Kanji to Hiragana Conversion Based on a Length-Constrained *N*-Gram Analysis

by,
Joe Picone, Tom Staples, Kazz Kondo, and Nozomi Arai
Tsukuba Research and Development Center
Texas Instruments
Tsukuba, Japan

Abstract

A common problem in speech processing is the conversion of the written form of a language to a set of phonetic symbols representing the pronunciation. In this paper we focus on an aspect of this problem specific to the Japanese language. Written Japanese consists of a mixture of three types of symbols: kanji, hiragana and katakana. We describe an algorithm for converting conventional Japanese orthography to a hiragana-like symbol set that closely approximates the most common pronunciation of the text. The algorithm is based on two hypotheses: (1) the correct reading of a kanji character can be determined by examining a small number of adjacent characters; (2) the number of such combinations required in a dictionary is manageable.

The algorithm described here converts the input text by selecting the most probable sequence of orthographic units (*n*-grams) that can be concatenated to form the input text. In closed-set testing, the *n*-gram algorithm was shown to provide better performance than several public domain algorithms, achieving a sentence error rate of 3% on a wide range of text material. Though the focus of this paper is written Japanese, the pattern matching algorithm described here has applications to similar problems in other languages.

EDICS: (1) SA 1.3.1; (2) SA 1.9

Please direct all correspondence to: Dr. Joseph Picone, Institute for Signal and Information Processing, Mississippi State University, Box 9571, Mississippi State, MS 39762, Tel: (601) 325-3149, Fax: (601) 325-2298, email: picone@isip.msstate.edu.

I. INTRODUCTION

A common problem in speech processing is the conversion of a written language to a set of phonetic symbols. Such algorithms are often called letter to sound rules in English, and are commonly a core component of a text to speech synthesis system [1]. More recently, as interest in large vocabulary speech recognition has grown, and speech database projects have become more ambitious, such algorithms have found applications in the development of phonetically balanced sentence sets [2,3] and in the evaluation of speech recognizers [4]. The algorithm described in this paper, in fact, was used to develop a database of 10,000 phonetically balanced sentences for a Japanese speech database project [5].

I.1 An Overview of Japanese Orthography

The Japanese writing system is logographic: each character in the writing system, referred to as a grapheme, can denote all or part of a word. The conventional writing system [6,7] consists of a mixture of three types of symbols[1]: kanji, hiragana, and katakana. Kanji symbols, adapted from the Chinese writing system, are used to represent many conceptual words and indigenous names. Hiragana symbols are used to write inflectional endings of the conceptual words written in kanji, and are used to write many types of native words not written in kanji. Katakana symbols are used to represent words of foreign origin. Hiragana and katakana are syllabaries [6].

A typical example of Japanese orthography is shown in Figure 1. Words, such as “watashi” or “tōkyō”, can be composed of several kanji graphemes. Verbs, such as “ikimashita” (to go) shown in Figure 1, contain a mixture of kanji characters indicating the base form (“iku”) or verb stem and hiragana characters representing the verb conjugation (past tense). Place names of foreign cities (and other foreign words), such as “nyū yōku” (“New York”) are written in katakana.

1. Roman letters (e.g, company names such as “NHK”) and arabic numerals (e.g., digit string such as “3,300¥”) are also used in conventional orthography. However, these character types present a problem of secondary importance, and one whose solution is not necessarily unique to written Japanese.

In Japanese orthography, the words within a sentence are not normally delimited with a space or some other marker to indicate a word or phrase boundary. Only sentence boundaries are delimited. Conceptually, we must segment the text prior to converting each group of graphemes to their meaning or pronunciation. In Figure 1, the correct segmentation of the text is also given. Automatic segmentation is one challenge in machine processing of Japanese text.

In most formal writing (e.g., newspapers, magazines, business letters, and electronic media) it is recommended that use of kanji characters be restricted to a national standard set referred to as Jōyō kanji ("Everyday Kanji"). This set contains 1,945 characters [6] and can be regarded as the core kanji set that an average reader is expected to read easily. To put the magnitude of the problem somewhat in perspective, the average college undergraduate can read about 3,000 characters, and a good dictionary contains about 12,000 characters [8]. In this paper, we will be most concerned with the accuracy of algorithms on text composed only of characters within the Jōyō set. Characters outside of this set most often must be handled as special cases.

On the other hand, two more extensive character sets have been introduced as an electronic representation for modern day computing environments. These representations serve a function similar to the ASCII character set — they provide a means to store text in a machine independent format. They are both supersets of the Jōyō list and define conventions for representing kanji characters outside of the Jōyō kanji list.

