REDUCING KEYSTROKE COUNTS WITH A PREDICTIVE COMPUTER INTERFACE

Ian H. Witten, John G. Cleary, John J. Darragh, and David R. Hill

Man-Machine Systems Laboratory, University of Calgary, Canada

Abstract

It has recently been demonstrated how techniques of non-deterministic structural modelling can be applied to reduce the number of keystrokes typed by a user in typical interactive computer sessions. The sequence of keystrokes is observed by the machine, modelled, and where possible used to form predictions of the characters about to be typed.

This paper discusses the application of such an interface to the physically handicapped. Particular attention is paid to reducing the number of keystrokes needed to enter different kinds of text, such as source programs, interactive commands, and plain English. In some cases up to 50% of a users keystrokes are correctly predicted, resulting in significant keystroke count reductions.

Introduction

Typical dialogues with interactive computer systems contain a great deal of redundancy. This redundancy arises from a number of sources. For example, minor errors by the user cause repetition of earlier sequences when the correction is made. If the dialogue is with a command interpreter there will be redundancy due to frequent use of a small subset of the possible commands and file names. If English text is being entered there will be the statistical redundancy of the language itself. Program text is usually even more redundant because of the restricted set of keywords and identifier names.

An earlier paper[1] described the operation and sketched the design and construction of a terminal interface to an operating system (Unix[2]) which is intended to aid the interactive user by reducing the amount he has to type. The system, which is called predict, works by predicting the entries that the user is about to make. To do this it treats the user dialogue as a sequential behaviour sequence which is presented as input to the system. Predictions are displayed in reverse video on the VDU terminal, and the user has the option of accepting correct predictions as though he had typed them himself. Incorrect predictions can be eradicated by simply typing over them; thus the user may ignore the predictions and continue typing normally if he does not wish to disturb his keying rhythm. In all cases the display looks as though he had typed the whole entry himself.

The present paper considers the application of such a software interface to the physically handicapped. After an introductory section which reviews typing aids for handicapped people, we describe the existing terminal interface.

There are some ways predict could be improved for use by the handicapped. For example, the current system selects a unique prediction (or none at all) which represents its "best guess" of the following characters. In the following section we review the possibility of using menu-based display techniques to allow several predictions to be shown simultaneously, the user selecting between them. This effectively trades the burden of key selection against cognitive load. Because of the higher cost of keystrokes which is associated with many forms of physical handicap, it is appropriate to pay more attention to the prediction algorithm so that the number of keystrokes needed to enter text can be reduced as much as possible. In the next section, we describe a new prediction method which has recently been proposed for use in text compression[3]. Although this has not yet been incorporated into the terminal interface, we have evaluated its performance off-line and show here what improvement can be expected. Finally we report on an evaluation experiment of the existing system, using able-bodied subjects.

The technique introduces a "meta-dialogue", above the normal dialogue level, in which the user accepts predictions or parts of them. The possibility of multi-modal input using simple isolated word recognizers for the meta-dialogue is explored, to achieve further keystroke reduction. The system performance is quite sensitive to the type of text being entered. In some cases, a large context is necessary to ensure that most predictions are correct: in others it is not. In some cases the predictor should be prepared to display a character when the model predicts it only weakly, while in others considerable confirmation of the prediction should be awaited first. These parameters of the prediction process are also suitable for voice control.

Typing aids for the handicapped

There has been a great deal of previous research on typing aids for the handicapped. This falls into two categories: work on input devices, including scanning techniques, much of which predates the use of computers; and acceleration and

CH1807-7/82/0000/0003$00.75

information amplification methods which increase the rate of user selections and the productivity of each selection.

The three input methods of scanning, encoding, and direct selection are ably reviewed by Raitzer et al.[4], who also cover physical implementations of the techniques. Basically, in scanning, the system steps through a series of options and the user responds when the appropriate item is reached. With encoding, the user keys a memorized code on a limited keypad to recall predefined text. Direct selection implies the use of pointing to pick a complete item from a screen menu or keyboard. The number of keys required thus increases as we proceed through the methods, and the dependence on software decreases correspondingly. Staisey et al.[5] give a further review of these techniques, specifically in the context of videotext. Any of the methods can be used with the prediction technique described in the present paper.

