 Conference on Human Factors in Computer Systems*. 91-97. New York: ACM.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*(1), 1-64.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Nichols, S., & Ritter, F. E. (1995). A theoretically motivated tool for automatically generating command aliases. In *Proceedings of the CHI'95 Conference on Human Factors in Computer Systems*. 393-400. New York, NY: ACM.

Payne, S. J., & Green, T. R. G. (1986). Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction, 2*, 93-133.

Peck, V. A., & John, B. E. (1992). Browser-Soar: A computational model of a highly interactive task. In *Proceedings of the CHI '92 Conference on Human Factors in Computing Systems*. 165-172. New York, NY: ACM.

Ritter, F. E. (1993). TBPA: A methodology and software environment for testing process models' sequential predictions with protocols (Technical Report No. CMU-CS-93-101). School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Ritter, F. E., & Bibby, P. (2001). Modeling how and when learning happens in a simple fault-finding task. In *Proceedings of ICCM - 2001 - Fourth International Conference on Cognitive Modeling*. 187-192. Mahwah, NJ: Lawrence Erlbaum.

Ritter, F. E., & Bishop, M. (1997). The effect of abbreviation on time and errors: Even novices can profit from aliases. unpublished manuscript:

Ritter, F. E., & Larkin, J. H. (1994). Using process models to summarize sequences of human actions. *Human-Computer Interaction, 9*(3), 345-383.

Ritter, F. E., & Ong, R. (1994). The simple-menu package, release 1.2. Available from The Ohio State University elisp archives on archive.cis.ohio-state.edu as file /pub/gnu/emacs/elisp-archive/interfaces/simple-menu.el.Z.

Sanderson, P. M., & Fisher, C. A. (1994). Exploratory sequential data analysis: Foundations. *Human-Computer Interaction, 9*(3&4), 251-317.

Sanderson, P. M., Scott, J., Johnston, T., Mainzer, J., Watanabe, L., & James, J. (1994). MacSHAPA and the enterprise of exploratory sequential data analysis (ESDA). *International Journal of Human-Computer Studies, 41*, 633-681.

Wagner, R. A., & Fisher, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery, 21*, 168-172.

Watts, R. A. (1984). *Introducing interactive computing*. Manchester: NCC Publications.

Westerman, S. J., Hambly, S., Alder, C., Wyatt-Millington, C. W., Shrayane, N. M., Crawshaw, C. M., & Hockey, G. R. J. (1996). Investigating the human-computer interface using the Datalogger. *Behavior Research Methods, Instruments, & Computers, 28*(4), 603-606.