Regular Paper

Knowledge-Based Spelling Correction in Unix Command Names

YOSHIHIKO EBIHARA*

Misspelling error correction in Unix commands and its measued data is presented as a prerequisite study for man-intelligent machine interface. An I-shell (Intelligent-shell) has been developed as part of the C-shell in the Unix system to correct misspellings. The I-shell incorporates knowledge-based dictionaries representing the characteristics of a user's keyboarding habits and performs knowledge acquisition of these characteristics for correct command prognosis. In practice, the I-shell corrected approximately 83.0% of the command misspellings and predicted the correct command on 97.3% of occasions when the correct command was among the 5 candidate commands with the highest priority.

1. Introduction

The study of computerized correction of command spelling errors in interactive processing systems has a relatively long history and still remains of considerable interest in the current research of a man-intelligent machine interface [1]. Despite a small vocabulary of Unix commands, there is still difficulty in driving the powerful and accurate correction algorithm. One of the reasons for this is due to the fact that each individual user has a different level of skill and training in computers and the correction strategy depends largely on these characteristics. Another is due to the fact that most Unix command names are short words. An earlier paper [2] emphasized that a short misspelt word was much more ambiguous than a longer one. Where a misspelling was assumed to contain one error a high rate of miscorrections by the correction algorithm occurred with respect to short command words.

This paper describes the command interpreter developed as part of the C-shell in the Unix system and is referred to here as an I-shell (Intelligent-Shell). The I-shell incorporates knowledge-based dictionaries representing the characteristics and habits of a user's keyboarding. The knowledge acquisition of a user's keyboarding behavior is dynamically performed during the operation of the I-shell. By using the acquired knowledge, the I-shell selects a plausible correction, where the corrected misspelling is limited to only command names, not including command attributes.

In practice, the I-shell corrected approximately 83.0% of the command misspellings and predicted the correct command on 97.3% of occasions when the correct command was among the 5 candidate commands with the highest priority.

Section 2 describes the concept of the correction algorithm, Section 3 shows the system structure of I-shell and Section 4 explains the knowledge-based dictionaries. The error types are defined in Section 5 and the knowledge acquisition and the error reversal methods in Section 6, with the experimental results given in Section 7.

2. General Concepts of Correction Algorithm

The command misspelling is corrected by two fundamentally different operations. The first step is to select one or more candidate commands which have been deduced by collating each type of error pattern most closely resembling the misspelling. Plausible commands are then stored in a command candidate set. We assume that an input command name contains only one error. These papers suggest that it is not as drastic a restriction as it may seem since a large percentage of misspellings in raw keyboarding typically contain one error [2–4].

The second step is to rearrange each member in the command candidate set in a plausible order. The plausibility of correction is evaluated by using the knowledge-based dictionaries. When a user indicates the correct command from the set, the knowledge data of dictionaries is renewed and the effectiveness of this correction is evaluated and measured by the I-shell.

3. System Structure of I-shell

The I-shell consists of 4 supporting processes in addition to the ordinary C-shell in the Unix system: Spelling checker, Candidate selector, Probability calculator and

*Institute of Information Sciences and Electronics, University of Tsukuba.

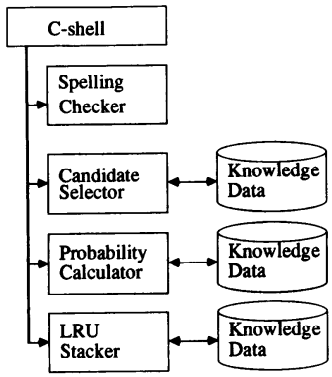

Fig. 1 System Structure of I-shell.

LRU stacker. Figure 1 illustrates the basic system structure of I-shell. In the knowledge database system, the candidate selector has knowledge data related to the error patterns of a user's keyboarding. The probability calculator has knowledge data of transition probability of a user's input commands. The LRU stacker has knowledge data of information on the recently used commands for a user. These processes are executed in parallel to reduce the processing overhead. They are activated when the C-shell finishes establishing the initialization and are stopped by signalling techniques when the C-shell is exited.