Acceleration can speed scanner input by increasing the dimensions of the scanning matrix[6], putting words, whole phrases, and even ideas in iconic form into the matrix[7,8], and sorting the scan into decreasing frequency order[9].

Information amplification goes further by increasing the productivity of selections, and the predict system is essentially a sophisticated form of such amplification. Previous authors have concentrated primarily on abbreviation expansion. Abbreviations are encoded beforehand and explicitly recalled by the user. Thus he must be familiar with the implemented abbreviation set. A simple example is the automatic insertion of two spaces following all periods, and automatic capitalization of the next word[10]. User-definable abbreviations have been implemented by Kelso and Vanderheiden[11], who allowed one to distinguish between several abbreviations beginning with the same stem using a unique terminating digit. The abbreviation expansion is displayed for checking before being entered into the text.

An extremely interesting scheme for information amplification uses pre-stored English tetragrams, based on representative text, which allow a character to be predicted[12] according to the three which precede it. In one implementation[13], the system displays the most likely next character and the user may accept it with a single switch closure. If he does not, the next most likely character is displayed. If this also is unacceptable, the machine resumes scanning. Goodenough-Trepagnier and Rosen[14] have further developed the technique. This scheme has many resemblances to predict. The key difference is that predict takes its n-grams from the text itself rather than from a representative sample of English, making it suitable for input of all kinds of text. Several authors[15,16] have commented on the necessity for this. Furthermore, predict looks ahead, in that more than one -- often many more -- predicted characters are displayed, and can be accepted, at once. Also, we have investigated the use of partial-string matching rather than a fixed trigram context for prediction.

An interface in operation

This section will try to give a feel for the use of the terminal interface, although it is always difficult to describe the operation of a highly interactive system on paper. The user invokes it when he logs on by typing predict. (It can be arranged to commence automatically on login.) Instructions are displayed which remind him how to use the system. From then on, all characters he types are intercepted by predict before being passed to the system -- whether they be destined for the command interpreter or for any other subsystem.

Predict makes predictions about what inputs will come next, based upon previous inputs. Although it can be run off-line to analyse pre-recorded terminal sessions, it is normally used interactively. In this mode it displays predictions in reverse video on the VDU screen. Because of obvious display limitations, alternative predictions are not shown: in effect predict guesses the most likely next entry and ignores other possibilities. For experimental purposes, we have made the predictor rather incautious in its operation. (This can be changed by altering parameters in the program.)

The predictor accepts characters one by one. If the first prediction, when accepted, would lead to others, a chain of predictions is made and displayed. This is normally the case, and so one usually sees several characters displayed in reverse video in front of the cursor position. However, no attempt is made to predict past a newline character. Up to a full line of predictions can be shown, but they are generally shorter than this -- either because predict's model indicates that a newline character will follow or because it reaches a point where there is no prediction.

When a prediction is displayed, the cursor is moved back to the position preceding it. The user can accept it in its entirety, or character by character, by pressing function keys. He can type over it if it is incorrect, and the prediction will disappear immediately (possibly being replaced by a new one). He can type over it even if it is correct, perhaps to avoid interrupting his keying rhythm.

Hence the system can be used in a completely transparent manner, although when typing quickly one finds the screen flashing with predictions in a diverting and distracting manner. The predictions do of course occupy bandwidth on the line connecting terminal to computer. Even at a high burst typing rate of 12 char/s, and in the worst case with 80 characters of prediction for each one typed, a 9600 bit/s connection can just about accomodate the traffic. In practice the scheme works well on a 1200 bit/s line.

