A New Input Technique for Accented Letters in Alphabetical Scripts

Uwe Waldmann
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken, GERMANY
+49 681 9325 227
uwe@mpi-sb.mpg.de

Abstract

SITMO is a new input technique for accented and special letters of the Latin alphabet (or other alphabets of comparable size), which combines in a uniform way short key sequences for frequently used characters with an easily memorizable scheme to enter rarely used characters. Compared with traditional modifier techniques, SITMO requires less additional keys and allows to access more characters, while for most European languages, the average number of keystrokes per derived letter is similar (that is, close to 2).

1 The Problem

For most computer users worldwide, the number of characters that they need to input more or less regularly exceeds the number of (possibly shifted) keys on a keyboard. This has been true since the transition from 7-bit character codes (ASCII and its national variants, with 95 printable characters) to 8-bit character codes, and it is even more of a problem in the age of Unicode. While the number of keys on commercially available keyboards has increased during the last decades, the number of keys that can be put on a newly designed keyboard is obviously limited, and the number of keys that can be conveniently accessed from the home row is even more limited. As a simple one-to-one translation from individual keystrokes to characters is impossible, using more complex input methods is unavoidable.

There is no uniform input method that is well-suited for, say, entering Czech text (one alphabet, 82 letters), classical Greek text (one alphabet, at least 166 letters), mixed English and Russian text (two alphabets, 116 letters), and Japanese text (four writing systems, at least some thousands of letters and ideographs). These different tasks clearly require different solutions (cf. Leisher [4]). In this paper we restrict to the case of a single language using the Latin alphabet (or another alphabet of comparable size), plus a small number of diacritical marks occurring in native words, plus a possibly large number of diacritical marks occurring only in borrowed words, proper nouns, quotes from other languages, etc. Our goal is to find a technique to enter accented and special letters efficiently and conveniently using a conventional keyboard.

2 Terminology

We call a character a *base character*, if it has its own key on the keyboard (possibly shifted), and a *non-base character*, otherwise. Often, a non-base character c_0 can be associated in some natural way with a base character c_1, for instance because c_0 is typographically similar to c_1, or because c_0 is commonly replaced by c_1 (or by a sequence $c_1 \dots c_n$ of base characters) if c_0 is not available in a given character set. In such a case, we call c_0 a character *derived from* c_1.

We emphasize that the division into base characters and derived characters need not match the conventions of a particular language or orthography. For instance, "ö" is considered as a first-class letter in Swedish, as a second-class letter in German, and as the letter "o" with a diacritical mark indicating separate pronunciation in Dutch. For our purposes, this distinction is irrelevant: In the context of a keyboard that does not have an [ö] key we would always take "ö" as a character derived from "o".

Unless explicitly said otherwise, we assume in this paper that the set of base letters is $\{\mathrm{A}, \dots, \mathrm{Z}, \mathrm{a}, \dots, \mathrm{z}\}$. Our results apply mutatis mutandis to other sets of base letters, say $\{\mathrm{A}, \dots, \mathrm{Z}, \mathrm{Å}, \mathrm{Ä}, \mathrm{Ö}, \mathrm{a}, \dots, \mathrm{z}, \mathrm{å}, \mathrm{ä}, \mathrm{ö}\}$ or $\{\mathrm{A}, \dots, \Omega, \alpha, \dots, \omega\}$.

3 Requirements

3.1 Primary Requirements

The primary requirements that an input method should satisfy can be summarized as follows. (As we will see in the sequel, these desiderata are not fully compatible.)

- *It should allow fast typing with a small number of errors.*
- *It should be easy to learn and to memorize.*
- *It should induce little mental and physical stress.*
- *It should have a large scope.*
- *It should have the prefix property.*
- *It should be usable independently of language-specific hardware.*

While the first three objectives are obvious, the last three ones need perhaps some more explanation and motivation:

The scope of an input method is the set of characters for which it is applicable. Ideally, an input method is complete – it allows to access *all* characters of the given character set. For large character sets such as Unicode, however, this demand may be rather high. Of course, in practice completeness can always be reached by supplementing a given incomplete input method with a complete fallback method. This destroys uniformity, however (and is thus detrimental to memorizability).

