Visual Clue: An Approach to Predict and Highlight Next Character

Manoj Kumar Sharma[1], Sayan Sarcar[2], Pradipta Kumar Saha[3] and Debasis Samanta[4]
IIT Kharagpur, Kharagpur, West Bengal, India - 721302
Email: [1]manojsharma.net@gmail.com, mailtosayan@gmail.com[2], [3]itsmepradipta@gmail.com, [4]debasis.samanta.iitkgp@gmail.com

***Abstract*—The motivation of this research is to develop a user friendly word prediction system augmented with virtual keyboard in the context of Indian languages. The objective would be not only to save keystrokes but also to reduce the cognitive load to compose the text accurately. In the context of Indian language, in addition to large alphabets sizes there are a lot of phonetically or graphically similar characters which needs more time to search the desired characters and occasionally leads to taping wrong characters. This issue can be addressed by highlighting the next probable characters and thus offering visual clue and mitigating errors. The proposed approach not only help in avoiding the error, it also highlight the required character when user have misspelled part of word while composing text.**

***Index Terms*—Human computer interaction, Visual clue, Predictive virtual keyboard**

I. Introduction

The composition of text in Indian languages has several issues. It contains a large number of alphabets, matra, halant and complex characters (yuktakshar) etc. [1], [2]. So, typing in an Indian language with conventional Qwerty keyboard is not an easy task as significant training is required to compose the text [3]. As an alternative to hardware Qwerty keyboard, researchers introduce virtual keyboard. A virtual keyboard is an on screen graphical display where keys are spatially arranged and can be tapped with finger tip or mouse pointer. Further, text entry rate with virtual keyboard is less compared with hardware keyboard. In addition, it becomes a major issue in the context of Indian languages where text entry rate through virtual keyboard is typically 3 - 5 wpm [4] compared to 10 - 15 wpm in English [5], [6]. Further, the performance of virtual keyboard can be improved with a text entry rate enhancement strategy such as word prediction [6], [7]. However, performance of predictive keyboard (i.e virtual keyboard with prediction) depends on many factors such as dictionary size, language characteristic, size and position of prediction window, block of text to be predicted, search strategy, prediction method etc. [8]–[10].

In context of Hindi, national language of India, follows Devanagari script[1]. It contains a large set of characters (13 vowels, 33 consonants, 12 matras and special symbols like anusvara, visarga, chandra bindu and nukta etc.) [11], [12] along with complex characters called "conjuncts" (character composed with two or more characters and "halant") [13]. Further, there are some graphically and phonetically similar characters [14] which increase the finite chance of confusion to select the proper character and occasionally lead toward tapping wrong characters. The above mentioned complexities make text composition task erroneous and time consuming, as a consequence yield poor text entry rate.

Finding a target character in predictive keyboard containing large character set demands a high reaction time. According to the Hick-Hyman law [15], [16], the reaction time is directly proportional to the number of objects in the interface. In other words, if the number of keys is large, then reaction time of visually searching a character increases moderately. To reduce the reaction time, one of the possible solutions is to contain a minimum number of keys. However, designers advocate to contain all keys necessary to compose text within a single layout. It is therefore a challenging task to meet both the requirements.

In recent literature, Joshi et al. [3] uses the color code to differentiate the consonant and vowel blocks in hardware keyboard like *Keylekh2* and *Keylekh3*. Whereas, another system *Guruji*[2] uses the color code to distinguish between consonants, vowels and numeric keys to provide a visual assistance in virtual keyboard. Some system [17], [18] highlight the next character in virtual keyboard at runtime. System developed by Laurent Magnien et. al. [18] highlight the next probable keys in bold after each keystroke at the time of typing. The prediction of character is based on a French dictionary of 1462 words using lexical tree. They concludes that visual clues improve the text-entry rate. However, the improvement tends to decrease when previously entered text is error-prone.

To address the above problem, we propose a method called *predicting next character highlighter* (*PNCH*) which provides visual clue by highlighting next probable character(s) in the predictive keyboard interface. Following are the objectives behind the development of *PNCH*.

- To provide visual assistance in composing texts and improve text entry rate.
- To reduce the search space by predicting next probable characters which may require for composing a word.
- To highlight the best suitable characters even in presence of some typographical error in the initial entry of a word.

[1]Devanagari script, http://en.wikipedia.org/wiki/Devanagari

[2]http://www.guruji.com/hi/index.html

978-1-4673-4369-5/12/$31.00

- To develop a framework which can be easily plugged to any predictive virtual keyboard in Indian languages.
- To develop a mechanism which will help used to learn a new keyboard layout.