TABLE 1 - Sample input sequence to the Unix operating system

```
cd ..
ls
cd modelling
ls -l
rm *.o y.tab.c
ls -l test2
cat test2
pr test2|opr
echo test*
rm test*
ls -l
rm tty tty.out
man compact
newgrp bin
cd ..
man compact
du
cd ../bin
cd bin
ls
cat bib.make
cd ../modelling
cp ../bib.make test3
cp ../bin/bib.make test3
compact test3
ls -l
od -c test3.c
od -c test3.C
pr test3.C ../bin/bib.make|opr
pr ../bin/bib.make|opr
od -c test3.C|pr|opr
rm test3.C
ls -l ../bin
cat ../bin/bib.indiv
cp ../bin/bib.indiv test4
pr test4|opr
compact test4
od -c test4.C
od -c test4.C|pr|opr
cd ../predictor
```

Instructions are displayed when predict starts-up that describe the function keys used to accept predictions. These function keys, identified by stick-on lables, include: char (accept one character), word (accept one word), line (accept the entire prediction), and help (print the start-up menu again). The most commonly used functions are char to accept one character of a prediction, and word to accept one word of a prediction. The program currently copes with four terminal types, the Ann Arbor Ambassador, Lanparscope400, General Terminal Corp's GT-400, and Digital Equipment's VT100. There are some restrictions in the use of the predictor which will be discussed later.

Table 1 shows a sample input sequence recorded by predict. It was produced in a live terminal session, and is all directed to the Unix command interpreter. It is not necessary to understand the meanings of the commands, but it is important to note the high degree of redundancy present. For example, "man compact" failed the first time and had to be retried after the environment had been altered. "od -c test3.c" failed because the file name should have been "test3.C". Considerable redundancy is also contributed by the frequent use of a small selection of commands and filenames. We will consider the performance of predict on this input sequence shortly.

It is worth emphasizing that although the example shows only an interaction with the operating system command interpreter, predict models the complete dialogue with the computer system, including subsystems such as editors and language interpreters. For example, when editing text one frequently repeats file names that have been mentioned in preceding dialogue with the command interpreter; either by reading the files, or temporarily escaping from the editor to perform some operation on them, or by editing a command file which is intended for interpretation at command level. In effect, a thread of context may run through the entire dialogue. If it does, predict will discern it.

Alternative modes of operation

There are several possibilities for reducing the number of keystrokes needed to enter a piece of text even further than does the current predict system.

One is to increase the options which face the user when a prediction is displayed. In the version just described he can ignore the prediction, accept a single character of it, or accept the complete line. We have recently included the ability to accept the next word of the prediction, having found in practice that it is often correct in cases where the entire prediction is not completely correct. A "word" in this context is defined as a space-separated string. But often in computer dialogues one thinks in terms of strings of alphanumeric characters separated by special characters like "/", ";", or "+". Hence it may be profitable to allow the user to accept the next alphanumeric string of the prediction. Such additions can substantially reduce the number of keystrokes that the user has to type.

A further possibility is for predict to display a menu of predictions at each stage instead of just one. This was not done in the present system because one aim was to keep use of the computer system with predict as close as possible to that without it. To display a menu, one would have to accept that the predictions will occupy a substantial part of the VDU screen instead of just the remainder of the current line as at present. Nevertheless, it would provide the user with much more powerful selection facilities. It is likely that if a handful of predictions were displayed, one of them would be applicable in most situations. Of course, the price paid is that more keys must be reserved for menu-selection purposes, and the user must spend more time

learning how to use the system.

It is natural to consider the use of an isolated word recognizer for controlling the generation and acceptance of predictions. Limited machines such as Threshold Technology's "Auricle", available on the market for some $3000, have the capability of identifying some 40 words, spoken in isolation, if trained appropriately by the user. These would be ample for controlling predict. For example, suppose that up to ten predictions were displayed, and that the user could select one and accept a character, word, string, whole line, or whole line completed by a newline character. Commands could be, for example,

"three character"

to accept the first character of prediction 3, or

"eight complete"

to accept the whole line of prediction 8, complete with newline character. The entire vocabulary needed is only 15 words.

One could argue that the user may prefer to dictate his whole message, character by character, instead of using a voice-controlled prediction technique. The predictions, however, will considerably reduce the number of voice entries. Furthermore, we feel that they achieve a useful separation between voice and keyboard entry which is likely to be faster than either would be alone, for many people. We have recently taken delivery of an isolated word recognizer and plan to test this hypothesis.