The first of these two sets is known as the 8-bit Japan Industry Standard (Shift-JIS) [9] character set. Shift-JIS is found on a wide variety of computing platforms and electronic media. For example, shift-JIS is supported within the Japanese language extensions to the popular X11 Window System [10] (available on most UNIX workstations). The second set tends to be used mainly in the Japanese Language Environment for Sun workstations (and associated third party

tools), and is known as the Extended Unix Code (EUC) [11] character set. Its main feature is that it is a multi-byte character set (one, two, and three byte sequences are currently supported), and supports both conventional ASCII text and Japanese orthography within the same character set.

Both of these character sets contain about 7,000 characters. It is easy to encounter text sources, such as electronic books and encyclopedias, that make extensive use of the non-Jōyō characters in these character sets. In electronic information processing today, there is significant pressure to restrict usage of kanji to characters within these sets. Hence, these character sets play an important role in the development of the text processing algorithm described in this paper.

I.2 A Simple Pattern Matching Approach

One of the reasons hiragana is an extremely useful representation of the written language is that its character set is limited to about 125 characters. Hiragana does not map directly to pronunciation —some amount of additional processing is required. However, hiragana is a representation that is quite close to the actual pronunciation, and therefore is very useful for measuring the complexity of the underlying sounds in a given segment of text.

One distinguishing aspect of Japanese is that all written text can be converted to hiragana with no loss in information about the pronunciation, though it is possible that there might be some loss in understanding. An example of such a conversion is shown in Figure 1. Another rather distinguishing aspect of Japanese orthography [6,12] is that the correct translation of a kanji character depends on the context in which it appears. Each kanji character normally has several common readings and must be disambiguated by examining the context in which the character occurs. In other logographic systems, such as Chinese, more kanji-like characters are used, but the translations of each character are less context sensitive, and a context-independent dictionary lookup procedure appears to suffice.

In this paper, we describe an algorithm for converting conventional Japanese orthography to a hiragana-like symbol set that closely approximates the most common pronunciation of the text. The algorithm is based on two hypotheses: (1) the correct reading of a kanji character can be determined by examining a small number of adjacent characters; (2) the number of such combinations required in a dictionary is manageable.

This algorithm selects an optimal translation of the text by constructing the most probable sequence of graphemes that can be concatenated to form the input text. Comprehensive dictionaries have been developed that contain kanji *n*-grams (mixtures of kanji and hiragana are also used) and their hiragana-like translations (implemented using ASCII codes). The accuracy of the algorithm is extremely dependent on the quality of these *n*-gram dictionaries (discussed in Section II).

Our general philosophy in this work is to favor algorithmic simplicity over CPU or memory efficiency. This is based on our observation that computer memory and CPU speed are increasing at a rapid pace. On-line dictionaries once thought to be too large to be practical can now be routinely stored in physical memory during processing. Our approach is inherently dictionary-based and memory intensive, rather than rule-based. A major design goal was to develop a system that can be easily improved by a novice. Modifications of rule-based systems often require extensive forethought and expertise in natural language processing, and historically have been difficult to maintain.

Several such systems already exist in the public domain. In Section IV, we compare performance to three public domain algorithms: JUMAN [13], WNN [14], and KAKASI [15]. Each of these algorithms contains a mixture of rules and exception dictionaries (not well-documented in the accompanying literature). Each system is fairly limited in the amount of

Japanese text that can be accurately processed. In each case, extension of the system is nontrivial[2]. Hence, we perceived a need to develop a new and simpler system based on statistical methods that would handle a diverse range of Japanese text.

In the next section, we give an overview of the dictionary design and summarize the current state of the dictionaries. Next, we discuss the basic parsing algorithm used to locate the best combination of patterns in the dictionary. Of course, the dictionary design and parsing algorithm are highly interrelated. In the last part of this paper, we present the results of some comparative evaluations on two text databases, and discuss some of the limitations of the system.

II. *N*-GRAM DICTIONARY OVERVIEW

We have decomposed the conversion problem into two steps: dictionary design and sentence parsing. In this section, we discuss the problem of dictionary design. Two major constraints on the dictionary design were that a dictionary must be simple to construct and a dictionary must be extremely easy to augment. Conceptually, the system can be viewed as using one large dictionary. In practice, we find it much easier to maintain a separate dictionary for each *n*-gram order (kanji character sequence length). The system described in this paper currently uses nine dictionaries containing *n*-grams of length one to nine.

