Restorable Backspace

Sunjun Kim, Geehyuk Lee
Human-Computer Interaction Lab, KAIST
291 Daehakro, Yuseong, Daejeon, 305-701, South Korea
{kuaa.net, geehyuk}@gmail.com

ABSTRACT
This paper presents *Restorable Backspace*, an input helper for mistyping correction. It stores characters deleted by backspace keystrokes, and restores them in the retyping phase. We developed *Restoration* algorithm that compares deleted characters and retyped characters, and makes a suggestion while retyping. In a pilot study we could observe the algorithm work as expected for most of the cases. All participants in the pilot study showed satisfaction about the concept of *Restorable Backspace*.

Author Keywords
Restorable Backspace; typing errors; error correction; word suggestion

ACM Classification Keywords
H.5.2 [Information interfaces and presentation]: User Interfaces. – Interaction styles.

INTRODUCTION
Users mistype all the time. Users correct mistyped characters as they notice them while typing. Typically, mistyped characters are corrected in one of the following two methods. In the first method, users use a backspace key to erase up to the mistyped characters and type again (*backspace method*). In the second method, users move the cursor to the mistyped characters, correct them, and restore the cursor back (*cursor method*). These two correction methods have different types of cost. In the *backspace method*, the major cost is retyping correctly inputted characters that have been erased by backspace keystrokes. In the *cursor method*, the major cost is cursor keystrokes or pointing operations for moving the cursor to the mistyped characters. Selection between these two methods will depend on the relative weights of the two different types of cost.

In a mobile touchscreen environment, the cost of moving the cursor is often relatively high. For example, the so-called *fat finger problem* [2] makes it difficult for users to place a cursor exactly at an insertion point quickly. For example, users often make multiple taps just to place a cursor at the right position. Considering the relatively high cost of moving the cursor, the *backspace method* may be a preferred way of correction in a mobile touchscreen environment. A concern about the *backspace method*, however, is the cost of retyping correctly inputted characters as mentioned earlier. In the paper, we introduce *Restorable Backspace*, an input helper for the *backspace method* that aims to reduce the cost of retyping correctly inputted characters, thereby making the *backspace method* even more preferred way of correction in a mobile touchscreen environment.

UIST '12, October 7–10, 2012, Cambridge, Massachusetts, USA.
ACM 978-1-4503-1582-1/12/10.

RESTORABLE BACKSPACE
Restorable Backspace stores characters deleted by backspace keystrokes, and restores them in the retyping phase. Figure 1a illustrates the basic idea of *Restorable Backspace* with an example: a user types "wrong characters" by mistake and corrects it to "right characters". First, a user backspaces up to 'w'. As characters are deleted, the system stores them in a buffer (backspace buffer in the figure). Then, he or she starts to type again. While he or she types, the system compares the new keystrokes with the stored characters in the backspace buffer using *Restoration* algorithm. If the system sees that some characters are restorable, it makes a suggestion. In the example of Figure 1, the system detects that some characters are restorable after the input "right ch", and suggests "aracters" as a preview. Then, the user can accept the suggestion by using *Restore* command (e.g., typing a TAB key or making a swipe gesture). Otherwise, continuous typing will simply discard the suggestion.

Restoration algorithm returns *restorable* (a string) which it thinks needs to be retyped. This algorithm requires two inputs: *inputted* and *buffered*. *Inputted* is correcting keystrokes and *buffered* is backspaced keystrokes. Figure 1b illustrates the detailed procedure of *Restoration* algorithm. Assume that $k = 2$. In this example, *inputted* is "right ch" and *buffered* is "wrong characters". Then, *chunk* is the last two characters of *inputted,* so it is "ch" in this example. Because there are only one "ch" occurrence in *buffered*, *restorable* is simply the rest of *buffered,* which is "aracters".

If there are two or more occurrence of *chunk* in *buffered*, then *restorable* is calculated from the occurrence of *chunk* which leads to the smallest Minimum String Distance (MSD) [3] between *buffered* and the potentially restorable sentence. MSD measures the minimum number of insertions, deletions, or substitutions required to transform one string to another. Therefore, the smallest MSD maximizes the similarity between *buffered* and the restored sentence.