Some prediction techniques

The prediction technique used in the current predict program is a simple lexical one, based on the previous occurrences of short sequences of consecutive characters[17]. All novel sequences of k characters (where k is a parameter of the predictor, typically 4) are stored as they are encountered in the input. Predictions are made at each stage by extracting every sequence that matches the last k-1 characters seen. If there are none, there is no prediction. If there is a unique one, its k'th element is the predicted character. Otherwise, predict must make a choice. We have experimented with three strategies: a conservative one which chooses not to predict in this circumstance, another which predicts the most recently encountered sequence unless another sequence has been seen a significantly greater number of times, and finally, a strategy which simply selects the most recently encountered sequence.

A hash-table storage and retrieval scheme makes the operation very fast, even on a multiuser system. Techniques which are more economical in space exist[18], but these take longer to update, and speedy interaction was deemed to be the most important requirement. Some care must be taken to cope properly with rubout and other line editing characters.

Although the method may seem simple almost to the point of naivety, it does have considerable predictive power in practice. (In fact it can only cope with input sequences which are "non-counting" in the sense of McNaughton and Papert[19].) It has been used for an earlier system which implemented a "self-programming" electronic calculator[20].

However, recent research has shown that better predictive power can be obtained by using a partial-string matching method. This forms a model in exactly the same way as described above. When the model is to be used for prediction, the current context is sought in a previously-stored k-tuple in the same way. Only when it is not found does the new method differ from the old. Then, the current context is shortened by one character and previous occurrences of this new context are sought. If one is found, prediction proceeds as above; otherwise the context is further shortened and the process repeats. A prediction will fail to be made only in the case when the last character typed has not been encountered before.

One should draw a distinction between a prediction being made and one being displayed. A prediction which is unique is automatically displayed. However, a decision procedure must be used for non-unique predictions, and unless one of them overwhelms the others (in terms of frequency of occurrence in the recent past), no prediction is displayed. The decision procedure should depend to some extent on the taste of the user: he may be pleased to see even unlikely predictions displayed, or he may be put off by the bad advice which he perceives predict to offer! We plan to experiment with different decision algorithms; but the problem of setting up a sufficiently controlled experiment, and the opportunity of "personalization" for different degrees of handicap, makes evaluation difficult.

It should be noted that for satisfactory performance with the hashing scheme and for the more complex techniques necessary for partial-string matching it is necessary to have the adaptive data present in primary memory. To alleviate some of the problems this can cause on time-shared systems we envisage a personal terminal which contains the predictive interface, and is capable of being coupled to larger computer systems.

Evaluation of prediction algorithms

Consider again the behaviour sequence of Table 1. With default values (length-4 predictions) predict makes predictions as shown in Table 2. Here, "~" represents a correctly-predicted character, while "!" flags the next character as having been incorrectly predicted. The large number of good predictions is obvious. Although there are also many bad ones, notice that the user need take no action on these other than typing what he would have entered anyway.

TABLE 2 - Sample input sequence with length-4 predictions

```
cd ..
ls
cd~!modelling
ls -l
rm *.o y.tab.c
ls~~~ test2
cat te~~~
pr te~~~!opr
echo te~~!*
rm~!te~~~
ls~~~!
rm~~!ty tt~!.out
man compact
newgrp bin
cd~!..
ma~~~~~~~~~
du
cd~~~/bin
cd~!bi~
ls!
ca~~!bi!b.make
cd~!.~~!mod~~~~~~
cp ..~!b~!b~~~~~ te~~!3
cp~~~~~~!n/bi!b~~~~~~~~~~
com~~~~ t~~~~
ls~~~
od -c te~~~.c
od~~~~~~~~~!C
pr~~~~~~~~ ..~~~!n~~~!b~~~~~!!op~
pr~!..~~~!n~~~!b~~~~~~~~
od~~~~~~~~~~!!pr!op~
rm~!te~~~~~!
ls~~~ !..~~~!n!
ca~~!..~~~~/~~!b!.!indiv
cp~~~~~~!n~~~b~~~~~~ te~~!4
pr~!t~~~~!op~
co~~~~~~~~~~!
od~~~~~~~~~.C
od~~~~~~~~~~!~~~~~~
cd~~~~!predictor
```