An input method has the prefix property, if there is no pair of characters that are to be entered using key or event sequences that are proper prefixes of each other. If the input method satisfies this technical condition, then an application can always determine unambiguously whether a character has been entered completely or not. Obviously this is useful for menu selections and yes/no questions without trailing [Return]. Furthermore, it is necessary if the input method is to be implemented transparently as a finite transducer, so

that application programs see only the completely entered characters, not the intermediate steps. For example, an input method with the prefix property can be integrated into a window system in such a way that applications need not even be aware of the existence of the input method, whereas an input method without the prefix property requires at least some kind of cooperation between the applications and the window system.

Independence from language-specific hardware is desirable since we can not assume that everybody who is writing in a particular language uses a keyboard that is specifically tuned to the orthography of that language – first, since there are users who switch between different languages (and who prefer not having to switch the physical keyboard, too); second, since some users want to write in languages that are used so rarely (globally, or in a particular region) that specialized keyboards are not commercially available.

3.2 Consequences

What are the concrete consequences of the general requirements listed above?

To improve learnability and memorizability and to reduce mental stress and the frequency of typing errors, the key sequences should of course be *mnemonic*. Note that, if the input method is sufficiently *uniform*, learning its general principle tends to be rather trivial. Learning and memorizing how to input individual characters may be a problem, though, in particular for infrequently used characters.[1] Furthermore, it is advantageous if the key sequences are *natural*, that is, if they correspond intuitively to the way in which the characters are written by hand.

Short key sequences, at least for frequent letters, are indispensable for rapid typing. While it is clear that shorter key sequences imply a smaller number of keystrokes for a given text, this is not the only reason: Long key sequences carry also a much higher risk of interrupting the user's train of thought, and the memory recovery process that is necessary afterwards adds to the mental stress and slows down the typist in addition. On the other hand, for a rarely used character, a long but mnemonic key sequence seems to be preferable to a short key sequence that has to be looked up in the manual.

It should be mentioned here that there is no user-independent notion of "rarely used characters": Even though the frequency of the letters "ř" or "ć" in average French texts is close to zero, a French author writing about Czech composers or Balkan politics may need to use these characters rather often. It is therefore necessary that the input method is *suitable for individualization*.

To avoid typing errors, the input method should furthermore be *unambiguous*. In other words, users must know what character they enter or have entered. This may look trivial – but it is not, if some characters in the character repertoire are visually indistinguishable.

To simplify learning and to reduce mental stress and the number of typing errors, it is also desirable that the input method *does not interfere* with the default way of using the keyboard: Anything that can be typed with the input method disabled using a certain key sequence should also be typable using the same key sequence if the input method is enabled.

Conveniently located keys are necessary both for fast typing and to reduce the physical stress. Having to move fingers far away from the home row (or even worse, having to move the hand away from the keyboard to reach the mouse) slows down typing.

Chording, that is, simultaneous pressing of several keys[2] is a double-edged sword. Un-

[1] For instance, it is easy to remember that accented characters are entered by typing first Compose, then the accent, then the base letter. It is not so obvious that in the absence of a ° key, the character "å" is obtained by typing Compose * a.

[2] For instance, using the Shift key to produce uppercase letters.

der optimal conditions – if the typist is sufficiently trained and the keys can be conveniently reached – the overhead for the second key is small. For an untrained typist, or for key combinations that require heavy distortion of the hand(s), both the typing time and the physical stress may be even larger than for two sequential keystrokes.[3]

In the rest of this paper, we abstract from the concrete positions of the keys on the keyboard: Every input method can be spoiled by assigning a crucial function to a sufficiently inconveniently located key. Furthermore, personal taste, anatomy, and previous accustoming have a great influence on which key positions are considered convenient. On the other hand, the number of keys that can be put on a keyboard is clearly limited, and the number of keys that can be conveniently accessed from the home row is even more limited. Usurping a key already in use means that some other character becomes less easily typable, so the problem is deferred, but not solved. It is therefore desirable that the input method *requires few (additional) keys* – the more keys it requires, the more likely some of them will be located inconveniently.

4 Traditional Techniques

The vast majority of keyboard-based approaches to tackle our problem falls into five groups.

A *one-to-one* translation from keystrokes to characters is the simplest way to deal with derived characters: The derived character gets its own dedicated key on the keyboard, that is, it is turned into a base character.