II. Proposed methodology

Development of *PNCH* is described in this section. We start our discussion with basic framework of PNCH followed by highlighting issues in candidate character computation. Next we present few terminology involved in our work. Then we discussed the steps involved in candidate character computation and its filtration in Algorithm walkthrough. Section 3 present the experimental result. Finally Section 4 concludes the paper.

A. Basic framework of the PNCH

The basic idea behind the *PNCH* is as follows. Let *SetA* consist of N_0 number of characters (the keys) present in a predictive virtual keyboard. We partitioned the *SetA* into two different set namely *SetB* and *SetC*. *SetB* consists of N_1 number of characters and represents the number of possible characters we may be interested to type next. Whereas, *SetC* contains the remaining characters ($N_2 = N_0 - N_1$) in the predictive virtual keyboard. As the number of next probable characters in a given context is less than the total keys present in the predictive virtual keyboard, so we can assume that $N_1 < N_2$. When a user needs to select a character from the predictive virtual keyboard, he first looks into *SetB*, if the target character is not found in this set then the user searches desired character in *SetC*. If the user finds the required character most of time in *SetB*, then the reaction time will be reduced, as this reduces the search space from N_0 to N_1.

The framework of augmenting visual clue with the word prediction system is shown in Fig. 1. This have two major module namely: prediction module and PNCH module. Prediction module are designed in such a way to handle to typing error and suggest the suitable candidate list of word. Whereas, PNCH takes input from this word prediction module and perform some addition processing to identify the possible set of next character need to be typed while composing the text. When a user enters a character through the predictive virtual keyboard, various scores are calculated based on phonetic or graphical similarity, spelling error and ngram probability [19]. These scores are then accumulated and sent for ranking the final list of the candidate words. After ranking is performed, we extract top seven words and display them in the prediction window. The generated list (say *CL*) acts as an input to the *PNCH* module. For each word in *CL*, let it be denoted as α, we need to identify the characters to be highlighted (see Fig. 2(a)). The *PNCH* module generates the list of the best candidates to be highlighted in the predictive virtual keyboard.

The development of *PNCH* consist of two modules: a) identification of candidate characters and b) filtration of candidate characters. We start our discussion with issues in candidate character identification.

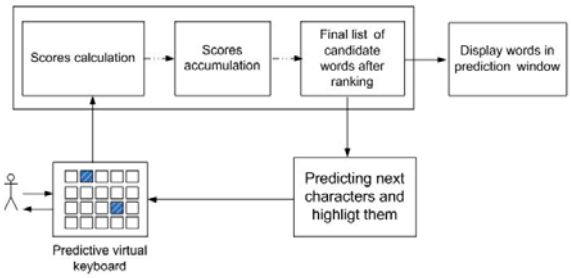


Fig. 1: Augmentation of PNCH with the proposed word prediction system

B. Issues in identification of candidate characters

The generated candidate list according to the word prediction method and character sequence entered by a user at any instance (say β) are taken as input to this task. Here, word prediction module suggests word based on word level trigram, phonetic or graphical similarity between typed and desired word, and typographical error. Suppose after entering β the desired word is not coming in prediction window, so next character is needed to be typed. We take each candidate word from this list and given β, to identify the candidate characters for highlighting (see Fig. 2(a)). As there exist two different situations while composing a text:

- User has typed the character sequence correctly.
- The typed sequence contains some typographical error.

This section illustrates the issues in identifying the candidate character to be highlighted. Suppose, typed sequence and candidate words are represented as β and α, respectively. Let β contains n number of characters. In the first case, the required next character for highlighting can be obtained at $(n+1)^{th}$ position in α. Whereas, in the second case, identification of next character to be highlighted become challenging due to the presence of error. Note that a typographical error may occur due to insertion, deletion and substitution of character(s). So, the desire character is not always at the $(n+1)^{th}$ position.

For example, β is मधय and α is मध्यप्रदेश. Here, β consist of three characters and α have 10 characters (see Fig. 2(b)). Suppose, for instance, user has omitted the *halant* while composing part of α as shown in β. When we look at $3+1$, that is, 4^{th} character in α it is य, but this character is already entered by the user in β and it should not be highlighted again. As in this case, our best candidate is प which is located at 5^{th} position instead of character य at 4^{th} position.

In the following, we define few terminology followed by our approach.

C. Some definitions

In this section, we briefly mention these terminologies.

CSID: Character set ID or *CSID* is unique ID for set of phonetically or graphically equivalent character(s). For example श, ष and स all are mapped to character श (see the Table I). Hence the character set ID (or $CSID$) represents the character which denotes this set here, $CSID$ for ष is श.