TABLE 3 - Sample input sequence with partial-string match predictions

```
cd ..
ls
!c~~!mod!el!ling
l~ -l
rm! *.!o! y.tab.c
l~~~~ t!e!s!t2
c!a!t t~~~~
pr! t~~~~!op~
echo !t~~~!*
r~~!t~~~*
l~~~~
r~~t!ty! tt~!.!o!ut
man! comp!act
newgrp b!i~
cd~!.~
m~~~~~~~~~~
du
cd~.~/bi~
cd~b~~
l~
ca~~!b~!b.!ma!ke
cd~.~~!mo~~~~~~~
cp !.~~b~!b~~~~~ te~~3
cp~~~~~~n!/b~b~~~~~~~~~~3
co~~~~~ t~~~3
l~~~~
od! -!c te~~3.c
o~~~~~~~~3~!C
p~~~~~~3~C .~~b~n~~~~~~~~~!!~~~
p~~!.~~b~n~~~~~~~~!~~~
o~~~~~~~~~3~C!!!pr!!o~~
r~~te~~3~C
l~~~~~!.~~b~n
ca~~.~~b~n~~~~~!indiv
cp~~~~~~n~~~~~i~~~~ te~~4
p~~t~~~4!o~~
co~~~~~~~~~~4
o~~~~~~~~~4!.C
o~~~~~~~~~4.~!~~~~~~
cd~.~~predi!ct!or
```

Predict has no tacit knowledge of the lexical or syntactic structure of the behaviour sequence. It s k-tuple model is initially empty and it must incorporate part of the behaviour sequence before predictions can be make.

The use of partial-string matching can increase the number of predictions substantially. For example, with certain parameters for the decision procedure we can produce predictions as shown in Table 3. There are many more correct predictions than before, as well as some additional incorrect ones. In fact, in the length-k case, newline is treated as a delimiter in the behaviour sequence, so it and the following k-2 characters are never predicted by the length-k model (notice how the first two characters of each line of Table 2 are never predicted). This deficiency has been rectified in the partial-string algorithm, which preserves the context from one line to the next. Table 3 shows that this alone increases substantially the number of predictions.

As we have hinted above, different parameter values in the prediction process can have radical effects on the number of correct and incorrect predictions. With length-k modelling, the value of k can be varied, as can the decision algorithm which determines when a prediction is displayed. Partial-string matching is insensitive to the value of the "k" parameter but again the decision procedure can be adjusted. The parameters affect the total number of predictions, and the trade-off between the number of predictions which are correct and the number that are incorrect. The techniques described here can be used to produce a whole spectrum of prediction algorithms; the choice of which particular one is best for any given user will depend on his typing skills and also on his personal preferences.