The second group of approaches uses some kind of *modifier* keys, also called dead keys. Two variants are common: Either there is one modifier key per accent, say [¨] for the umlaut (two dots) accent, [`] for the grave accent, and so on. Then the letter "ö" can for instance be typed as [¨] [o] ("leading modifiers", e. g. using the input method *latin-1-prefix* in the editor Emacs) or [o] [¨] ("trailing modifiers", e. g. using the input method *latin-1-postfix* in Emacs). Alternatively, one restricts to a a single modifier key (usually with chording, say [AltGr-press] [o] [AltGr-release]). The additional derived characters that can be obtained in this way are then often engraved in the key caps.

The third group is characterized by using one special *compose* key for initiating a (more or less) *mnemonic* key sequence that yields a derived character. For instance, "ö" can be typed as [Compose] [¨] [o] on a Sun keyboard. Again there are numerous variants of this scheme: Microsoft Word uses [Ctrl] with chording instead of [Compose] and [:] instead of [¨], so that "ö" is typed as [Ctrl-press] [:] [Ctrl-release] [o]; ISO 9995-3 defines [AltGr-press] [[] [AltGr-release] [o]. Sometimes, a composite key sequence takes the role of the compose key, e. g. in Emacs [Ctrl-press] [x] [Ctrl-release] [8] [¨] [o].

Numeric input schemes constitute the fourth group of approaches. They allow to access a character by entering its code number in a given character set. Again, there are several variants, with or without chording, using decimal digits (Microsoft Windows: [Alt-press] [0] [2] [4] [6] [Alt-release]), octal digits (Emacs: [Ctrl-press] [q] [Ctrl-release] [3] [6] [6]), or hexadecimal digits (ISO 14755: [Shift-press] [Ctrl-press] [f] [6] [Ctrl-release] [Shift-release] [3]).

Permanently switching from one input method ("keyboard") to another one can be considered as a meta-method; usually it combines several one-to-one or modifier techniques.

Apart from these keyboard-based approaches, there are selection-based techniques, where characters are copied from some panel (for instance the Microsoft Windows "Char-

[3] The ergonomic problems of chording keys can be alleviated by duplicating the keys so that they can be pressed alternatively with the left or the right hand. (It is highly regrettable that the [AltGr] key is *not* duplicated on standard PC keyboards.)

acter Map") to the editing region using the mouse or another pointing device, or where the mouse is used to switch permanently between different keyboards.

How do the traditional techniques compare with respect to the requirements listed before? Obviously the numeric schemes have the largest scope,[4] followed by mnemonic compose, followed by modifier methods. One-to-one translation is only applicable for extremely few characters.

Another advantage of the numeric schemes is shared by the mnemonic compose method: both occupy only one additional key. By contrast, modifier approaches usually occupy one key per diacritical mark, and one-to-one methods need even one key per character. (Even then the new base character will most likely displace some other character, and unless one-to-one methods are combined with permanent switching, the problem is only deferred.)

When we compare the techniques with respect to mnemonicity and to the length of the key sequences, the order is essentially reversed. One-to-one techniques are the most mnemonic (provided that the characters are engraved in the key caps), numeric techniques are the least mnemonic, and modifier and mnemonic compose methods range in between. The length of the key sequences is 1 for one-to-one input methods, at least 2 for modifier methods, and at least 3 for mnemonic compose methods; numeric schemes require generally the largest number of keystrokes. Permanent switching is rather poor for isolated characters (at least one keystroke to switch, one keystroke to type the character, one keystroke to switch back), but it becomes much better for longer runs.

Leading and trailing modifier methods differ essentially in two ways: Trailing modifiers follow the model of handwriting more closely, where the base letter is written before the accent. They are thus more natural than leading modifiers. On the other hand, in contrast to all other traditional input methods, trailing modifier methods do not have the prefix property, so that the implementation is more expensive.

Numeric schemes and one-to-one methods are the two extremes among the traditional techniques. It is obvious that both the short key sequences of one-to-one methods and the large scope of numeric schemes do not come for free – and it is also obvious that both properties are out of reach for other approaches. In the next section we will show, however, that the relative advantages of (trailing) modifier and mnemonic compose methods are *not* mutually exclusive: The input method that we will introduce yields natural, mnemonic and short key sequences (like trailing modifier methods), still it requires only a single additional key (like mnemonic compose methods) and provides a relatively large scope.