TABLE I: Graphical or phonetic similar sets of characters in Hindi

CSID	Similarity set			
अ	अ	आ	ा	
इ	इ	ई	ि	ी
उ	उ	ऊ	ु	ू
ऋ	ऋ	ॠ	ृ	
ए	ए	ऐ	ॅ	ॆ
ओ	ओ	औ	ॉ	ॊ
अं	अं	अँ		
क	क	ख		
ग	ग	घ		
च	च	छ		
ज	ज	झ		
ट	ट	ठ		
ड	ड	ढ		

CSID	Similarity set			
त	त	थ		
द	द	ध		
न	न	ञ	ण	ं
प	प	फ		
ब	ब	व	भ	
म	म			
य	य			
र	र			
ल	ल			
श	श	ष	स	
ह	ह			
◌	◌			
◌	◌			

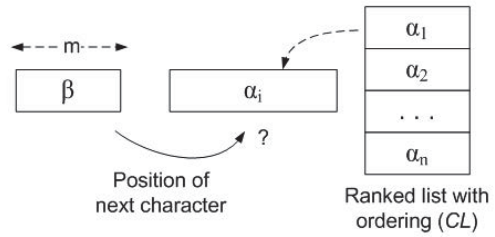
Processed CSID$_{seq}$: It represent a sequence of CSID after some processing. Computation of Processed $CSID_{seq}$ is described in Algorithm 1. Here, α represent the word whose $CSID_{seq}$ is required to be computed.

Algorithm 1 $CSID_{seq}$ computation

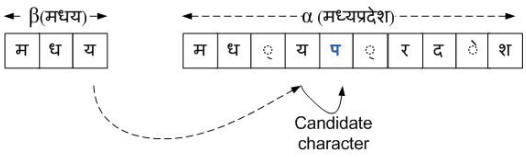
Input: Words α
Output: Returns $CSID_{seq}$ for α
1: $\gamma \leftarrow$ Convert each entry in α into its respective $CSID$
2: $CSID_{seq} \leftarrow$ Remove "*halant*" followed by deletion of consecutive duplicate *CSIDs* in γ
3: Return $CSID_{seq}$

Let β, α, μ and ω be the sequence of typed characters, candidate word, *processed* $CSID_{seq}$ of β and α, respectively. The symbols and there meaning are summarized in Table II.

Sliding Window: A *Sliding Window* provide a range to cover $(n-1)$, n and $(n+1)^{th}$ character in ω where n is the number of characters in μ. Let C_0 refers to some specific character which we are interested in. We need to search the occurrence of C_0 within this sliding window.



(a) Identification of next character



(b) Pointing to correct character

Fig. 2: Next character identification

TABLE II: List of symbols used and there meaning

Terms	Description
$CSID$	Character set ID
α	Candidate word
β	Sequence of character typed towards composing word
β'	Normalize value of β after removal of ZWJ and ZWNJ
C_0	Some specific characters
CH	Stores the value C_0
μ	*Processed* $CSID_{seq}$ of β
ω	*Processed* $CSID_{seq}$ of α
n	Number of characters
τ	Candidate character
Pos	Position of candidate character
CL	Initial candidate list from prediction window

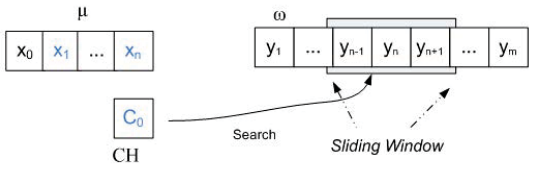


Fig. 3: Sliding Window

Window Shift counter ($WindowShift_{counter}$): It represents the number of shifts the *Sliding Window* requires to find a desired character (say C_0) in ω.

Location counter ($Location_{Counter}$): It indicates the position of C_0 in α, where $C_0 \in \omega$.

Halant counter ($Halant_{counter}$): It indicates the number of *halants* in β after the last occurrence of C_0 in β.

D. Algorithm walkthrough

Our procedure to compute candidate characters for highlighting is precisely stated in Algorithm 2. We take a list (i.e. CL) from prediction module, the character sequence entered by the user (denoted as β) and number of characters (N) to be highlighted as input. The steps involved in computation of candidate character is as follows:

We first normalize the β (unicode normalization[3]) and delete all the occurrence of "ZWJ" (Zero width joiner[4]) and "ZWNJ" (Zero width non joiner), if they are present, and store it into β' (Step 1 in Algorithm 2). Next, we take the list of candidate list (CL) from the prediction module and take out one word at a time (see Fig. 2a). Let this word be denoted as α here, α_i indicate i^{th} instance of the candidate words in the list CL. We convert both β' and α_i into its respective *processed* $CSID_{seq}$ and store in μ and ω, respectively (see Fig 4 and Steps 5 and 6 in Algorithm 2). Next, we compute the $WindowShift_{counter}$ using Algorithm 3 and retrieve the value of C_0. Once we properly identify the value of $WindowShift_{counter}$ and C_0,

[3]http://unicode.org/reports/tr15/
[4]http://unicode.org/review/pr-27.html

Algorithm 2 PNCH

Input: β, CL and N are typed sequence of characters, list of candidate words from prediction module and number of characters to be highlighted in the keyboard, respectively.
Output: Returns a set of candidate characters in L_{final} for highlighting.

