Answer Markup Algorithms for Southeast Asian Languages

presented at the 8th Annual International CALICO Symposium April 6, 1991

George M. Henry

Abstract: Hart, Nesbit and Nakayama, and others have described answer markup methods for providing feedback to short answers entered by foreign language learners. These methods are not directly applicable to the languages of Southeast Asia, which are not written in a strictly linear fashion. Instead, these languages contain written tones and vowel fragments which appear above and / or below the main line. Thus a given column may have several written letters making unmodified columnwise edit markup misleading. This paper describes the modification of Hart's edit markup software and a second variation based on a simple edit distance algorithm adapted to a general Southeast Asian font system devised by the author.

Keywords: edit markup, edit distance, Thai, answer feedback, Asian languages, Asian fonts

Introduction
Many techniques have been proposed for providing suitable feedback for responses to short answer questions typed by students taking Computer Assisted Language Learning (CALL) lessons. One method which has been used frequently in CALL courseware is to display a set of "edit markups.; that is, a set of symbols displayed near the student's answer indicating what areas of the answer are wrong, and the general operations needed to correct the answer. These operations typically are substitution, insertion, deletion, and transposition of characters. The markup symbols do not indicate the exact correction needed, nor do they explain the student's error in grammatical or semantic terms. The student is expected to consider the editing hints given by the markup symbols and produce a new (hopefully correct or more nearly correct) answer. (See Tenczar & Golden (1972) for details on early edit markup work with Tencore and PLATO, Hart (1981) for use with PLATO, and Pusack (1983), for a discussion of DASHER, an early microcomputer-based edit markup program.)
The advantages and disadvantages of this approach have been discussed at length elsewhere (Hart 1989; Pusack 1983). Briefly, the advantages are that 1) grammar-based feedback maybe inappropriate for certain pedagogical and learning styles; 2) lack of a grammar for a language is no problem; and 3) the method can be used with little or no modification for different languages.

Several recent edit markup schemes for roman alphabet answers have been described in detail by Nesbit and Nakayama (1990a, 1990b), Hart (1989), and others. These methods are based on evaluation of an answer by an "edit distance" algorithm, which compares the model with the answer and returns a string of edit markup symbols indicating a smallest set of edit operations required to correct the answer. (See Kruskal & Sankoff (1983) for the basic algorithm.)

However, while these methods are language independent in the sense that they do not depend on grammar or semantics, they are not directly suitable for languages such as Thai, Burmese, and other languages of Southeast Asia which use non-linear writing styles and non-roman scripts.

The goal of the work described in this paper was to modify existing edit markup schemes developed for linear roman alphabets, so that useful edit markup feedback can be incorporated in CALL courseware for Southeast Asian languages. This seemed a reasonable expectation (at least for algorithms based on edit distance determination) since both the model and the student's answer are a (temporal) sequence of keystrokes, whether the display is roman or a second language (L2) script, linear or not.

This work was motivated by the need to provide feedback for responses to short answer questions in Thai and Burmese in an audible CALL system running on MS-DOS computers developed by the author and his colleagues at Northern Illinois University. The L2 font definition and display software used was developed for this audible CALL system, but similar work could be done for other L2 software, assuming a one key one-character correspondence.

**Nature of Southeast Asian Orthography**
Several aspects of the orthography of Southeast Asian writing systems distinguish them from other writing systems, and thus define important aspects of the task of creating useful software for text entry, display, and edit markup of these languages.

These languages are basically phonetic, not ideographic. Thus they have a limited number of distinct character forms. Thai, for example, has about 100, including numeric
digits. As a practical consideration, this means that a representation of the character set can be held entirely in computer memory.