5 The SITMO Technique

5.1 The Idea

The SITMO (*Single Iteratable Trailing Modifier*) technique requires a single additional key labelled [Accent] on the keyboard. Typing [Accent] replaces the character immediately before the cursor by another character, usually derived from the same base character. For instance, [Accent] might replace "a" by "ä", "ä" by "à", and "à" by "á". The cursor is not moved during this operation, so it is possible to type the letter "à" using the key sequence [a] [Accent] [Accent]. Typing [Ctrl-press] [Accent] [Ctrl-release] gives rise to the inverse transformation, hence [a] [Accent] [Accent] [Ctrl-press] [Accent] [Ctrl-release] and [a] [Accent] are two ways to produce the letter "ä".

[4] For a character set as large as Unicode, numeric schemes seem to be the only keyboard-based input methods that are both uniform and complete.

Having to type [Accent] a dozen of times may look like a rather clumsy way to access a single letter. However, as we will show below, being able to access frequently used letters using only two or three keystrokes greatly outweighs this inconvenience.

The SITMO technique is parameterized by a "replacement scheme". To define the latter, we first need the notion of a "row": A *row* is a string in which no character occurs more than once. Usually the first character of a row is a base character and all further characters are derived from this base character.

Example 1 *A row starting with the base character "a", followed by ten derived characters:*

a ä à á â ã å æ ą ă ā

Intuitively, rows are thought to be cyclic. If a row r has length n, then the successor of the i-th character ($i < n$) in r is defined as the $(i+1)$-th character of r, and the successor of the n-th character is defined as the first character of r. Analogously, the predecessor of the first character of r is defined as the n-th character of r.

A *replacement scheme* is a set of rows, such that no character occurs in more than one row.

Example 2 *An excerpt of a replacement scheme containing Latin letters used in European languages (MES-1, [1]):*

A Ä À Á Â Ã Å Æ Ą Ă Ā
a ä à á â ã å æ ą ă ā
C Ç Č Ć Ċ Ĉ
c ç č ć ċ ĉ
D Đ Ď Đ
d ð ď đ
E É È Ê Ë Ě Ę Ē Ė
e é è ê ë ě ę ē ė

Given a replacement scheme s and a character c, the successor $\mathrm{succ}(c, s)$ of c in s is defined as the successor of c in r, if c occurs in some row r of s, and as c itself otherwise. The predecessor of c in s is defined analogously, it is denoted by $\mathrm{pred}(c, s)$.

Typing [Accent] replaces the character c immediately before the cursor by $\mathrm{succ}(c, s)$, where s is the currently selected replacement scheme; typing [Ctrl-press] [Accent] [Ctrl-release] replaces c by $\mathrm{pred}(c, s)$.

Example 3 *Given the replacement scheme of Example 2,*

typing		*produces*	
	[D]		`D_`
	[e]		`De_`
	[Accent]		`Dé_`
	[j]		`Déj_`
	[a]		`Déja_`
	[Accent]		`Déjä_`
	[Accent]		`Déjà_`
	[–]		`Déjà-_`
	[v]		`Déjà-v_`
	[u]		`Déjà-vu_`

where _ symbolizes the cursor.

5.2 Implementation

Unfortunately, the SITMO technique does not have the prefix property. For this reason, a transparent implementation is not possible, and the input method has to be integrated into the application programs. Fortunately, this integration is straightforward: Suppose that the Backspace key is bound to the function `DeletePreviousChar()` and any printable character c is bound to `Insert(c)`, and that `GetCharBeforeCursor()` returns the character before the cursor, then the Accent key is bound to

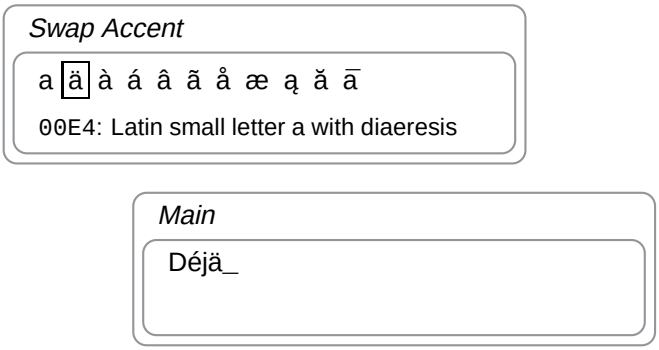
```
if (not CursorAtBeginningOfLine()) then
  c := GetCharBeforeCursor();
  DeletePreviousChar();
  Insert(succ(c,s));
endif
```

where `s` is the currently selected replacement scheme.