An entry in the dictionary contains three essential fields: the kanji sequence, its associated hiragana-like translation, and a weight or probability of occurrence. We will refer to the hiragana-like translation as its reading in order to maintain consistency with the literature[3]. Currently we use ASCII representations for these symbols. Our present symbol set was derived from an ASCII hiragana symbol table suggested by the Japan Electronic Industry Development Association

2. KAKASI, the latter system, appears to be the easiest to extend, though it is not completely clear from the documentation what the interactions are between the dictionary entries and the parser.

3. Clearly, in many cases (such as those described in the next paragraph) the dictionary entries reflect pronunciation. However, the term “reading” is adopted because it is more commonly found in textbooks and dictionaries dealing with the writing system, and conversion from kanji to pronunciation involves, at least conceptually, first determining the correct reading of a character.

(JEIDA) [16]. The entire symbol set is shown in Figure 2.

We have faithfully followed the JEIDA standard except for one significant modification. We have extended the representation to explicitly model long vowels — syllables ending in long vowels are marked with a "@" symbol. For example, "ōkii" (the Japanese adjective for "big") is translated to "o@ ki@." The major motivation for this change was based on some pronunciation considerations.

Vowel duration is important in spoken Japanese. There are at least three types of vowel durations. There are short vowels, such as "o" in "hon". These are mapped to the appropriate symbols shown in Figure 2. Long vowels, such as "ō" in "ōkii" are output as the vowel followed by an "@" (e.g., "o@"). Sometimes two successive vowels in a word internal position will be pronounced as a single long vowel. In this case, the "@" symbol is also used. Hence, the "ii" in "okii" is output as "i@" since it is normally pronounced as "ī."

Finally, there are situations where two consecutive vowels will be pronounced[4] as two separate vowels. In this case, two vowels will be output (i.e., two short vowels such as "i i" or a long vowel/short vowel combination such as "o@ o"). Multiple consecutive vowels tend to occur across word boundaries or in long vowel/short vowel contexts within words. We have used an explicit representation of the various vowel contexts anticipating that differentiation of these contexts might be useful in subsequent speech recognition research.

The general pattern used to describe each entry in the dictionary is:

$$[k^{m}]k^{n}[k^{l}] \rightarrow h^{p} \ , \tag{1}$$

where the *n*-gram order, N, is defined as:

$$N = m + n + l. \tag{2}$$

4. Of course, there are no absolutes in such situations. The system will output consecutive vowels only if we are confident they will be pronounced as separate vowels. Our bias is to output long vowel symbols when in doubt.

k represents a kanji character, m and l represent the number of characters preceding and following the context of interest, and n represents the number of characters within the context. h^p represents the output sequence of hiragana characters defined by this n-gram entry.

Left and right context are optional. This is indicated in Eq. (2) by the square brackets surrounding k^m and k^l. Of course, it would be convenient if only context independent entries were required. For a significant percentage of entries (e.g., proper nouns), this is clearly the case. The largest percentage of entries, however, use right context to define the proper reading. Right context is most often an adjacent set of kanji characters, and commonly a mixture of kanji and hiragana for higher order n-grams. Occasionally left context is used, mainly in defining hiragana to kanji transitions. Such transitions are often useful for characters that might have a few common readings that are strongly correlated with their function in the sentence. Sometimes, this can be isolated by examining neighboring hiragana characters.

Though Eq. (1) appears to make a dictionary entry look complicated, each entry is in fact very simple. Samples from the dictionaries for n-grams of order one, two, and three are shown in Figure 3. Each entry consists of an n-gram coded using an EUC representation, an ASCII mapping, and a weight.

Initially, we expected the weight for a dictionary entry to be probabilistic. We planned to compute these probabilities by analyzing the statistics of a large text database [5]. We were anticipating the use of probabilities to resolve ambiguities and to “learn” the most common interpretation of a character sequence. More complex statistical models along these lines have been tried elsewhere [17,18], but it is unclear as to the impact that continuous-valued probabilities have played in the system accuracy (clearly binary valued weights have some value).

To our surprise, we found that several simple rules for assigning weights to dictionary entries

were sufficient. First, the nominal weight associated with an *n*-gram is linearly proportional to its length. In our case, each character has a nominal weight equal to 1.0. Second, when ambiguities occur, *n*-grams of higher order are preferred over combinations of *n*-grams of lower orders. For example, from Figure 3, the nominal weight for a 3-gram is slightly higher than any permissible combination of 1-grams and 2-grams.

Third, *n*-grams containing all kanji entries are preferred over combinations of hiragana and kanji. Sometimes, a sequence of a hiragana character followed by two kanji characters can be interpreted in multiple ways — a 1-gram followed by a 2-gram or vice-versa. Adjusting the weights of the *n*-grams containing all kanji characters has proven to be an effective way to make sure the sequence is segmented properly.