These languages are not written in a strictly linear, left to right manner. Rather, individual consonants (which are always written on the main level) may be surrounded by a "constellation" of vowel and tone fragments. These fragments may appear above, below, to the left, or to the right of the base consonant. Tones are always written above a consonant, possibly above a vowel which is directly above the consonant. Four distinct levels are used in Thai, for example. This arrangement is considered by some (e.g. Becker (1989)) to be a deep reflection of the basic cosmology and paradigms of Southeast Asian culture, not merely a stylistic device. Indeed, Becker reported that his Burmese teacher gently objected to his linguistic notes (in which Burmese was written in (linear) phonetic transcription), saying that he (Becker) was "hurting" the language.

Software capable of displaying these languages must be aware of the appropriate level for each symbol, and also must provide a "dead key" facility; that is, the cursor or carriage must not advance when tones and vowel fragments (which should be positioned over or under a consonant) are entered.

The individual characters of the languages are relatively complex compared to (printed) roman characters: Thai, for example, is rich in loops and tends to have more strokes per character than the roman alphabet. This visual complexity suggests that characters should be displayed on screen somewhat larger than normal roman characters, at least for beginning learners.

Many Southeast Asian languages do not consistently separate each word from the next with spaces. This fact has obvious consequences for edit markup schemes based on an analysis of individual words separated by delimiters such as spaces, commas, and so on.

**Schemes of Computer L2 Input**

There are two basic schemes by which L2 material may be entered into the computer. The first is to devise a romanization of the language precise enough for software to convert romanization to L2 characters unambiguously.

Romanizations have certain advantages. They can be roughly phonetic, and thus readable and pronounceable by non-experts in the language (librarians and new learners, for example). Romanized L2 material is well-suited to western computer hardware and software. Keyboards have the letters printed on them. Word processing software can be used to enter and edit romanized text, and other more specialized (but
still standard) software can process it further (for example, bibliographic data bases, concordance programs, and edit markup software).

However, there are serious disadvantages to romanization. For most Southeast Asian languages, no suitable romanization standard exists, nor does general agreement seem likely. Any sufficiently precise scheme seems fated to be awkward. For example, in Thai, there are several letters that make the "t" sound, therefore any romanization will either be ambiguous (if it uses "t" for each of them) or awkward (t + diacritic, e, t2, Alt-t, or whatever). Finally and perhaps the fatal objection to the use of romanization in learning a second language - is the fact that it is not, in fact, the language. It is an intermediate code which at best is a temporary expedient, and at worst, may be considered to "hurt" the language.

The second basic method for L2 input is simply to map L2 characters onto the keyboard without reference to the roman characters printed on the keys. Thus the varieties of "t" could be produced by 'W', "4", and "=" for example. Perhaps the most natural method is to model the computer keyboard after typewriter keyboards used in the L2 country. The normal keys (and their shifted modes) provide sufficient characters for the languages in question. Stickers, alternate key caps, a screen display, or printed diagram can be provided to show students the L2 keyboard.

This method avoids the disadvantages inherent in an intermediate romanization code, but has problems of its own. Providing suitable keyboards can be awkward, especially if a given machine is used for several languages (one cannot change key caps or stickers quickly). Custom software must intercept all keystrokes, convert each to the proper symbol, and display that symbol at the proper location on the screen. Thus existing software that assumes linear roman (i.e. standard ASCII) characters may not work. (Redefining text mode characters on MS-DOS machines could avoid some of these difficulties, but, in any case, the resulting characters are probably too small for beginning students. The Apple Macintosh, a graphics-based computer, has operating system features that avoid some of these problems.)

L2 Support for CALL
Given the above considerations, our audible CALL project designed software to support L2 text entry and display. This is the underlying system with which the L2 edit markup software described in the following sections will work. This software:

- supports multiple character sets via a graphics font creator/editor.
• features a typewriter-like interface. The creator of the character set defines key correspondences and alternate keyboard definitions are possible. Only shifted and unshifted keys are used; one keystroke per character.
• allows up to 127 characters per character set.
• supports "dead" keys, i.e. keys which do not advance the cursor/carriage.
• correctly positions tones and vowels above (or below) the baseline consonant.
• displays letters large enough for beginning students to see clearly (about 40 - 60 per line).