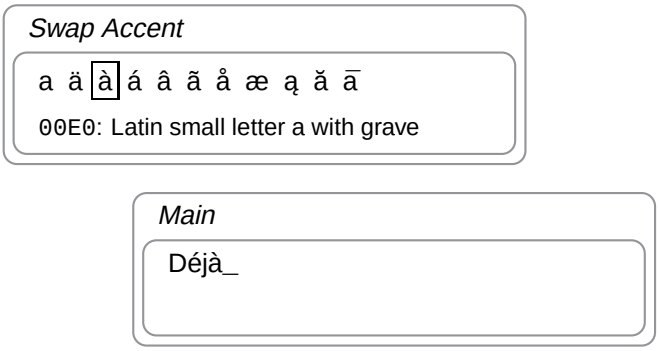
5.3 Improving the Feedback

Optionally, the functions "succ(c, s)" and "pred(c, s)" can be extended in such a way that, as a side effect, the list of characters in the current row (with the current character highlighted), the code number of the current character, and its name are displayed in a dedicated further window.

Example 4 *Typing* D e Accent j a Accent *in the window "Main" displays*



After typing Accent *once more, the display changes to*



In this way, the user can see in advance when the desired character will appear. In particular, to enter a character located at the end of the row one can either keep the Accent key pressed until the desired character is reached, or one can pick it from the "Swap Accent" window and paste it into the active window using the mouse. Besides, displaying the full name of the character eliminates ambiguity, especially, if the character repertoire contains characters that are (almost) visually indistinguishable. For instance, in certain fonts it may be difficult to tell "ĕ" ("e with caron") from "ĕ" ("e with breve"), and the upper case forms of the Icelandic letter "ð" and the Croatian letter "đ" share even the same glyph "Đ".

5.4 User Configuration

To be conveniently usable, it is essential that an input method is configurable and suitable for individualization. For the SITMO technique, this happens in two ways: First of all, every user can (or rather: must) pick a predefined replacement scheme for the language in which he wants to write. Starting from such a predefined replacement scheme, a more extensive configuration is possible by rearranging characters in the scheme according to personal needs. Using a drag-and-drop interface, it is relatively easy to ensure that no character occurs more than once in the replacement scheme. It should be noticed that by moving a character into a particular row, the remaining characters are shifted by one position, but they remain accessible using the standard rule: "type the base character and then type Accent sufficiently often." By contrast, when a key sequence in, say, the mnemonic compose technique is redefined, then the character that was previously accessed using that key sequence becomes inaccessible.

6 Evaluation

Like the trailing modifier method, the SITMO technique allows to enter accented characters in the "natural order" (first the letter, then the accent), and like the trailing modifier method, the price it has to pay for this is that it lacks the prefix property. The technique is obviously uniform, and the effort required to learn the SITMO principle seems to be negligible: our test subjects usually became comfortable with it within a few minutes. As for the numeric and the mnemonic compose method, a single additional key is sufficient. While being unlimited in theory, the scope of SITMO ranges in practice between mnemonic compose and modifier methods. Character repertoires where multiple accents on a single letter are common (such as Vietnamese or polytonic Greek) are an exception; in this case, modifier methods are superior.

There are two requirements for which it is not obvious how well SITMO satisfies them: Does it allow fast typing, that is, are the average key sequences short enough? And, are the key sequences sufficiently easy to learn? For rare derived letters foreign to the given language this is not a problem: in contrast to the mnemonic compose method, the user need not learn the exact key sequence by heart; it suffices to figure out the base letter and to type Accent until the desired derived letter appears. Key sequences for characters that are used in the standard orthography must be fairly regular, however, otherwise touch typing becomes impossible.

The number of letters that are derived from a single base letter varies enormously between different languages. In Vietnamese (Quốc Ngữ), the letter "o" has 17 derived letters (it can optionally carry a circumflex or a horn, plus optionally an acute, a grave, a hook, a tilde, or an underdot). In the old (polytonic) orthography of Greek, the letter "α" can

optionally carry one out of two breathings, plus optionally one out of three accents, plus optionally a iota subscriptum, yielding altogether 23 derived letters. Obviously, SITMO is not suited for such cases, and specialized modifier techniques [4] are a much better choice here. Fortunately, for many languages, these figures are much smaller. Most European languages, for instance, use orthographies in which the number of derived letters per base letter lies between one and three, rarely four. The question is now how to arrange these derived letters in a replacement scheme.