Observe that in Figure 3, the last four entries in the 1-gram set are tagged as "JYO" or "EUC.", and the weights of these entries are less than the nominal weight of a single character (1.0). This was introduced as a diagnostic measure. Some characters should never be translated as a 1-gram because their reading is highly ambiguous. Characters in this category that fall within the Joyo set are tagged with the symbol "JYO". Characters that fall outside the Joyo set are tagged with the symbol "EUC." Because the weights of these entries are set to be lower than all other 1-gram entries, these entries will only be used when there are no other choices (the character essentially falls through the cracks in the system). This is an important diagnostic tool for determining cases for which the dictionaries need to be improved.

Finally, and most importantly, we need to discuss the rationale for adding a dictionary entry. The *n*-gram dictionaries are explicitly one-to-one mappings: each *n*-gram has only one entry in a dictionary. Usually, a character that has a dominant reading is entered into the 1-gram dictionary. Unfortunately, this is not the case for most characters — common characters will typically have

two to four highly probable alternatives [6].

Our strategy is to define as many of the contexts for which the reading deviates from some default reading in the higher order dictionaries, and enter to the default reading in the 1-gram dictionary. An example of this is shown in Figure 4. The character shown in the 1-gram entry is normally read as “tsubasa” (which means “wing”). However, when it appears as a 2-gram, the same character is read as “yoku” (which also means “wing” but is used in contexts such as “left wing” and “right wing”)[5]. In this case, about 30 entries are required in the 2-gram dictionary to identify all common exceptions to the default reading.

This is, in effect, the normal procedure for adding a new entry to the dictionary: (1) identify the default behavior and add it to the low-order dictionaries; (2) identify the exceptions and add them to the higher order dictionaries. This procedure is easily accomplished by consulting a good Japanese kanji dictionary [13,19]. We have been successful at training novices to perform such dictionary maintenance — all that is required is an above average kanji reading level (we have used secretaries with two years of post-college education) and a good kanji dictionary. Hence, we believe that we have achieved our first requirement: simple dictionary maintenance.

A summary of the size of each *n*-gram dictionary is given in Table 1. The total size of the dictionaries is 145,753 entries. Not unexpectedly, the distribution of entries peaks with the 2-gram dictionary, and tails off quickly. The entries in these dictionaries initially came from several publicly available electronic dictionaries [20,21], and were subsequently manually corrected based on some experimental results (see Section IV).

One might wonder how many of these entries are actually needed. This is hard to determine. The tails of the distribution in a text database tend to be large. Unless one processes vast amounts

5. This difference is referred to in Japanese as “on-yomi” and “kun-yomi.”

of data from numerous diverse sources, one cannot be sure of the effectiveness of a given entry. We conducted a simple experiment in which we processed a text database of 900,000 sentences chosen from a wide range of text sources [5]. An analysis of the use of each dictionary entry showed that over 40% of the entries were never used. Informal reviews of the current dictionaries have verified this result. Many of the dictionary entries are superfluous. For example, many 4-gram and 5-gram entries are not required because the same contexts are covered in the 2-gram and 3-gram dictionaries.

We are in the process of manually reviewing the dictionaries and consolidating duplicate entries. A topic of further research will be to optimize the dictionaries algorithmically based on grammar compiler techniques. Even without optimization, however, the current dictionaries occupy about 15 Mbytes of computer memory — not much given the power of modern desktop computers. If the dictionaries are not loaded into memory, CPU requirements would be even more modest by today's computing standards.

Search time in the dictionaries is also not a significant issue. Currently, the dictionaries are searched using a bisection search algorithm that has a complexity of $O(\log N)$. With this algorithm, even if we double the size of the dictionaries, the incremental cost in CPU time is insignificant. Hence, aside from memory concerns, extremely large dictionaries are not a problem. We are fairly confident that even if we expand the coverage of the current system, it is unlikely that the total size of the dictionaries would even double over their present size.

III. LENGTH-CONSTRAINED DYNAMIC PROGRAMMING

In this section we discuss the problem of efficiently parsing the text for the best combination of dictionary entries. As mentioned before, the search algorithm and the dictionary design are closely coupled. The weights of the dictionary entries strongly influence the choice of the best

path in the search algorithm. Because the number of permissible combinations could potentially be large, and the dictionaries are very large (by design), an efficient search algorithm is very important.