The spelling checker receives an input command data from the C-shell by the pipeline technique and examines the syntax of the incoming command. If there was an error in the command name, the candidate selector not only selects command candidates by using the misspelling patterns but also dynamically performs knowledge acquisition of misspelt patterns. The probability calculator is activated to hold a valid command history for a user whenever the incoming command is correctly executed. It calculates the command transition probability from the command history and keeps updating the value in knowledge data during its operation. The LRU stacker creates LRU command stacks for each directory used by an individual user and loads the new corresponding command stack in memory whenever the current directory is changed. It also uses the command stacks for a good command name prognosis. A command stack is a group of the five most recently used commands.

4. Knowledge-Based Dictionaries

The knowledge database system has three kinds of knowledge-based user dictionaries. They are an Error Pattern Dictionary (EPD), a Transition Probability Dictionary (TPD) and a LRU Stack Dictionary (LSD) per directory.

The EPD contains a pair of a group m numbers identifying a misspelt error pattern and an occurrence probability P_m of its misspelt error pattern. In addition, this dictionary also contains valid names of Unix standard commands and a keyboard arrangement which is used to locate adjacent letters and evaluate whether a typed letter belongs to the left or right hand (assuming that a user employs touch-typing methods) of the target key.

$Q_{ij}(n)$ is defined as a transition probability with which an incoming command j will appear at the n-th time of input commands after the command i occurred in the past.

The TPD has each file identified by a command name. The file i, identified by command name i, consists of the name of the command j, and a transition probability value, $Q_{ij}(1)$, where command j occurs immediately after the previous command i.

The LSD is prepared with a pair of a name of command i and its transition probability value, $R_{ii}(n)$, $n=1\sim5$, per dictionary.

5. Error Types

In Papers [3, 4], four basic spelling error patterns are identified: omission, insertion, substitution and transposition. In addition to these four patterns, we have defined four more subgroups in insertion, substitution and transposition, respectively. Thus, in total 13 error patterns ($m=13$) have been classified to reverse a misspelling as shown in Table 1.

A definition for each error pattern is explained in the following.

(1) Omission is defined as one character left out of the string of a command name. This misspelling belongs to subgroup 1 in Table 1.

(2) In the case of insertion, there are four subgroups of misspelling error patterns: subgroup 2—

Table 1 Classification of Error Patterns.

Group	m	Subgroup
Omission	1	One omission
Insertion	2	One extra adjacent character
	3	A chattering
	4	One repetition
	5	Others
Substitution	6	A substitution in the left hand keys
	7	A substitution in the right hand keys
	8	A substitution between the left and the right hand keys
	9	Others
Transposition	10	An interchange between two adjacent characters
	11	An interchange between the small and the capital letters
	12	A chattering transposition
	13	Others

one extra adjacent character is inserted in the string; subgroup 3—a chattering of a double letter; subgroup 4—one extra character which has previously appeared in the string is inserted; subgroup 5—other insertions other than the above three error patterns.

(3) In the case of substitution, there are also four subgroups: subgroup 6—a substitution between two left hand keys; subgroup 7—a substitution between two right hand keys; subgroup 8—a substitution between a left and a right hand key; subgroup 9—all other substitutions.

(4) There are also four subgroups for transposition: subgroup 10—an interchange of two adjacent characters; subgroup 11—an interchange between a capital and a small letter of the same character; subgroup 12—a chattering transposition; subgroup 13—other transpositions. The chattering transposition is caused by at least one repetition of a key stroke. Here, it is treated as an exception. For example, if one typed "little" as "litlle", it may be regarded as one omission and one chattering insertion. However, for practical purposes we classified this kind of misspelling as a chattering transposition.

6. Error Reversal

Here we explain the error pattern knowledge, command transition and command locality. Knowledge acquisition and error reversal methods are also described in this section.

6.1 Knowledge of Error Patterns

The most suitable correction strategy for Unix command names depends on both the nature of a small vocabulary with short words and the misspelling characteristics of an individual user's keyboarding. Thus, the I-shell has been adapted to positively learn misspelling characteristics. Knowledge acquisition is executed as follows. Whenever the correct member of the command candidate set was determined by the user, the I-shell examines it to discover which error pattern caused the misspelling and dynamically updates the value of the corresponding error pattern probability.