Let p_{ij} be the frequency with which the j-th letter in the i-th row of a replacement scheme s occurs in some fixed orthography.[5] Then the average number of keystrokes for derived letters is

$$k_s = \frac{\sum_i \sum_{2 \le j \le n_i} j p_{ij}}{\sum_i \sum_{2 \le j \le n_i} p_{ij}}$$

Provided the association from derived letters to base letters is fixed, then we can minimize this fraction by arranging the derived letters for any base letter by order of decreasing letter frequency (in the given orthography). We call this replacement scheme the *frequency-based replacement scheme* for the orthography.

Unfortunately, a small average number of keystrokes is not enough for our purposes. If we want the input method to be fast and memorizable, we need to find replacement schemes that yield a small average number of keystrokes, *and* in which the derived letters are arranged in a sufficiently mnemonic order.[6]

For which languages is this possible? To evaluate the method, we have inspected sample texts in 28 European languages.[7] Nine of these languages use at most one derived letter per base letter (e. g. in Latvian: ā, č, ē, ģ, ī, ķ, ļ, ņ, š, ū, ž), except in foreign words. For these languages, the frequency-based replacement scheme is obviously sufficiently mnemonic: the typist only has to remember the rule "to type an accented letter of the given language, type its base letter, followed by [Accent] *once*". The following table shows the percentage d of derived letters in the sample texts and the average number k_{freq} of keystrokes per derived letter.[8] As expected, for all nine languages k_{freq} is very close to 2, the difference being due to occasional foreign words and proper names in the sample texts.

Language	d	k_{freq}
Albanian	7.5	2.0000
Azeri	15.5	2.0001
Finnish	3.9	2.0005
German	1.6	2.0006
Irish Gaelic	5.7	2.0000
Latvian	9.0	2.0000
Maltese	3.8	2.0000
Slovene	2.5	2.0000
Turkish	9.3	2.0001

[5] Recall that rows start with a base character.

[6] As explained above, the order of rare derived letters *foreign to the given language* need *not* be mnemonic. We will leave out those derived letters in the following examples.

[7] That is, all European languages using some extension of the Latin alphabet $\{A, \ldots, Z, a, \ldots, z\}$, for which the author could find a 100,000 character sample of newspaper texts (in standard orthography) on the web.

[8] More precisely: per derived lowercase letter. The key sequence for an uppercase letter differs from the key sequence for the corresponding lowercase letter by one chorded [Shift] keystroke (except for the letter pairs İ, i and I, ı in Turkish and Azeri).

Icelandic is an example of a language where some base letters have two derived letters. It uses the consonants “ð” (eth) and “þ” (thorn), six vowels with an acute accent (á, é, í, ó, ú, ý), and two more derived vowels, namely “æ” and “ö”. The frequency-based replacement scheme is

a	á	æ
d	ð	
e	é	
i	í	
o	ó	ö
t	þ	
u	ú	
y	ý	

For this scheme, there is again a memorizable rule: “for the derived consonants, type Accent once, to put an acute accent over a vowel, type Accent once, to obtain *the other* derived vowels, type Accent twice”.

For Italian, Norwegian, and Spanish the situation is similar. In all these languages, the frequency-based replacement scheme is sufficiently mnemonic.[9]

Language	d	k_{freq}
Icelandic	10.7	2.1354
Italian	0.6	2.0717
Norwegian	2.1	2.0814
Spanish	2.0	2.0005

The frequency-based replacement scheme for Estonian is not quite as regular as for the languages considered so far:

a	ä	
o	õ	ö
s	š	
u	ü	
z	ž	

With this scheme, “ö” requires typing Accent twice, whereas the other doubly-dotted letters “ä” and “ü” require typing Accent only once. By swapping two characters in the “o”-row

a	ä	
o	ö	õ
s	š	
u	ü	
z	ž	

[9] All letter frequency tables and replacement schemes are available at `http://www.mpi-sb.mpg.de/~uwe/paper/AccInput-bibl.html`.

this can easily be fixed. The drawback is, however, that using the new scheme the average number of keystrokes increases from 2.1271 to 2.2867.

For Estonian both the frequency-based and the regularized replacement scheme seem to be acceptable. This holds likewise for Catalan and Polish. For Romanian, regularizing the rows leads to such a significant loss of efficiency, that working with a very slightly irregular replacement scheme appears to be a better choice. On the other hand, for Czech, Slovak, Dutch, and West Frisian, the frequency-based replacement schemes are so irregular that the advantages of regularizing clearly outweigh the increased number of keystrokes. The following table shows the average numbers of keystrokes k_{freq} and k_{reg} for the frequency-based and the regularized replacement schemes, respectively. An asterisk (*) marks non-recommended replacement schemes.