Perhaps the simplest approach to a dictionary-based algorithm is to scan the text from left to right and select the largest *n*-gram found in the dictionary. This strategy, which we refer to as the "largest *n*-gram first" approach, is summarized below:

```
for 1 ≤ i ≤ M  {
    for 1 ≤ j ≤ N  {
        if ∃k_l:k_{l+j-1}  {
            k_l:k_{l+j-1} → h^p;        (3)
            i = i + j
            break;
        }
    }
}
```

where M is the length of the input string, k_l denotes the l^{th} character in the input, and N denotes the maximum *n*-gram order. Though at first glance it might appear shorter entries would be found first using this strategy, the weights in the dictionary are adjusted[6] such that on the average longer matches are favored over shorter matches.

For example, starting at the first character, search the 9-gram dictionary for the first nine characters. If no match is found, search the 8-gram dictionary for the first eight characters. When a match is found for an *n*-gram order of i, convert the first i graphemes, and move to the $(i + 1)^{st}$ grapheme. Since most Jōyō kanji graphemes have a 1-gram entry, usually at the very least, a reading consisting of all 1-grams will be output.

This strategy is worth mentioning because it is quite effective — it is valid for perhaps 80% of

6. These adjustments are largely heuristic in nature, but were applied using semi-automated techniques.

the ambiguity typically encountered. However, one quickly learns that this strategy is not sufficient for high performance. Often, fixing a choice early will result in improper translations later in the sentence. Such an example is shown in Figure 5. In this case, choice of an initial 2-gram results in a nonsense parse for the next three characters. In this case, there is a possibility that the succeeding three characters could be treated as a 3-gram or three 1-grams, but in both cases the readings are incorrect. Clearly what is needed is a global optimization: translate the "most-probably" or "best" sequence in the entire text first, and then work our way backwards and forwards to complete all unresolved *n*-grams. Since it is likely that there will be competition amongst alternatives, it is best to cast this problem as an optimization problem, and solve it using a fairly standard dynamic programming (DP) approach.

Since the number of entries in the dictionaries is by necessity large, we would like to limit the number of times the dictionary is searched. With this in mind, we will pose this optimization problem using an approach in which incremental scores are accumulated primarily on both the nodes [22] (a "Type N" case). Our node score is defined as:

$$d_{node}(i, j) = w[k_{i-j+1}:k_i] , \tag{4}$$

where $w[k_{i-j+1}:k_i]$ represents the weight of an *n*-gram dictionary entry of length j corresponding to characters k_{i-j+1} to k_i in the input text. The best path will be chosen as the path with the maximum total score.

The transition score is very important in limiting the search space, and is defined as:

$$d_{trans}((i, j)|(i-k, l)) = \begin{matrix} 1 & j = k, \\ 0 & elsewhere, \end{matrix} \tag{5}$$

where d_{trans} indicates the score due to making a transition from one node to the next. Let us denote the maximum *n*-gram order as N_{max}. Eq. (5) indicates that we need to search one and only

one previous column for each node, in the range: $1 \le l \le N_{max}$.

Before analyzing the implications of Eqs. (4) and (5) too deeply, let us illustrate the solution using a dynamic time-warping scenario. The formulation is similar to that used in other text processing problems [23] and speech recognition problems [24]. We will assign kanji characters to the horizontal axis and *n*-gram order to the vertical axis. The dynamic programming-based search for the example of Figure 5 is shown in Figure 6. We call our approach length-constrained dynamic programming because it is obvious from Figure 6 that the best path is simply an arrangement of acceptable *n*-gram orders under the constraint that the sum of their lengths equal the length of the input text.

At node (i, j) the node score is computed by searching the appropriate dictionary for an *n*-gram composed of kanji characters k_{i-j+1} to k_i. Since the dictionaries are a one-to-one mapping, there can only be one choice possible at each node. Even if multiple choices were allowed, the most probable (or entry with the largest weight) would be chosen at each node in the optimization process — there is no need to keep alternatives in finding the best path. However, for other applications, retaining multiple readings of an *n*-gram at each node might be desirable (for example, all possible readings of the text could be output).

Of course, the feature of DP is that the number of previous alternatives that need to be explored are kept to a minimum. In our case, the number of previous partial paths that need to be searched are a function of the *n*-gram order j. Only column $i - j$ needs to be searched for a previous partial path backpointer, because the *n*-gram at the current node will consume all characters from the end of the previous partial path from column $i - j$ to the current node. In this sense, the *n*-grams are designed to consume input characters backwards in position — they consume characters in the direction towards the beginning of the text. This is depicted by the

curved arcs in Figure 6.

Fortunately, the number of competing hypotheses is often small, so the search time is actually very close to linear with the length of the input text. On an evaluation database described in the next section, 19% of the sentences contained at least one ambiguous character sequence. On a character by character basis, about 5% of the characters (columns in the DP grid) display multiple partial paths. Hence, if one desires truly high performance, a DP matching algorithm must be employed. On the other hand, if an algorithm needs only about 80% accuracy (the performance achieved by many public domain algorithms), the need for DP matching will not be apparent.