Adaptation of Hart's Method to Southeast Asian Languages
Robert S. Hart (Hart 1989) published and made available a complete edit markup software system with a number of enhancements and additions to answer markup resulting from a simple application of the edit distance algorithm. This software, in the form of a Turbo Pascal unit, served as the basis for the work described in this section. The approach was to use Hares software with as few modifications as possible; that is, to use it as a black box which takes (as input) the student’s answer and the exemplar model, and which then returns an edit markup string. This edit markup string is then used as the basis for the Southeast Asian L2 markup. Figure 1 shows two examples of a demonstration program of this method. The Thai displayed is not meaningful authentic Thai, but rather the results of typing several non-Thai words (which are displayed below the markup boxes). This is done so that non-readers of Thai can more easily understand the examples. (Note: in the interest of brevity "Thai" will be used to denote the more generic phrase "Southeast Asian L2.")

![Figure 1](image-url)
A brief description of Hares algorithm is in order. The basic edit distance algorithm compares the student’s answer and the model as two simple strings, and generates an answer markup accordingly. This can lead to nonintuitive markups, especially if words are omitted or out of order. Hart’s algorithm does the comparison on a word-for-word basis, attempting to match words in the answer with words in the model before the markup is generated for each word. Missing words and out of order words can be detected and edit markup generated accordingly, although the exact word order may not be explicitly indicated. Certain non-intuitive markups are detected and modified to appear more intuitive. The reader is referred to Hares technical report (Hart 1989) for details and examples.

In addition, Hart allows the lesson author to indicate (as part of the model) that certain words are "ignorable" (that is, they are discarded from the student’s answer wherever they occur). Hart also allows the author to specify sets of "synonyms," each of which can appear at any fixed place in a sentence. If any of a set of synonyms appears at the correct place in the sentence, it is counted as correct. Slightly misspelled synonyms will be recognized and edit markup generated for them.

A number of problems were encountered in adapting Hart’s software to Thai script which necessitated small changes to the code. However, these changes were made in a way that does not prevent the software from doing linear roman edit markup.

1. The delimiters for ignorable words and synonyms ([... ] for ignorable words and <…> for synonyms) may actually be used for valid Thai characters. A decision was made to use Alt-] etc. to enter these delimiters. They are displayed as [, ], <, and > on screen, through the use of software that detects the Alt key. Internally, these characters are kept as non-displayable control codes. Thus, the software that collects keystrokes for the model creates two versions: a graphics display version and a character suing version to pass to Hart’s markup routines. In order for Hart’s routines to process the input model string correctly, it was necessary for the calling routine to alter the set of delimiter characters and the test for delimiter characters in his code.

   The test itself was changed to check a new global boolean variable "roman - font" (default value true) to determine the appropriate set of delimiters.

2. Hart’s program has several ways to handle capitalization errors, none of which was appropriate for Thai. In Thai, a capital "@' and a lowercase "a" are not variants of the same letter, but are entirely different letters. Again, a minor change to Hart’s code was made in which a test was made to see if "roman-font" was true or false. If true,
the normal action was taken (set the character to its lower case version), if not the character was NOT lower-cased.

3. Hart's software places the edit markup determined directly on the screen. Since the placement and spacing of these symbols is inappropriate for Thai, a routine was added to save the edit markup in a string variable instead of directly displaying it.

These small changes were sufficient. It should be noted that the modified software is still completely suited for roman character markup. A single program could use the (modified) Hart software to do edit markup alternately in roman-based languages and Southeast Asian, simply by changing the delimiter set and the value of "roman-font."

For Southeast Asian languages, however, additional processing is required. The output from Hart's markup software (the sequence of edit symbols) is accessible to the calling program in a string variable. The next task is to use this information, together with knowledge of the displayed form of the Thai answer to correctly position the edit marks.

There are three basic problems involved here.