Language	d	k_{freq}	k_{reg}
Romanian	5.6	2.0833	2.4049*
Catalan	1.9	2.2312	2.3696
Estonian	3.1	2.1271	2.2867
Polish	5.6	2.0093	2.1228
Czech	9.7	2.0899*	2.1430
Dutch	0.1	2.3265*	2.3878
Slovak	8.4	2.0226*	2.0282
West Frisian	1.1	2.2213*	2.2342

In Croatian, Danish, Swedish, Lithuanian, and Hungarian, the frequency-based replacement schemes are either quite regular or can be regularized as above. However, both the resulting average numbers of keystrokes and the frequency of derived letters in these languages are comparatively high. In such a case, shifting a few derived letters from their default row to a different one may lead to a significantly faster input method. For instance in Swedish, by shifting the letter “å” from the “a”-row in the frequency-based replacement scheme

a ä å à
e é
o ö
u ü

to an arbitrary other row, say, the “s”-row, the average number of keystrokes is reduced from 2.3198 to 2.0005:

a ä à
s å
e é
o ö
u ü

We denote the average number of keystrokes for the shifted replacement scheme by k_{shift}

Of course, we have to pay for the increased speed by a slightly less mnemonic and uniform input method, but since the irregularity affects only a single frequently-used letter, this seems to be acceptable. Alternatively, this problem could be fixed by mixing SITMO

with the one-to-one technique, that is, by turning "å" into a base letter. (This would be particularly useful for Hungarian "ö" and "ü".)

Language	d	k_{freq}	k_{reg}	k_{shift}
Croatian	2.6	2.2226		2.0004
Danish	2.3	2.2674		2.0114
Hungarian	10.0	2.2739*	2.3089	2.0840
Lithuanian	6.1	2.0976*	2.3112	2.0700
Swedish	4.2	2.3198		2.0005

In French and Portuguese, the sets of accents that can be combined with a particular base letter vary so widely that it is almost impossible to arrange the derived letters in a mnemonical order, much less in a mnemonical and efficient order. To handle these two languages with SITMO, shifting one or two derived letters to alternative base letters or turning them into base letters[10] is inevitable.

Language	d	k_{freq}	k_{reg}	k_{shift}
French	2.9	2.2562*	2.4778*	2.1163
Portuguese	2.8	2.3428*		2.1456

Summarizing, we see that for those 28 European languages we have investigated, the average numbers of keystrokes per derived letter range between 2.0 and 2.4; for 25 languages they are below 2.2; for 20 languages even below 2.1. Concerning the average number of keystrokes, SITMO is thus comparable to the leading or the trailing modifier technique; if the modifier technique requires the use of the Shift key to type some of the modifiers, SITMO may even be superior.

One may ask to what extent our results are biased by the fact that we have used the same sample texts to generate the frequency-based replacement schemes and to test them. Would we have obtained substantially different results if the frequency-based replacement schemes had been generated using a different collection of sample texts? This is unlikely for the following reason: The order of the frequent characters is rather stable for sufficiently large sample texts. The selection of rare characters occurring in sample texts varies widely, but its influence on the average number of keystrokes is minute. For example, our 100,000 character sample text for German contained exactly two letters derived from "e", namely "é" (14 occurrences) and "ë" (1 occurrence).[11] There is no doubt that "é" is the most frequent letter derived from "e" in German and that it ought to be the second element of the "e"-row. However, there is no uncontested second most frequent letter, so "ë" might conceivably have turned out to be the fourth or sixth element of the "e"-row. But even under the unrealistic assumption that "ë" were the tenth element of the "e"-row, and even if there had been 10 occurrences of "ë" rather than a single one, the average number of keystrokes for the sample text would increase only from 2.0006 to 2.0501.

[10] A very regular replacement scheme for French can be obtained if "é" gets its own key on the keyboard. In this case, the average number of keystrokes for derived letters including "é" drops to 1.4979 (excluding "é": 2.3047).

[11] Occurring in a few French proper names.