There is a trade-off between a quick response and accuracy of correction. For practical purposes, the number of error patterns is limited to 13 groups. A misspelling is examined in order of an error pattern with the highest occurrence probability and is transformed into one or more Unix commands by collating error patterns most closely resembling the misspelling. 13 error patterns are executed by a single application of the corresponding error operation. If necessary, the error operations use the similar string-distance measurement technique [3, 5, 6], differing in length by 0, 1.

6.2 Knowledge of Command Transition

A strong correlation between specific commands exists in the command sequences. The correlation varies with the individual characteristics of a user. Thus, the I-shell prepares the TPD for each user at the time of login. It dynamiclly updates the value of transition probability in the dictionary whenever the user utilizes a valid command. For example, when the current command j was correctly executed, the I-shell can calculate $Q_{ij}(1)$ as it memorized the previous command i. Then, it updates its value in the dictionary. The Unix system is supported by more than 300 standard commands. However, we have selected the 300 standard commands with the highest rate of usage. Neither the remaining Unix command nor the editing subcommands are included in the dictionary. The size of each user's dictionary approaches the order of 300×300.

The large part of mean processing time for a command is represented by the mean processing time of a valid command because valid commands occupy the most of the total commands. Thus, the mean processing time is measured from the time the C-shell sends an incoming command to the spelling checker until the time the probability calculator completes to update the TPD. The measurement was executed when almost 10 terminals were connected to a Sequent S81 (Main Memory 24 Mbytes) and very few background jobs were running. It takes the mean processing time of $70 \sim 120$ mS to examine an incoming command and update the transition probability. This includes the pure processing time of $10 \sim 15$ mS plus the disk I/O processing time of $60 \sim 105$ mS.

6.3 Knowledge of Command Locality

By looking closely at a user's history of command sequences, it was found that a specific command that occurred in the past will recur in the very near future. This means that a locality of command reference exists. A typical example is that a combination of the "emacs" editing command and the "make" command appear very frequently when a user is developing a program. Another phenomena is that the command history depends not only on user characteristics but also on what job the user is doing on the computer [7]. In other words, the number and kinds of commands used vary according to different directories. This locality will be included as knowledge data to improve a plausible correction. When the LRU stacker puts a valid command i on the LRU stack, it learns the distance n of the current command i from the previous command i and updates dynamically the value with the new probability $R_{ii}(n)$ in the LSD. The command i in the command candidate set has the value of $R_{ii}(n)$ only when the previous command i is found in the current LRU stack.

6.4 Priority Decisions

When the command candidate set has been retrieved, each member is tested for plausibility of corrections by a plausible vaue. The plausible value is calculated as follows. If command j followed by the previous command i is one candidate and the same command j occurred in the past n times of input commands, the plausible value T_j of command j is simply yielded by a sum of each of the probabilities.

$$T_j = P_m + Q_{ij}(1) + R_{ij}(n),\ n = 1 \sim 5 \qquad (1)$$

Thus, each member of the command candidate set is sorted in the right place in the order of the highest plausible value.

7. Experimental Results

In order to examine the effectiveness of the knowledge-based error correction, several experimental measurements have been conducted by installing the I-shell in a Unix system. Collection of data was started at the stage of steady status after one month's running of the I-shell for knowledge acquisition of user's characteristics. The three experimentees we chose are graduate students who can touch-type. They used 80 different kinds of commands on an average. The total number of used commands is about 2,000 for a user. During the measurement period, they programmed for system development of almost the same job. Thus, it seems to be all right to consider that the variation of different percentage for the error patterns among them mainly comes from their personal experiences rather than the differences of job content. The measured results are shown in the following.

(1) Error detection

Figure 2 shows the number of candidates and its percentage as an average, achieved by using only the error detection technique without any knowledge data. The results show that many candidates are ambiguously selected in this correction operation. The percentage of correction was approximately 14.8%.

(2) Effect of the EPD

By using the EPD, the I-shell determines the value of probability for each member of the command candidate set according to the error pattern of misspelling. A mean occurrence probability of each error pattern is illustrated in Table 2 and the variation of error pattern frequency among the three users is also shown in Fig. 3. Table 3 shows the effectiveness of the EPD in correction operation. The result explains that this operation applied by the EPD in effect corrected 54.7% of misspellings and predicted the correct one on 86.5% of occasions when the correct command was among the 5 candidates with the highest priority. It is concluded that this correction operation performs with a higher ac-

Table 2 Occurrence Probability of Error Ptterns.