FIGURE 1 - Performance regions for prediction algorithms

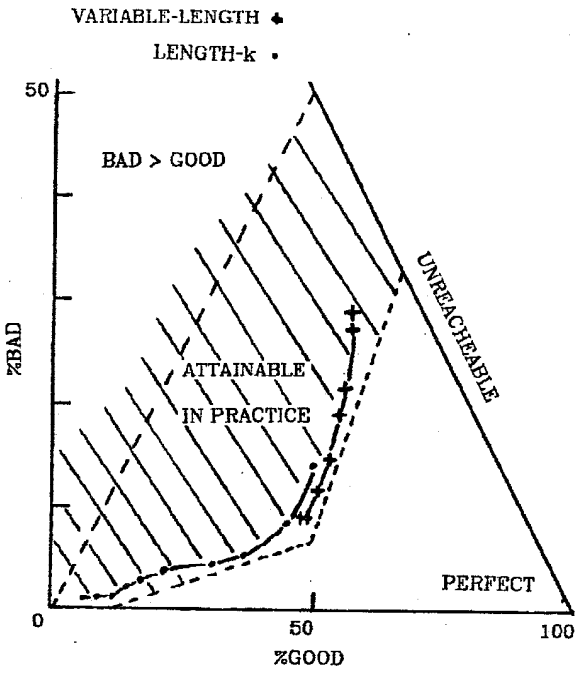

Figure 1 shows a good way to visualize this spectrum. Along the horizontal axis is plotted the percentage of characters which are predicted correctly. The vertical axis shows the percentage of incorrect predictions. The graph is bounded to the right for obvious reasons (for example, if 67% of the characters are predicted correctly, no more than 33% of them could be predicted incorrectly). Points in the upper left part of the graph correspond to large proportions of incorrect predictions; in particular, those above the diagonal have more wrong than right predictions. Points near the left side have hardly any correct predictions. The ideal operating point is the lower right-hand one, where all characters are predicted correctly (and none incorrectly). In practice, we have found that, with suitable decision procedures, our algorithms can reach the points shown in the shaded region. The precise location of this region depends to some extent on the kind of text being entered; but its general shape and position remain the same.

The two example lines shown in Figure 1 represent results with the data of Table 1. One is for the length-k method, with a particular decision procedure, and the points on it represent values of k from k=3 at the top right end of the line to k=6 at the bottom left. The other is for partial-string matching, with k fixed at 6 but

TABLE 4 - Success of length-k prediction for the behaviour of Table 1

k	correct	incorrect	unpredicted
3	50%	14%	36%
4	44%	8%	48%
5	36%	5%	59%
6	30%	4%	66%

TABLE 5 - Success of partial-string prediction for the behaviour of Table 1

decision parameter	correct	incorrect	unpredicted
0.2	57%	29%	14%
0.3	57%	27%	16%
0.4	55%	21%	24%
0.5	54%	19%	27%
0.6	52%	14%	34%
0.7	50%	11%	39%
0.8	49%	9%	42%
0.9	47%	9%	44%

varying a threshold probability. This threshold determines a decision procedure in which only predictions whose probability of occurring is estimated to be above the threshold will be displayed. Data for both of these lines, in terms of the percentage of correct and incorrect predictions, are given in Tables 4 and 5. The first and second values in each entry show the percentage of correctly and incorrectly-predicted characters, respectively; while the third gives the percentage of characters that were not predicted at all. Thus, for example, with length-4 operation, 44% of characters were correctly predicted and 8% were incorrectly predicted (corresponding to 232 and 40 characters, respectively, in the 530-character test sequence). From Table 5, with k at 6 and the threshold set to 0.9, 47% of characters were correctly predicted and 9% were incorrectly predicted (corresponding to 251 and 46 characters respectively).

We have considered the Table 1 example in detail because, unlike most techniques of adaptation by statistical induction, *predict* works well even over a short behaviour sequence. It does not rely on ergodic sources or require accurate statistics from a large training sample. In fact there is a danger that long sequences may cause the predictions to deteriorate because of excessive richness in the k-tuples encountered. To test this effect, we have run our prediction algorithms on samples of data much larger than

that of Table 1. The results are too extensive to be reported here in detail, but they all fit well within the framework shown in Figure 1.

One sample was a 4 Kbyte source program in the "C" language[21]. This language has an unusually large vocabulary of tersely-coded operators, which makes prediction difficult. Using a value of k=4, 50% of characters were predicted correctly by the length-k method, while 13% were predicted incorrectly. Partial-string matching was able to predict 53% of characters correctly, with 12% being predicted incorrectly. Other parameter settings for the decision procedure obtained different results, of course. The results quoted represent favourable choices; more correct predictions could only be obtained at the expense of a considerable increase in incorrect ones, and fewer incorrect ones caused significantly fewer correct ones.

Results are not so good for ordinary English rather than program text. On a largish (45 Kchar) scientific paper, the partial-string algorithm could only predict 50% of characters at the expense of 30% being predicted incorrectly. To reduce the number of incorrectly-predicted ones to, say, 7.5% produced a concomitant drop to 30% in the number of correct predictions.

Evaluation of the system

The use of predict has not yet been evaluated with handicapped subjects. Indeed, we have only recently begun to investigate the more sophisticated partial-string matching technique, on the grounds that it may provide more accurate predictions. However, evaluation experiments have been conducted using able-bodied subjects with the older, fixed-length matching, version, and it may be of interest to report briefly on the results.