Finally, the output from the *n*-gram algorithm is given in Figure 7 for the example in Figures 5 and 6. The ASCII reading is output along with some debugging information. Normally, all punctuation is stripped from the input and not displayed in the output. If a symbol such as "JYO" or "EUC" is displayed in the output, then we know something went wrong. A DP debugging display is available that shows the partial paths, competing dictionary entries, etc. These are extremely useful in debugging the algorithm and determining the required improvements in the dictionary.

IV. EVALUATIONS

As we mentioned in the introduction, kanji coverage is a crucial issue in evaluating the performance of a system. As part of an ongoing project to collect a large database of spoken Japanese, we have generated a large sentence database from a diverse set of text material. This database [5], known as the Electronic Book (EB) database, includes sentences extracted from almost 1 Gbyte of data and includes 19 different text sources. Among them are several standard Japanese dictionaries, Encyclopedia Britannica in Japanese, a leading newspaper, and some Japanese literature. A summary of the text database is given in Table 2.

The algorithm was trained extensively on the first five text sources in Table 2. The last two columns of Table 2 show the number of sentences that, with the current dictionaries, still contain unidentifiable character sequences. Due to limited resources and time, we were not able to manually correct errors for all text sources. The first five sources were thought to be most useful for general text processing. The remaining sources require mainly proper noun additions to the dictionaries.

We conducted two formal evaluations of the *n*-gram algorithm. First, we developed a set of 1,000 sentences by randomly sampling all text database sources shown in Table 2 except the FJ News database. Second, we developed a sentence set of 1,000 sentences by randomly sampling the FJ News database. Both of these sets were extensively reviewed such that they contain sentences which had unambiguous readings and did not contain any pathological problems for machine conversion (more on this in the next section). We evaluated several public domain algorithms on these databases and compared their performance to the *n*-gram algorithm. A summary of the results for each database is given in Tables 3 and 4 respectively.

Three public domain algorithms were chosen for inclusion in this test based on their availability, widespread use within the community, and adoption as somewhat of a *de facto* standard for certain key applications. The first of these is a system denoted as JUMAN (Japanese University Morphemic ANalysis) [13]. Kanji to hiragana conversion is a small part of this extensive package that supports general Japanese language text processing and dictionary access. This package is widely used within the Japanese research community, mainly because it supports on-line interaction with some standard CD-ROM dictionaries included in the text database of Table 2.

The second public domain algorithm evaluated was Wnn [9]. Wnn is part of many Japanese

wordprocessors available under Unix. It is used to handle input and display of kanji text for the Japanese language extensions to the X11 Window System [10] and for the Japanese language version of emacs (a popular WYSIWYG text editor). Several popular commercial wordprocessing packages also use the Wnn software.

The third algorithm chosen was KAKASI [15]. We discovered this algorithm after working extensively with the first two algorithms previously described (and after noting their deficiencies). KAKASI is more focused on the problems of text conversion. Though all three of these algorithms use some level of dictionary lookup, KAKASI's dictionaries are clearly the most extensive. In fact, they appear to be a derivative of the Kojien dictionary [21]. KAKASI is also the computationally most efficient of these algorithms, and appears to be the most extensible since its dictionary format is fairly well-documented and simple. One of the major drawbacks, however, is that its pattern matching approach appears to be simplistic and unable to exploit sophisticated dictionaries.

In this evaluation, the output from each algorithm was hand-scored by an expert, and the resulting errors were tabulated in two classes: (1) a rejected sentence — the system did not output a valid translation; (2) a substitution error — the system incorrectly converted one or more kanji characters. The composite performance of the system is the sum of these two types of errors. In addition, we also kept track of the total number of kanji character errors generated for each evaluation. The most commonly misread characters are shown in Figure 8. Because the process of hand-scoring is time consuming, we were limited to an evaluation database size of approximately 1,000 sentences.

It is not surprising that the results for the *n*-gram algorithm are so promising. The results presented in Table 3 are the result of extensive training of the algorithm on the EB database. In

this sense, it is a closed-set test and we would expect superior performance under such conditions. In fact, we expect that the performance of each of the public domain algorithms could be significantly improved if more attention was given to their dictionaries. In Tables 3 and 4 we also see that KAKASI is significantly better than the other two algorithms. The performance levels of JUMAN and Wnn make these algorithms virtually useless for general research use.