Group	m	Subgroup	Probability $\times 10^{-3}$
Omission	1	One omission	184
Insertion	2	One extra adjacent character	163
	3	A chattering	56
	4	One repetition	28
	5	Others	72
Substitution	6	A substitution in the left hand keys	32
	7	A substitution in the right hand keys	5
	8	A substitution between the left and the right hand keys	187
	9	Others	5
Transposition	10	An interchange between two adjacent characters	221
	12	A chattering transposition	2
	13	Others	16

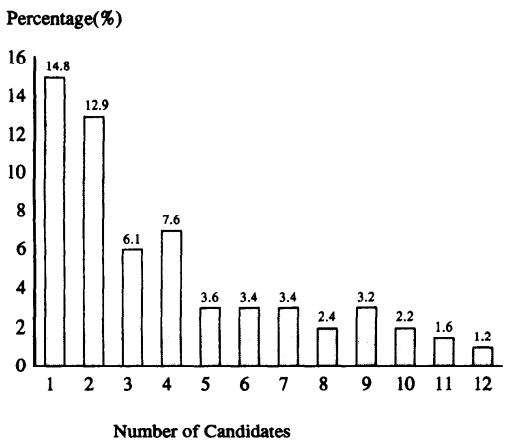

Fig. 2 Number of Candidates for the Error Detection.

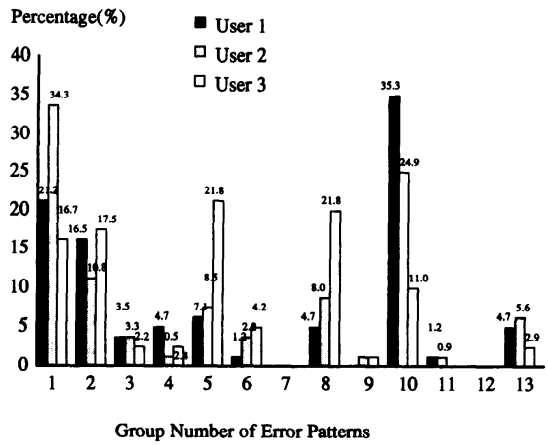

Fig. 3 Error Patterns of the Three Users.

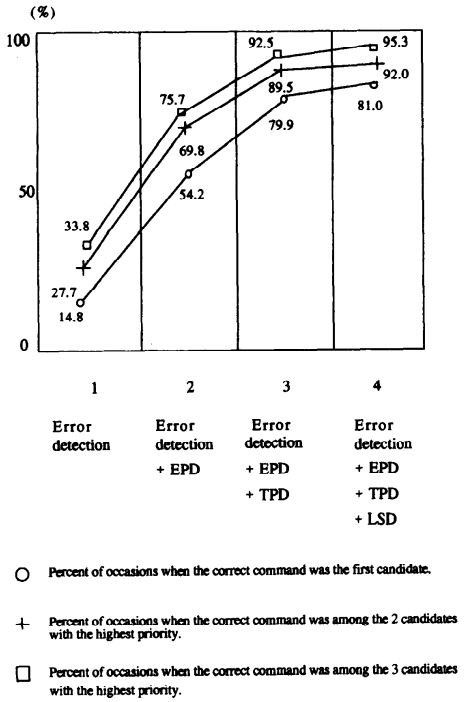

Fig. 4 Effect of the Combined Dictionaries.

curacy of correction than when only the error detection operation is executed, even though there is still liable to be a slight variation between the users.

(3) Effect of the TPD

By referencing the TPD, the I-shell selects the value of transition probability for the candidates and sorts them according to the highest order. Table 4 shows the effectiveness of the TPD. On the average, the command dictionary alone corrected 32.7% of misspellings and predicted on 68.4% of occasions if the 5 command candidates with the highest priority are included. The results suggest that the variation of the correction accuracy among the users has improved in comparison to that of the EPD, although the plausible correction rate remains low.

(4) Effect of the LSD

Table 5 illustrates the comparison of error correction effectiveness between the stand-alone operation of the TPD and the combined knowledge that includes the TPD and LSD. By applying the LSD to the correction operation, the accuracy of error corrections has been slightly improved by approximately 5% on an average. With the two dictionaries together, the I-shell forecasts the correct command on an average 73.3% of occasions if the 5 command candidates with the highest priority are included.