Predict does not provide much help to good typists. For them, the time taken to decide whether a prediction is correct is often less than the time it would have taken to type the characters. Interestingly, subjects liked the system, and generally thought it was speeding up input even when it was slowing it down! Furthermore, they found it extremely easy to use, and almost never consulted the help menu that was provided. Some difficulties were experienced during the experiments because a time-shared computer was used and response times were sometimes irritatingly slow -- predict requires a fair amount of computation and should be implemented on a separate processor[22].

However, we found that for some people, particularly the slower typists, predict did increase the rate of input. One of our samples was a Cobol program, which exhibited a high degree of redundancy. This makes it favourable for predictive modelling, and the number of keystrokes was very significantly reduced using predict. Extrapolating from the data gathered in the experiment, the character rate would be increased for subjects who type at less than 0.56 char/s. Although none of our subjects came into this category (the slowest was 1.05 char/s), many handicapped people will. For example, Kelso and Vanderheiden[11] quote rates of between 0.5 and 3 words/min (0.05 to 0.3 char/s); and Clarkson and Poon[10] report 3.4 to 5.7 selections/min (.06 to .1 char/s) using unenhanced scanning techniques with single- and two-switch scanning systems.

Discussion

Predict is a completely transparent system in the sense that its predictions can be overridden effortlessly. It has considerable advantages over other methods of reducing redundancy by explicit abbreviation or by explicit invoking of a history-list search. The user need not specify anything either to make an abbreviation or to invoke it. The operation is completely automatic up to the point where he decides whether or not to accept an already-displayed prediction. This, we feel, removes the burden of constantly bearing in mind and reviewing abbreviation possibilities. Admittedly the user still has to learn how to accept predictions, but he will be strongly motivated to do so if the predictor is seen to make sensible suggestions.

The interface is only useable if the prediction can be reviewed before it takes effect. Reverse video highlighting of predicted characters provides a natural display when lines are buffered by the device driver before being executed. The technique cannot be used when keystrokes are acted upon immediately without being echoed (as in modern display editors). The harm done by accepting an incorrect prediction could be anulled by a comprehensive "undo" command, but the system would probably confuse a user much more than it helped him[23]. A different medium such as voice[24] may be suitable for announcing predictions for review in such circumstances.

The utility of a predictive interface such as the ones described here to a handicapped person will depend strongly upon the degree and nature of his handicap. Essentially, the method trades off physical keyboard activity against mental decision-making as do all such amplification techniques. Good typists find it no help because they can key faster than they can think; common keying patterns become assimilated into chunks and are evoked with negligible mental or physical effort. Those people for whom the cost of a keystroke is much higher than the cost of reviewing a prediction and making a decision will obviously appreciate the help that predict provides. It often allows a sequence of several characters to be entered as one. Some able-bodied people who are not good typists come into this category.

With a severe physical handicap, the cost of a keystroke can become quite large in relationship to the cost of reviewing predictions and making a decision. In this case a menu of alternative predictions could be displayed. The significant mental effort of scanning a menu of several alternative predictions only makes this worthwhile if the keying cost is high. There are interesting

trade-offs between the menu size (number of alternative predictions displayed) and the selection count.

The prediction technique permits a great variety of different system configurations to be built using multimodal input devices (voice or mouse, and keyboard). It raises the attractive possibility of tailoring the system to a user's particular disability. One needs to quantify the relative cost of scanning a menu (for various menu sizes), making a voice entry (for various sizes of vocabulary), and entering a keystroke (on keyboards of various sizes). Physical constraints such as screen size (for menu presentation), sizes of available keyboards, and vocabulary size for voice input need to be known also. Other factors such as the statistical character of the text being entered can, if known, be taken into account by priming predict in advance with a representative sample. With such inputs, one can imagine a procedure which optimizes the rate of information entry from a handicapped person. We need hardly say that a great deal of research needs to be undertaken before this ideal is approached.

Acknowledgements

The idea of interactive prediction owes a great deal to John Andreae and Brian Gaines; Linda Barr and Michael Saya kindly provided us with an entry into the recent literature on computing to aid the handicapped. This research is supported by the Natural Sciences and Engineering Research Council of Canada. The third author gratefully acknowledges support from the Alberta Heritage Foundation for Medical Research.