Finally, the *n*-gram algorithm makes on the average one character error per sentence in error. In each of these cases, there is an obvious second choice. However, none of the errors can be fixed with the current pattern matching approach — higher level contextual information is required. Further, we doubt that the current level of computation linguistics technology would be capable of accurately quantifying such context. Hence, we believe we are reaching something of a lower bound on the error rate.

After completing evaluations on the EB training data, we executed a true open-set experiment for the *n*-gram algorithm on the FJ News database. This data had never been previously used for training by the *n*-gram algorithm. The performance of the *n*-gram algorithm was somewhat disappointing — initially 19% sentence errors. Over 75% of the errors involved misreadings of about five common characters. After examining the errors on the FJ News database, and making appropriate corrections to the dictionaries to reflect the errors we observed (and hence no longer making this an open-set test), we achieved the results shown in Table 4.

The FJ News database is somewhat special — it consists of sentences extracted from the several Usenet newsgroups in which people discuss various social activities. The data has an uncharacteristically high content of characters relating to people. This type of data was not well-represented in our training database and contains some characters that are traditionally difficult to read. Hence, a handful of new entries were required to improve performance.

To place the results presented here in perspective, recall that the evaluation databases contain sentences that were specially selected to use characters within the Jōyō kanji character set. The performance in Tables 3 and 4 should be regarded as a lower bound on performance more than an average. Clearly, as shown in Table 2, processing large databases poses many challenges in handling non-Jōyō characters — these problems ultimately dominate performance.

Finally, we investigated CPU time as a function of the length of the text. The results are shown in Figure 9. As noted previously, because the dynamic programming grid dimensions are a function of *n*-gram order and input length, and because the number of competing hypotheses are small, the CPU time required by the algorithm is linearly proportional to the input length. The CPU time shown in Figure 9 is computed on a Sun Sparcstation 10/30 with 128 Mbytes of memory for a program written entirely in C++. On the average, the processing time is about 1 msec per character. Because the current software loads the dictionaries in memory prior to processing, approximately 20 secs of CPU time is required for initialization. For small amounts of input text, this is the dominant factor in the total CPU time.

V. PROBLEMS

Unfortunately, as shown in the previous section, there are limitations to the current algorithm. Aside from the problem of unknown kanji characters, there are three common types of errors that occur. These are shown in Figure 10. Generally speaking, they suffer from the same fundamental problem — more sophisticated contextual information is required.

The first class of problems involves a misreading of the character for person, which is pronounced “hito.” The general problem in this case is determining whether to use the Chinese-derived reading, referred to as “onyomi,” or the Japanese-derived reading, referred to as “kunyomi” [6]. Usually, when the character occurs as part of a group of kanji characters, the

onyomi reading is used. When the character is read by itself (a 1-gram) the kunyomi reading is used. For many characters, it is difficult to predict which reading will be chosen.

The character underlined in Figure 10(a) presents such a case. It alternates between the onyomi reading “jing” and the kunyomi reading “hito.” In the first sentence, the correct reading is the kunyomi reading “hito” (“people” — the translation of the sentence is “Do work that is useful for people.”). In the second sentence, the correct reading is the onyomi reading “jing” (a slightly different usage — the translation of the sentence is “There are no Japanese victims involved.”). It is often difficult to resolve such cases without additional contextual information.

The second case, shown in Figure 10(b), is an example of a common problem in Japanese — counters. The first example is “ju ichi nin” (in English, “eleven people”). The second case is “hitori” (in English, “one person”). Depending on the type of object being referred to, a number followed by an object will be pronounced with a special ending. For example, “ichi” is the word for “one”, but when counting long slender objects such as pencils, we use “ippon no empitsu” for “one pencil.” This type of problem is best handled by rules. Pattern matching algorithms suffer from a combinatorial explosion problem — every number for every type of object would need to be handled.

Finally, there are cases where higher level contexts are required. In Figure 10(c), we present a reading of two characters as “monaka” and “saichū”. In the first sentence, the correct reading is “monaka” which is a bean-filled wafer (the sentence means “I like a Monakas”). In the second sentence, the correct reading is “saichū” which means “in the midst of” (the sentence means “I am in the midst of working”). There is no straightforward way of distinguishing the two readings without some knowledge of the meaning of the sentence or syntactic structure.

The second pair of sentences in Figure 10(c) contains a similar context-dependent case. The

basic meaning of that kanji is different in the two sentences: one means a wind, and it's pronounced "kaze" ("A cold wind is blowing."); the other means "style" and is pronounced "hu u" ("Please don't say it in such a way"). Again, choosing the correct reading in this case is quite difficult.

VI. SUMMARY

We have presented a high performance kanji to hiragana conversion algorithm. Its performance was shown to be superior to three public domain algorithms on two extensive evaluations. The algorithm currently handles a wide rage of common kanji characters, and can be easily extended to more difficult text by augmentation of its dictionaries.