1. When a person types a Thai answer, there may be several possible correct keystroke sequences. The notion of viewing the model and answer as temporal sequences of keystrokes (for comparison purposes) breaks down here. For example, the sequence "consonant, tone, vowel" (i.e. levels 3, 1, 2 in a given column) should be judged the same as the sequence "consonant, vowel, tone" (i.e. levels 3, 2, 1 in a given column). The problem was solved by internally sorting all such sequences to a common order upon input (before either is passed to the edit markup routines).

2. As noted above, the nth character typed is generally not displayed in the nth column from the start of the string. An array of column locations was created to indicate the column location of each character, based on the level number of the Thai characters (only main level characters increment the column position).

3. Since several characters can appear in a given column, an edit mark at a column location will not indicate which character is in error. An array of level indicators was created to specify the level of each character. (Note: the level of a character is part of its definition.)
Given the information available as a result of all this, (the markup string, the Thai display, and the level and column location arrays), the Thai markup routines can display the markup symbols in the appropriate locations. Although all Thai display is done in a graphics mode, and thus any markup symbols could be displayed, the original symbols used by Hart in the text mode are used here as well, for consistency.

Testing of this software revealed several problems which are inherent in the edit markup software and which are not amenable to easy solution.

1. Since Hares algorithm works on individual words, it is necessary for both the model and answer to contain word delimiters (spaces, etc.). This does not seem unreasonable, especially for beginning students, but does not accurately reflect the written language. In the current version, if the model has spaces which the response does not, the markup returns a "run together" symbol or marks the word as "unrecognized". If the model doesn't have spaces where the response does, the software indicates either “unrecognized” or “missing letter(s) + unrecognized.” It would be possible to devise a word (or syllable) division algorithm to parse input and return a set of syllables (see Henry (1986) for a description of such a system for Thai romanization); however, such an algorithm would be language specific, and has not been attempted. Feedback based on edit markup applied to the whole answer would sometimes be superior, but would require giving up other features of Hares software that are based on a word-level analysis.

2. Word order problems are not always explicitly corrected, and sometimes are misleading, especially if many words are out of order.

3. Perhaps most seriously, short misspelled words are "unrecognized." That is, "bit" for "bat" would be marked as unrecognized. This is a consequence of the attempt to match words independent of their position. A cutoff value indicating the proportion of matching letters (2/3 is suggested) is used to determine potential matches. In the case of "bit" and 'bat”, since 1/3 of the letters don't match, the words are not regarded as matching. A change to the cutoff value could change this mismatch to a match, but at the expense of finding other false matches. There seems to be no way of reducing both types of error. Since beginning students will most likely often be typing short simple words, this problem seems quite serious.
Adaptation of Nesbit's Edit Distance Algorithm

Experiments with a simple edit distance algorithm (see Nesbit (1990)) revealed that it has no trouble with short word spelling errors (the markup of "bit" for "bat" is a single substitute operation for the "i"); although it entirely — and purposely — lacks some of the sophisticated features of Hart’s software. This section describes one particular elaboration using this core algorithm which was developed in an attempt to provide better feedback for the word order and short word spelling problems mentioned above.

This elaboration consists of using a brute force method for finding matches for words which are misplaced in the response. Each word in the model is placed in a array, and likewise, each word in the response is placed in a second array. Then corresponding words (i.e. both first words, both second words, etc.) are checked by the edit distance algorithm for a close match.

The degree of closeness of match can be set (as in Hares method); currently, the only allowed deviation from an exact match is a single transposition of characters. The "wrong" words are noted (a word may be "wrong" because of spelling or position), and then every possible whole answer made up of positional permutations of the 'wrong" words is generated and tested by the edit distance algorithm.

The permutation with the lowest edit distance cost is then used as the basis of the edit markup. The basic idea here is to correct any word order errors before markup, and then to display a correctly reordered (and edit marked) version of the student’s response.