References

1. Witten, I.H. (1982) "An interactive computer terminal interface which predicts user entries" Proc IEE Conference on Man-machine Interaction, 1-5, Manchester, England, July.
2. Ritchie, D.M. and Thompson, K. (1974) "The Unix time-sharing system" Comm ACM, 17, 365-375.
3. Cleary, J.G. and Witten, I.H. (1982) "Data compression using adaptive coding and partial string matching" Research Report 82/103/22, Department of Computer Science, University of Calgary.
4. Raitzer, G.A., Vanderheiden, G.C. and Holt, C.S. (1976) "Interfacing computers for the physically handicapped - a review of international approaches" Proc National Computer Conference, 209-216.
5. Staisey, N.L., Tombaugh, J.W. and Dillon, R.F. (in press) "Videotex and the disabled" Int J Man-Machine Studies, 17.
6. Rosen, M.J. and Goodenough-Trepagnier, C. (1982) "The influence of scan dimensionality on non-vocal communication rate" Proc 5th Annual Conf on Rehabilitation Engineering, 4, Houston, Texas, Aug 22-26.
7. Gaylord, A.S., Smith, S. and Beak, P. (1981) "Text writing, speaking, and appliance control for the severely physically handicapped" Proc Johns Hopkins 1st National Search of Personal Computing to Aid the Handicapped, 178-180, Oct 31.
8. Baker, B. (1982) "MINSPEAK" Byte, 7 (9) 186-202, September.
9. Jones, R.L. (1981) "Row/column scanning with a dynamic matrix" Proc Johns Hopkins 1st National Search of Personal Computing to Aid the Handicapped, 6-8, Oct 31.
10. Clarkson, T.G. and Poon, P.E. (1982) "Utilizing eye position sensing and user defined vocabularies to enhance the communication rate of non-vocal people with severe physical impairment" Proc IEE Conference on Man-machine Interaction, 174-177, Manchester, England, July.
11. Kelso, D.P. and Vanderheiden, G.C. (1982) "Ten-branch abbreviation expansion for greater efficiency in augmentative communication systems" Proc 5th Annual Conf on Rehabilitation Engineering, 3, Houston, Texas, Aug 22-26.
12. Balesta, G.S. (1977) An intelligent communication device for the severely disabled. MS Thesis, Engineering, Tufts University.
13. Thomas, A. (1981) "Communication devices for the non vocal disabled" IEEE Computer, 25-30, January.
14. Goodenough-Trepagnier, C. and Rosen, M.J. (1982) "Optimal language menu for a one-switch non-vocal communication device" Proc 5th Annual Conf on Rehabilitation Engineering, 2, Houston, Texas, Aug 22-26.
15. Brady, M., Kelso, D.P., Vanderheiden, G.C. and Buehman, D. (1982) "A data-based approach to character/syllable/word sets" Proc 5th Annual Conf on Rehabilitation Engineering, 1, Houston, Texas, Aug 22-26.
16. Brown, D.N., Grigg, P.J., Watts, P. (1982) "A communication interface using a microcomputer for severely handicapped children" Proc IEE Conference on Man-machine Interaction, 201-204, Manchester, England, July. 17. Andreae, J.H. (1977) Thinking with the teachable machine. Academic Press, London.
18. Witten, I.H. (1979) "Approximate, non-deterministic modelling of behaviour sequences" Int J General Systems, 5, 1-12, January.
19. McNaughton, R. and Papert, S. (1971) Counter-free automata. MIT Press.
20. Witten, I.H. (1981) "Programming by example for the casual user" Proc Canadian Man-Computer Communication Conference, 105-113, Waterloo, Ontario, June.
21. Kernighan, B.W. and Ritchie, D.M. (1978) The C programming language. Prentice-Hall, Englewood Cliffs, New Jersey.
22. Vanderheiden, G.C. (1982) "Computers can play a dual role for disabled individuals" Byte, 7 (9) 136-162; September.
23. Halbert, D.C. (1981) "An example of programming by example" Technical Report, Xerox Office Products Division, Palo Alto, California.
24. Witten, I.H. (in press) Principles of computer speech. Academic Press, London.