(5) Effect of the combined dictionaries

By applying the three dictionaries together, the I-shell

Table 3 Effect of the Error Pattern Dictionary.

Precent (%)

Priority	1	2	3	4	5	Total
User 1	63.4	5.2	13.1	3.9	4.6	90.2
User 2	53.2	16.9	3.9	0.1	2.6	76.7
User 3	47.4	32.3	3.2	5.0	4.8	92.7
Average	54.7	18.1	6.7	3.0	4.0	86.5

Table 4 Effect of the Transition Probability Dictionary.

Percent (%)

Priority	1	2	3	4	5	Total
User 1	34.0	18.0	13.0	7.0	2.0	74.0
User 2	25.0	18.0	10.0	8.0	5.0	66.0
User 3	39.0	13.0	6.0	5.0	2.0	65.0
Average	32.7	16.3	9.7	6.7	3.0	68.4

Table 5 Effect of the LRU Stack Dictionary.

Percent (%)

	LSD+TPD	Diffrence from the effect of TPD
User 1	75.0	1.0
User 2	75.0	9.0
User 3	70.0	5.0
Average	73.3	4.9

Table 6 Effect of the Combined Dictionaries for the Three Users.

Percent (%)

Priority	1	2	3	4	5	Total
User 1	79.7	13.7	3.9	0.7	0.7	98.7
User 2	84.3	9.3	1.0	0.5	1.0	96.1
User 3	84.9	8.1	2.9	0.5	0.5	96.9
Average	83.0	10.4	2.6	0.6	0.7	97.3

corrected on an average 83.0% of the misspellings and predicted the correct command on 97.3% of the time if the 5 command candidates with the highest priority were counted in the percentage as shown in Table 6.

Finally, Figure 4 illustrates an improvement process in the case of the general public is also included. It shows how an improvement of error corrections has been achieved as an average when the three dictionaries are added one by one to the correction operation.

8. Conclusions

The I-shell corrected approximately 83.0% of the command misspellings and predicted the correct command on 97.3% of occasions when the correct command was among the 5 command candidates with the highest priority. This measurement data shows that the knowledge-based error correction algorithm is useful

and practical.

The I-shell has been developed as part of the C-shell in the Unix system and required a slight modification of the C-shell by using the signal and pipeline techniques. Thus, the I-shell has an advantage in that the impact of modification on system reliability is much smaller compared to a program being built from scratch. From this viewpoint of system implementation, I-shell implementation is more practical and useful than other methods.

The I-shell required an average processing time of about 160 mS per input command. This fact still guarantees the quick response time for an interactive processing system.

The knowledge-based error correction algorithm is based on the simple sum of each probability and works well in the correction. The logical and reasonable explanations on this affair still remain to be investigated by further research.

References

1. IKEDA, K. Man-Intelligent Machine Interface, *J. of IECE*, **69**, 11 (1985), 1160-1166.
2. POLLOCK, J. J. and ZAMORA, A. Collection and Characterization of Spelling Errors in Scientific and Scholarly Text, *J. Am. Soc. Inf. Sci.*, **34** (1983), 511-58.
3. POLLOCK, J. J. and ZAMORA, A. Automatic Spelling Correction in Scientific and Scholarly Text, *Comm. ACM*, **27**, 4 (1984), 358-368.
4. GATES, A. I. Spelling Difficulties in 3867 Words, *Bureau of Publications*, Teachers College, Columbia University, New York (1937).
5. HALL, P. A. V. and DOWLING, G. R. Approximate String Matching, *Comput. surv.*, **12**, 4 (1980), 381-402.
6. ULLMANN, J. R. A Binary N-Gram Technique for Automatic Correction of Substitution, Deletion, Insertion and Reversal Errors in Words, *Comput. J.*, **20**, 2 (1977), 141-147.
7. TAKANO, S., EBIHARA, Y. and IKEDA, K. Intelligent Man-Machine Interface with Knowledge of Keyboard-Array, Command-History and User-Habits, *Proc. of the 2nd Annual convention AI Japan* (1988), 505-508.

(Received July 23, 1991; revised February 10, 1992)