1: $\beta^{'}$ = Normalize (β) /* Unicode normalization for β and removal of all the occurrence of "ZWJ" and "ZWNJ" */
2: **while** ($flag == true$) **do**
3: **for** ($i = 0; i \leq Length(CL) - 1; i++$) **do**
4: $\alpha_i = CL[i]$ /* Get candidate word from list CL */
5: μ = *Processed* $CSID_{seq}$ for $\beta^{'}$
6: ω = *Processed* $CSID_{seq}$ for α_i
7: Compute $WindowShift_{counter}$ and C_0 for $\beta^{'}$ and α_i
8: Compute $Location_{counter}$
9: Compute $Halant_{counter}$
10: $Pos = WindowShift_{counter} + Location_{counter} + Halant_{counter} + 1$
11: $\tau_i = \alpha_i[Pos]$ /* τ_i contains candidate character for α_i word */
12: /* Filter the candidate character for highlighting */
13: **if** (L_{final} does not contain τ_i) **then**
14: Add τ_i into L_{final}
15: $Count++$
16: /* Check the number of character to be highlighted */
17: **if** $Count == N$ **then**
18: flag= $false$
19: /* Skip further computation if N number of character is identified */
20: Break
21: **end if**
22: **end if**
23: **end for**
24: **end while**
25: Return L_{final}

we use these data for computation of $Location_{counter}$ and $Halant_{counter}$ using Algorithm 4 and Algorithm 5. Next, we compute the position of target characters using Pos (Step 10 in Algorithm 2) . Finally, characters can be obtained using $\tau_i = \alpha_i[Pos]$.

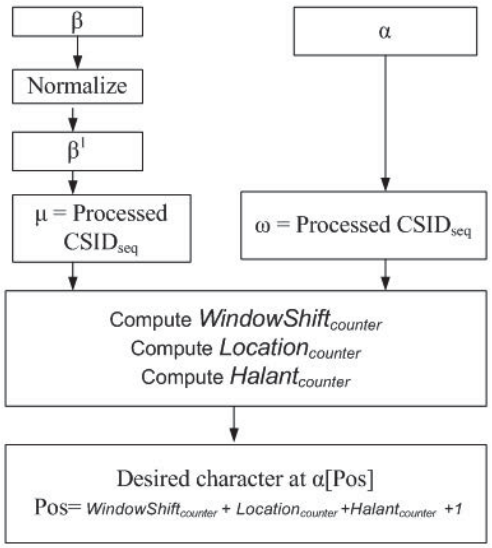


Fig. 4: Steps involved in the computation of candidate character

We take a list L_{final} which is initially empty and append this obtained character τ_i into L_{final} if this character is not already present in the list. This will also maintain the order given by the prediction module. We simultaneously check the number of character present in L_{final} is less than the N (Steps 13 to 22 in Algorithm 2). If a sufficient number of characters are computed, then we stop processing. Further, we highlight all characters present in L_{final} into the predictive virtual keyboard in their respective order.

Algorithm 3 takes α, $\beta^{'}$, μ and ω as candidate word, normalized typed sequence of character and processed $CSID_{seq}$ for $\beta^{'}$ and α, respectively. Suppose, n number of characters are present in μ, we store the last character in μ into C_0 (Step 2 in Algorithm 3). Now, we need to initialize the size of *SlidingWindow*. It starts from $SlidingWindow_{start}$ and stops at $SlidingWindow_{stop}$. $SlidingWindow_{start}$ will be initialized to 0 when n is 1, otherwise, it will be set to $(n-1)$ (Steps 3 to 7). Whereas, $SlidingWindow_{stop}$ is calculated as $(n+1)$. Now we match the content of C_0 in this *SlidingWindow*. When we do not find any match, we shift the *SlidingWindow* by one towards left by decreasing the value of n (Step 17) and increment the $WindowShift_{counter}$. Each time *SlidingWindow* is adjusted we update the value of C_0 (Step 1 and 2 in Algorithm 3).

We use Algorithm 4 for the computation of $Location_{counter}$. It takes α, $\beta^{'}$, C_0 as candidate word, normalized typed sequence of characters and specific character set from $WindowShift_{counter}$ module (using Algorithm 3), respectively. Each character present in $\beta^{'}$ is first converted to its respective $CSID$ and then matched with C_0. We record the number of matches into Occ. Next, we convert each character of α into its respective $CSID$ and match with C_0. We find the Occ^{th} occurrence of C_0 into α and stored into $Location_{counter}$.