7 Extensions

7.1 Beyond European Languages using the Latin Alphabet

The SITMO technique works of course also for non-European languages and for alphabets different from the Latin one – provided the character repertoire has about the same size and the number of derived letters per base letter is small. So syllabaries and ideographic writing systems must be excluded, just as Vietnamese, polytonic Greek, fully pointed Hebrew, or other orthographies where the number of diacritical marks that can be put on a letter is too large. Even four or five derived letters (e. g. in Yoruba: é, è, ẹ, ẹ́, ẹ̀) are already too much, unless some of these derived letters are extremely rare. Still this restriction leaves for instance monotonic Greek, Yiddish (using the Hebrew alphabet with a very limited number of points), and numerous East European and Asian languages that employ variants of the Cyrillic alphabet. Of course, for changing between different alphabets, some permanent switching mechanism is required: all character-by-character input methods discussed in this paper are too clumsy for typing, say, Greek phrases within an English text.

7.2 Beyond Letters

Until now, we have considered only letters – characters that are part of some alphabetical script. Apart from that, letter-like symbols, such as £, ¥, ©, ®, ™, or § can of course be included in the respective rows. Furthermore, many punctuation or mathematical characters have some "closest relative" in the ASCII character set, and could be accessed for instance using a replacement scheme like

!	¡												
?	¿												
<	«	≤	←	⇐	⊂	⊆	∈	⊏	⊑	≺	⋖	◁	⊴
>	»	≥	→	⇒	⊃	⊇	∋	⊐	⊒	≻	⋗	▷	⊵
*	×	⋆	⊗										
=	≠	≡	≐	⇔									
∼	≈	≃	≅										

On the other hand, SITMO is rather unsuited for characters for which there is no obvious mapping to base characters, for example line graphics or dingbats.

7.3 Beyond Typing

While we have considered only keyboard-based input methods so far, the SITMO technique is also applicable to pen-based input systems, such as Unistrokes [2] or Graffiti [5, 6]. Here we have to cope with the fact that the stroke patterns should be both simple and memorizable for the user and reliably distinguishable for the stroke recognition software. The number of such stroke patterns is again limited, however, so representing every possible diacritical mark by an individual stroke pattern becomes problematic. Using SITMO, any number of diacritical marks can be produced with a single stroke pattern (or a tip into a particular spot of the writing area) that takes the role of the Accent key.

8 Conclusions

The SITMO technique combines in a uniform way short key sequences for frequently used characters with an easily memorizable scheme to enter rarely used characters. As we have shown in this paper, for most European languages the frequency distributions of derived letters are sufficiently unbalanced that the resulting average numbers of keystrokes per derived letter are close to 2. With respect to the number of keystrokes, SITMO is thus comparable to leading or trailing modifier techniques. Its scope, however, is much larger. Furthermore, like the mnemonic compose technique, SITMO requires only a single additional key, whereas leading or trailing modifier techniques require usually one key per accent.

The main drawback of the SITMO technique is that it lacks the prefix property, so a transparent implementation is not possible. This is the price we have to pay for a more natural input technique. In this respect, SITMO shares the advantages and the disadvantages of trailing modifiers compared with leading modifiers.

References

[1] European Committee for Standardization. *Multilingual European Subsets in ISO/IEC 10646-1*. CEN Workshop Agreement CWA 13873:2000, March 1, 2000. Available at `http://www.evertype.com/standards/iso10646/pdf/cwa13873.pdf`.

[2] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Conference on Human Factors in Computing Systems, INTERCHI'93*, Amsterdam, The Netherlands, April 24–29, 1993, pp. 80–87. ACM.

[3] International Organization for Standardization. *Input methods to enter characters from the repertoire of ISO/IEC 10646 with a keyboard or other input devices*, ISO/IEC 14755:1997, 1997.

[4] Mark Leisher. Input metthod design. In *Pre-conference tutorials proceedings: Software development + the Internet: going global with Unicode: Ninth International Unicode Conference*, San Jose, CA, USA, September 4–6, 1996. The Unicode Consortium. Available at `ftp://crl.nmsu.edu/CLR/multiling/unicode/tutorial.ps.gz`.

[5] I. Scott MacKenzie and Shawn X. Zhang. The immediate usability of Graffiti. In Wayne A. Davis, Marilyn Mantei, and R. Victor Klassen, eds., *Graphics Interface '97*, Kelowna, B.C., Canada, May 21–23, 1997, pp. 129–137.

[6] Palm, Inc. Graffiti. `http://www.palm.com/products/input/`, 2000.