Clearly this procedure is computationally expensive. The number of different edit distances that must be calculated is n!, where n is the number of "wrong' words. On a MS-DOS 80386-33 computer, processing five "wrong" words requires 120 different permutations and takes about 5 seconds. Processing four wrong words takes but a second or two. If we assume that for beginning students most responses will be short and/or that there will be relatively few "wrong" words, the response time will not be unreasonable. In the worst case, if there are too many "wrong" words for a given application, this fact is known almost immediately, and a message to the student can be displayed: "wait a minute: this will take some time", or even "there's a lot wrong here: do you want to try again?" Longer answers, even with a low density of errors would not be appropriate for this approach because of the delays involved in processing six or more "wrong" words on current microcomputers. Even faster microcomputers will not provide sufficient processing power to eliminate this problem, since the number of permutations increases as n!. (See Nesbit (1990a) for an approach to longer multi-word responses.)
One other modification proved necessary. The edit markup string returned by the algorithm did not always indicate the proper column for each correction even for roman markup (it was meant to be read from right to left, applying one change at a time, and as such, was somewhat non-intuitive). A subroutine was written to modify this string so that appropriate marks were written at the proper column for roman input. These changes included compressing multiple “I” (insertion) symbols into a single “^” mark, and displaying “> <” for the single character “T” (transposition).

Finally, the same routines used for the Hart version were used to position the symbols of the roman edit markup string in locations appropriate for Thai. One additional feature was added: since the algorithm "knows" which words have been moved (and were spelled correctly), those words - corresponding to word order errors - are redisplayed as green. Words that were spelled wrong (and possibly had been moved) are displayed as red, and words that were "almost right" (i.e. transposed characters) are displayed in cyan. Correct words in the correct position are displayed in white. The student's original response together with a (possibly) reordered and marked version are displayed for the student's inspection. Figure 2 displays two examples of this version. The feedback box shows the student's original answer and the (possibly) reordered version with edit markup below it. In the top example, the first Thai word is white (correct), the second is red (wrong) the third is cyan (almost right) and the last two are green (word order error, but words are correct), in the example above, both words are in red (wrong). Note that the "S" markup symbol is at level 1, indicating that the tone is wrong, not the main line consonant in the same column.
This feedback gives the student more explicit information about word order errors and generally will also give better feedback information about spelling errors for short words (i.e. "bit" for "bat"). However, it should be noted clearly that this version does not include some of the more sophisticated features of the Hart algorithms.

**Conclusion**
Two different edit markup algorithms were used to produce edit markup strings for a model and response considered as sequence of keystrokes. These markup strings were then used by software which positions the edit markup symbols correctly for a response displayed in a Southeast Asian language orthography. Probably no single scheme of edit markup determination is ideal for all applications; however, if an edit markup string can be determined on a one character per keystroke basis, this string can be used by the software described here for proper row and column markup symbol display.

**Future Research**
Two major areas of inquiry are suggested by this work:

1. Which of the various L2 feedback schemes is most helpful to students of Southeast Asian languages? Does the best method of feedback vary with question type or complexity, or with learner cognitive style? Are the answers to these questions different for linear roman alphabet languages?

2. Nothing in this paper has suggested when or how L2 script should be taught and learned. This is a major question on which very little research has yet been conducted due to the small number of students studying these languages.

When these algorithms are implemented in the audible CALL system mentioned in the introduction to this paper, it will be possible to investigate these questions.

**Availability**
Both versions of the L2 edit markup and display software, as well as the L2 font generation and display routines are written as Turbo Pascal units, callable from a main program. The edit markup algorithms are callable with identical syntax except for the names of the markup routines. The modified version of Hares unit is also available. The L2 software runs in EGA graphics mode on MS-DOS computers.
REFERENCES


AUTHOR'S BIODATA
George Henry, M.S. Computer Science, Ed.D, Instructional Technology; Assistant Professor of Computer Science at Northern Illinois University, has developed FLIS, and audible CALL system for Southeast Asian (and other) languages.

AUTHOR'S ADDRESS
George M. Henry
Computer Science Department
Northern Illinois University
(815) 753-6496