Algorithm 5 is used to compute the $Halant_{counter}$ which takes input $\beta^{'}$ and C_0. We convert each character of $\beta^{'}$ into its respective $CSID$. Next, we find the match of C_0 into this converted $CSID$ from right-hand side. Once the position of match is known, we extract the chunk of string from this data

Algorithm 3 Computation of $WindowShift_{counter}$

Input: α, $\beta^{'}$, μ and ω are candidate word, normalized typed sequence of characters and processed $CSID_{seq}$ for $\beta^{'}$ and α, respectively.
Output: Set value of C_0 and returns $WindowShift_{counter}$.

1: **while** ($flag == false$) **do**
2: $C_0 = \mu[n]$ /* Update C_0 from μ */
3: **if** ($n \leq 1$) **then** /* Specifying the range for *Sliding Window* */
4: $SlidingWindow_{Start} = 0$
5: **else**
6: $SlidingWindow_{Start} = n - 1$
7: **end if**
8: $SlidingWindow_{Stop} = n + 1$
9: /* Finding match between C_0 and sequence of character in *Sliding Window* */
10: **for** ($i = SlidingWindow_{Start}; i \leq SlidingWindow_{Stop}; i++$) **do**
11: **if** ($C_0 == \omega[i]$) **then**
12: $flag$=true
13: Break
14: **end if**
15: **end for**
16: **if** ($flag == false$) **then**
17: $n = n - 1$ /* Shift *Sliding Window* */
18: $WindowShift_{counter}++$
19: **end if**
20: **end while**
21: Return $WindowShift_{counter}$

Algorithm 4 Computation of $Location_{counter}$.

Input: α, $\beta^{'}$, C_0 are candidate word, normalized typed sequence of characters and specific character from $WindowShift_{counter}$ module, respectively.
Output: Returns $Location_{counter}$

```
for (j = 0; j ≤ length(β') − 1; j++) do
    if (C0 == CSID(β'[j])) then
        Occ++    /*Count the occurrence of C0 in converted CSID of β' */
    end if
end for
for (j = 0; j ≤ length(α) − 1; j++) do
    if (C0 == CSID(α[j])) then
        count++
        /*Locate Occ^th occurrence of C0 in converted CSID of α */
        if (count == Occ) then
            Location_counter = j
            Break
        end if
    end if
end for
Return Location_counter
```

Algorithm 5 Computation of $Halant_{counter}$

Input: $\beta^{'}$ and C_0 are normalized typed sequence of characters and specific character from $WindowShift_{counter}$ module, respectively.
Output: Returns a value for $Halant_{counter}$

```
Stop=length(β') − 1        /* Get position of last character in β' */
for (j =Stop; j ≥ 0; j−−) do    /* Scan β' from right to left*/
    if (C0 == CSID(β'[j])) then
        Start=j        /* Identify position of correct match of C0 in β' */
        Break
    end if
end for
Copy the part of β' from Start to Stop into Chunk
Count occurrence of halant in Chunk and store into Halant_counter
Return Halant_counter
```

Fig. 5: A snapshot of $*hInidA$ augmented with PNCH

and count the occurrences of *halant* in it.

In our work, we have selected top seven characters to be highlighted in the virtual keyboard (i.e $N = 7$). The seven characters are highlighted into two different colors. The top three most probable characters are marked in one color and the rest four characters are marked in other color.

For example, suppose a user enters β as **शब** (where his target is to compose α_i which is **शब्द**. The next probable character is predicted and highlighted as shown in Fig. 5. The required character, which is ् (*halant*), is properly predicted, as it is matra so it combines with last consonant and highlighted **ब्** while displaying in virtual keyboard.

Further, example for identification of candidate characters in presence of typing errors are shown in Table III. Here, α represents the target word and β is the typed sequence of characters; ω and μ are processed $CSID_{seq}$ for α and β.

TABLE III: Generating candidate character in various conditions in PNCH

Sl.	Description	β	μ	α	ω	C_0	τ
1	Error free	कब	कब	कबुतर	कबउतर	ब	ु
2	Deletion	उतप	उतप	उत्पादन	उतपअदन	प	ा
3	Deletion	मधय	मधय	मध्यप्रदेश	मधयपरदएश	य	प
4	Substitution	उतश	उतस	उत्साह	उतशअह	स	ा
5	Substitution	उतक	उतक	उत्साह	उतशअह	क	ा
6	Insertion	हांथिय	हअनतइय	हाथियों	हअतइयओन	य	ो

Suppose, we consider a deletion error where α is **उत्पादन** (consisting of seven characters) and β is **उतप** (consisting of three characters). Note that ् (*halant*) is missed in β for composing α. The proposed PNCH module correctly identifies the character ा (represented as τ) in α at the fifth position. Further, this table demonstrates identification of τ in error-free condition as well as with deletion, substitution and insertion error.