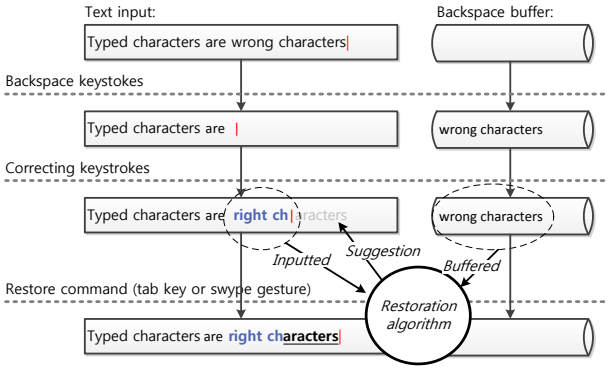
As an example, consider the following case:

Typed Text: `resd books with ebook`

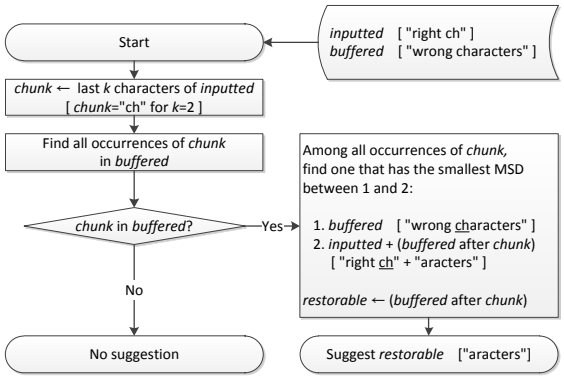
Deleted Text: ~~`resd books with ebook`~~

Retyped Text: `read bo`

In this case, *inputted* is "ead bo" and *buffered* is "esd books with ebook". *Chunk* is set to "bo" and two "bo"s are found (underlined) in *buffered*. For the first "bo", MSD between *buffered* and "ead books with ebook" is 1 and, for the second "bo", MSD between *buffered* and "ead book" is 13. Therefore, *restorable* is set to "oks with ebook".



(a)



(b)

Figure 1: *Restorable Backspace*: (a) an example sequence (A user changes "wrong characters" to "right characters".) and (b) *Restoration* algorithm (example values in brackets)

PILOT STUDY

We conducted a pilot study to observe that the proposed algorithm is working as intended and to confirm that the concept of *Restorable Backspace* is acceptable to users. Four participants (all male, ages from 26 to 29) performed a transcribing task with *Restorable Backspace* on 11" MacBook Air for 30 minutes. All participants were touch-typists. Sentences were randomly selected from the MacKenzie and Soukoreff phrase set [1]. The parameter k of *Restoration* algorithm was set to 2 throughout the pilot study because we wanted the algorithm to make suggestions as early as possible but, with $k = 1$, suggestions were too frequent and often incorrect.

Throughout the pilot study, 18 suggestions were accepted by participants and they expressed favorable opinions about the concept. One participant said "It feels great because my effort of typing was not wasted." From logged data and video recording, we could observe the followings:

1. *Restorable Backspace* worked well in most cases. Among the 18 accepted suggestions, 17 suggestions (for 10 insertion errors, 6 omission errors, and 1 substitution error) recovered right words successfully.
2. It did not work in some cases. For example, a participant typed "needs" instead of "knees". He deleted it and typed "knee". Then, "ds" appeared as a suggestion. This was not a correct suggestion, but he accepted it unconsciously, and corrected it again.
3. Participants often stayed stalled for a while after automatic restoration. One participant said "I lost my flow of typing." Especially, they delayed more when the final word of the restored text was incomplete.

CONCLUSION AND DISCUSSION

We proposed *Restorable Backspace* concept and implemented *Restoration* algorithm for it. All participants in a pilot study showed satisfaction about the new concept.

As mentioned in the beginning, *Restorable Backspace* will be more useful in a mobile touchscreen environment. First of all, as we discussed in the introduction, *backspace method* is more likely in this environment and therefore more correctly inputted characters will be deleted. Another reason is that users in a mobile environment focus more on the onscreen keyboard than on the text input field, and therefore errors are more likely to be detected belatedly.

Although the algorithm worked fine for most of the cases, there is a lot of room for refinement. For example, the current algorithm suggests all characters in the buffer at once. As people often treat one word at a time, suggesting one word at a time may be more effective. Also, problems observed in the second and the third observations emphasize the necessity of a word level analysis and algorithm.

ACKNOWLEDGEMENT

This work was supported by the IT R&D program of MKE/KEIT. [10039161, Development of core technologies for high-physicality control interfaces and personalized intelligent user interfaces based on Smart TV user experience]

REFERENCES

1. MacKenzie, I. S., and Soukoreff, R. W. Phrase sets for evaluating text entry techniques. *CHI 2003 extended abstracts*.
2. Siek, K., Rogers, Y., and Connelly, K. Fat finger worries: How older and younger users physically interact with pdas. *INTERACT 2005*.
3. Soukoreff, R. W., and MacKenzie, I. S. Measuring errors in text entry tasks: an application of the levenshtein string distance statistic. *CHI 2001 extended abstracts*.