III. Experiment and Experimental Results

All experiments in this work have been carried out on a Window 7 operating system, Intel Core 2 Duo processor with 2 GB primary memory. The proposed algorithms and Hindi predictive virtual keyboard have been developed using $C\#$ 3.5 in the Visual Studio 2008 environment. A web version of developed system is hosted at Human Computer Interaction Lab, Indian Institute of Technology, Kharagpur, India. It can be accessed at http://www.nid.iitkgp.ernet.in/Manoj/WordPrediction/. We refer the developed word prediction system as **hIndiA*.

To evaluate the performance of *PNCH*, we use two systems namely $hIndiA$ and $*hIndiA$ [19] and conduct test using with and without *PNCH*. In $hIndiA$, use of word prediction is disabled whereas the user can select a word from the prediction window in case of $*hIndiA$. In Table V, say *Case 1* indicates text composition with $hIndiA$ without *PNCH*, whereas *Case 2* indicates the situation when *PNCH* was augmented with $hIndiA$. Similarly, *Case 3* and *Case 4* represent the use of word prediction without and with *PNCH* approach, respectively.

TABLE V: PNCH on different conditions

System	PNCH	
	Without	With
hInidA	Case 1	Case 2
*hIndiA	Case 3	Case 4

We use few metrics to measure the performance of proposed *PNCH* method. These metric are described below.

A. Metrics for performance measure

The metrics used to evaluate *PNCH* are as follow.

Text entry rate: This metric represents the number of words a user can enter in one minute and expressed in *wpm*.

TABLE IV: Summary of benchmark texts used in experiments

Domain	Text ID	Reference Text	Domain of Text	Number of Sentences	Number of Words
Out-of-domain	H_1	*Godan* by Munsi Premchand	Novel	14	204
	H_2	*Dohri Zindagi* by Vijaydan Detha	Story book	13	191
	H_3	*Patrakarita: ek parichay* by Sandip Kumar Srivastava	Journalism article	17	261
	H_4	*Atmakatha* by Rajendra Prasad	Autobiography	15	223
In-domain	H_5	*Wikipedia*	Miscellaneous	16	247

$Pnch_{hit}$*:* It is referred to the percentage of times the next character is correctly highlighted.

PKS: This metric represents the number of keystrokes required by simulation program to compose a text.

The calculation of *PKS* also includes one additional keystroke required to select a word from the prediction list as in Wolf et. al. [20].

B. Performance evaluation

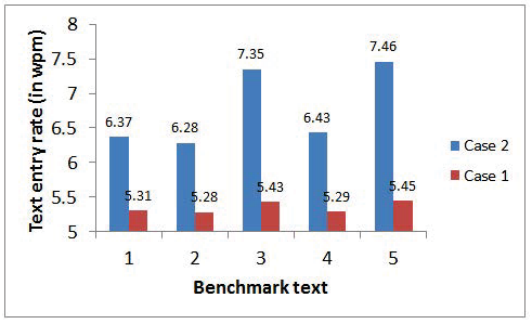
We have performed both empirical and simulated evaluation for *PNCH* in different conditions, and they are described below.

Empirical evaluation We have selected 14 users to evaluate the performance of *PNCH*. The users are instructed to use five benchmark texts shown in Table IV following the *Read and Type (RT)* and *Listen and Type (LT)* method for text composition. We have conducted the test on four different conditions (see Table V). The text composition task is repeated five times for each user.

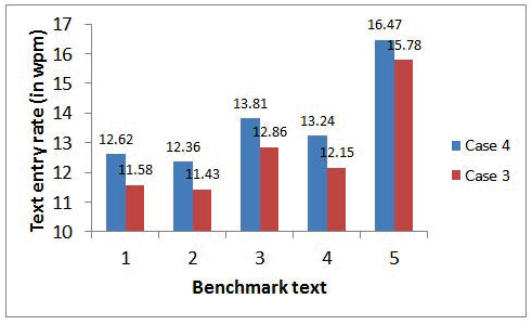
We observe that augmentation of *PNCH* increases the text entry rate of $hIndiA$ to 26.64% (i.e. *Case 2* over *Case 1*). In *Case 2*, we also observe that, text entry rate of *in-domain* text is higher compared to *out-of-domain* text (see Fig. 6(a)). In case of $*hIndiA$, that is, text entry using word prediction with and without *PNCH*, only 7.37% of improvement is observed (see Fig. 6(b)).

Simulated evaluation: In this method, a simulation component has been developed which reads words from the different benchmark texts (see Table IV) and passes them character by character to the word prediction system developed by [19]. It processes and populates the word in the prediction list. Whenever the prediction list contains the target word, it accepts the word and processed the next word until the text is complete. If the word is presented in the prediction list and selected by the simulated component, it automatically increments the hit count and appends space in the composed text. We record *PKS*, *KuP* and *HR* into log file.

To understand the effect of $Pnch_{hit}$ and keystroke savings (*PKS*) on different modes of simulation, we have conducted the test on *Mode 1* and *Mode 2*. In *Mode 1*, simulation program reads a word from the benchmark texts and composes the text without any error in it. In *Mode 2*, the simulation program also takes the location number where an error can be incorporated (let it be the i^{th} position). The program reads the word from the benchmark texts and randomly chooses a character and



(a) $hIndiA$ with and without PNCH



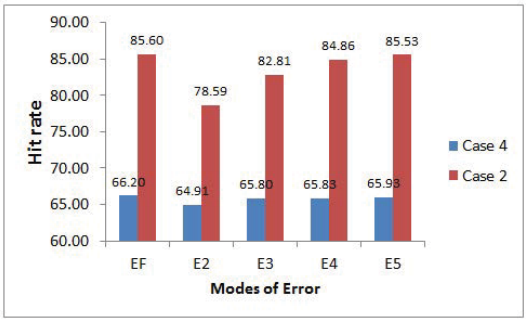
(b) $*hIndiA$ with and without PNCH

Fig. 6: Performance of the systems with and without PNCH

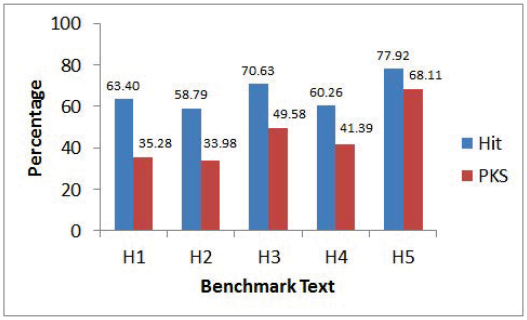
alter it with the character at i^{th} position in the word. Here, $i > 1$, that is, error can be at second position or onward.

We observed that in *Case 1* and *Mode 1* (i.e. using $hIndiA$ in error-free condition), the proposed system predicts the character correctly 85.60% of times on five benchmark text (i.e $Pnch_{hit} = 85.60\%$). Whereas, in *Mode 2*, that is, in E_2, E_3, E_4 and E_5 condition, the $Pnch_{hit}$ are 78.59%, 82.81%, 84.86% and 85.53% (as shown in Fig. 7(a)), but there are no keystroke savings.

The simulation study on *Case 4*, that is, $*hIndiA$ with *PNCH*, reveals that the $Pnch_{hit}$ is 66.20% on *Mode 1*, that is, in error-free condition (see Fig. 7(a)). On the other hand, it is about 64.9%, 65.80%, 65.89% and 65.93% in E_2, E_3, E_4 and E_5 conditions. However, this $Pnch_{hit}$ comes with the saving of keystroke required to compose the text. Finally, the relation between $Pnch_{hit}$ and keystroke savings are shown in Fig. 7(b). We can observe that 77.92% of character is correctly predicted with keystroke savings of 68.11% in case of *In-domain* text (i.e. in H_5). However, in case of H_1, we achieve only 63.40% of $Pnch_{hit}$ and keystroke savings of 35.28%.



(a) Hit rate achieved between $hIndiA$ and $*hIndiA$ on different modes of simulation



(b) Hit rate and PKS

Fig. 7: Hit rate and keystroke savings when word prediction is used

IV. Conclusion

A new mechanism called *PNCH* has been proposed to reduce visual search time and hence improve text entry rate. While composing text, *PNCH* predicts the next possible character set which may be required and highlight them into virtual keyboard. Unlike the existing character highlighter, which work on tree structure and fails when there is error in initial entry of a word, the proposed *PNCH* augmented with word prediction system predict the correct character even in presence of error. It also guide the user to select the proper character if the desired word is present in prediction list and missed by user to select. The *PNCH* mechanism has two parts, identifying the suitable character and filtering suitable candidates. Special attention is needed while identifying the candidate character as composed text may contains typing error. We also observed the there is relation between $pnch_{hit}$ and keystroke saving.

As observed from the experimental result, the augmentation of *PNCH* increases the text entry rate of $hIndiA$ upto 26.64% over the interface not having *PNCH* facility. In case of $*hIndiA$, only 7.37% of text entry improvement is observed using word prediction with and without *PNCH*. Accessing $hIndiA$ in error-free condition, the proposed system correctly predicts the character for 85.60% times on five benchmark text. On the other hand, the simulation study on $*hIndiA$ with *PNCH* signifies that the $Pnch_{hit}$ becomes 66.20% on error-free condition. However, this $Pnch_{hit}$ comes with the saving of keystroke required to compose the text.

We observe that *PNCH* module also helps user to enter a character correctly, before he gets confused and causes an error. So, with the help of the highlighter, it is being assured that wrong text composition will be minimized. It is important to note that the proposed visual clue works even in presence of typing error. Most of the time user can get desired character as highlighted. Hence, it can be selected more quickly and as a result, text-entry rate increases.

References

[1] O. N. Koul, *Modern Hindi Grammar*. Indian Institute of Language Studies, 2008.

[2] T. Mohanan, *Argument Structure in Hindi*. CA, USA: Center for the Study of Language and Information, Leland Stanford Junior University, 1994.

[3] A. Joshi, A. Ganu, A. Chand, and V. P. G. Mathur, "Keylekh: a Keyboard for Text Entry in Indic Scripts," in *Extended Abstracts on Human factors in Computing Systems (CHI)*. New York, USA: ACM, 2004, pp. 928–942.

[4] S. Sarcar, S. Ghosh, P. K. Saha, and D. Samanta, "Virtual Keyboard Design: State of the Arts and Research Issues," in *IEEE Students' Technology Symposium*. Kharagpur, India: IEEE, 2010, pp. 289–299.

[5] I. S. MacKenzie, I. S. Shawn, X. Zhang, and R. W. Soukoreff, "Text Entry Using Soft Keyboards," Behaviour & Information Technology, 18(4), 235-244., 1999.

[6] K. Trnka, J. McCaw, D. Yarrington, K. F. McCoy, and C. Pennington, "User Interaction with Word Prediction: The Effects of Prediction Quality," *ACM Transaction on Accessible Computing*, vol. 1, no. 3, pp. 1–34, 2009.

[7] A. Fazly, "The Use of Syntax in Word Completion Utilities," Master's thesis, Department of Computer Science, University of Toronto, 2002.

[8] N. Garay-Vitoria and J. Abascal, "Text Prediction Systems: a Survey," *Universal Access in the Information Society*, vol. 4, no. 3, pp. 188–203, 2006.

[9] M. Herold, "The Use of Word Prediction as a Tool to Accelerate the Typing Speed and Increase the Spelling Accuracy of Primary School Children," Ph.D. dissertation, University of Pretoria, 2004.

[10] M. K. Sharma, S. Dey, P. K. Saha, and D. Samanta, "Parameters Effecting the Predictive Virtual Keyboard," in *IEEE Students' Technology Symposium*, April 2010, pp. 268–275.

[11] T. U. Consortium, "Unicode Detail," 2009, available: http://www.unicode.org, Accessed on June 2010.

[12] A. Gupta and G. Jamal, "An Analysis of Reading Errors of Dyslexic Readers in Hindi and English," *Asia Pacific Disability Rehabilitation Journal*, vol. 17, no. 1, pp. 73–86, 2006.

[13] I. S. MacKenzie and K. Tanaka-Ishii, *Text Entry Systems: Mobility, Accessibility, Universality*. Morgan Kaufmann, 2007.

[14] P. K. Ghosh and D. E. Knuth, "An Approach to Type Design and Text Composition in Indian Scripts," Ph.D. dissertation, Stanford University, 1983.

[15] W. E. Hick, "On the Rate of Gain of Information," *The Quarterly Journal of Experimental Psychology*, vol. 4, no. 1, pp. 11–26, 1952.

[16] R. Hyman, "Stimulus Information as a Determinant of Reaction Time," *Journal of Experimental Psychology*, vol. 45, no. 3, pp. 188–196, 1953.

[17] J. Gong, B. Haggerty, and P. Tarasewich, "An enhanced multitap text entry method with predictive next-letter highlighting," in *CHI'05 extended abstracts on Human factors in computing systems*. ACM, 2005, pp. 1399–1402.

[18] L. Magnien, J. Bouraoui, and N. Vigouroux, "Mobile Text Input with Soft Keyboards: Optimization by Means of Visual Clues," *Mobile Human-Computer Interaction–MobileHCI 2004*, vol. 3160, pp. 197–218, 2004.

[19] M. K. Sharma, "Word Prediction System with Virtual Keyboard for Text Entry in Hindi," Master's thesis, Indian Institute of Technology Kharagpur, India, 2012.

[20] E. Wolf, S. Vembu, and T. Miller, "On the Use of Topic Models for Word Completion," *Advances in Natural Language Processing*, vol. 4139, pp. 500–511, 2006.