We are still concerned about coverage in open-set testing. As we gain more experience with wider varieties of text, we will refine the dictionaries and minimize the number of entries required. We are confident that most problems concerning kanji to hiragana conversion can be dealt with, as we have demonstrated in this paper. The only real limitation at this point is the labor score in augmenting the dictionaries. Reducing the skill level required to augment the dictionaries goes a long way toward minimizing labor costs while guaranteeing consistency.

Ironically, most of the algorithms we have reviewed seem to suffer from this same problem — inadequate dictionaries. The reality is that kanji processing takes a lot of hard work. Most of the tools we have seen, especially the public domain ones in this paper, are very good, but do not have the investment in time required to develop their dictionaries and rules. In kanji processing, there is no real way around this — at some point you must deal with a large exception dictionary. We feel that our dictionaries are adequate for general tasks, but need improvement mainly in exception cases such as proper nouns.

Finally, it is clear there is room for combining some level of rule-based processing. One

important area of research will be to add a phrase analysis [25] and evaluate its usefulness. We believe that the basic algorithm presented here is valid, but can be improved somewhat through the addition of simple phrase level information. The challenge will be to produce a highly accurate segmentation algorithm.

VII. REFERENCES

1. D.H. Klatt, "Review of text-to-speech conversion for English," *Journal of the Acoustical Society of America*, vol. 82, no. 3, pp. 737-793, September 1987.
2. W.M. Fisher, G.R. Doddington, and K.M. Goudie-Marshall, "The DARPA Speech Recognition Research Database: Specifications and Status," in *Proceedings of DARPA Speech Recognition Workshop*, pp. 93-99, February 1986.
3. B. Wheatley and J. Picone, "Voice Across America: Toward Robust Speaker Independent Speech Recognition For Telecommunications Applications," *Digital Signal Processing: A Review Journal*, vol. 1, no. 2, pp. 45-63, April 1991.
4. J. Picone, G. Doddington, and D. Pallett, "Phone-Mediated Word Alignment for Speech Recognition Evaluation", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 3, pp. 559-562, March 1990.
5. J. Picone, T. Staples, and N. Arai, "A Phonetically Balanced Sentence Database for Japanese Language Continuous Speech Recognition," Technical Report No. TRDC-TM-93-09, Texas Instruments, Tsukuba Research and Development Center, 17 Miyukigaoka, Tsukuba, Ibaraki 305, Japan, June 1993.
6. W. Hadamitzky and M. Spahn, *Kanji and Kana: A Handbook and Dictionary of the Japanese Writing System*, Charles E. Tuttle, Tokyo, Japan, 1981.
7. H.I. Chaplin and S.E. Martin, *Japanese: A Manual of Reading and Writing*, Charles E. Tuttle, Tokyo, Japan, 1987.
8. Y. Ozaki, H. Tsuru, H. Nishioka, K. Yamada, and T. Yamada, Daijigen (Large Character Sources), Kadokawa Publishing, Tokyo, Japan, 1992 (ISBN4-04-012800-1 C0581).
9. K.R. Lunde, "Electronic Handling of Japanese Text," Japan.Inf Version 1.2, Adobe Systems Incorporated, 1585 Charleston Road, P.O. Box 7900, Mountain View, California, 94039-7900, USA, March 1992.
10. *The X Window System*, vols. 1-7, O'Reilly and Associates, Inc., Sebastopal, California, US, May 1990.
11. Unicode Consortium, *The 1991 Unicode Standard: Worldwide Character Encoding*, Version 1.0, vol. 1, Addison Wesley, Tokyo, Japan, 1991.
12. J. Gelb, *A Study of Writing*, University of Chicago Press, Chicago, USA, 1952.
13. Y. Myoki, "General Japanese Dictionary and Morphemic Analysis System," presented at the 42nd annual meeting of the Japan Information Processing Society, 1991. (JUMAN is available from the ftp site sparta.nmsu.edu in the file /incoming/juman -mcc-92-11-05.tar.Z).

14. M. Hagia, et. al., “An Overview of GMW+Wnn System,” in *Advances in Software Science and Technology,* Academic Press, Tokyo, Japan, vol. 1, pp. 133-156, 1989. (Wnn Version 4 is available from the ftp site uwtc.washington.edu in the file /pub/Japanese/Unix/Wnn4.106.tar.Z).

15. H. Takahashi, “KAKASI: Kanji Kana Simple Inversion Program,” version 2.2.3, available from the ftp site uwtc.washington.edu in the file /pub/Japanese/Unix/kakasi-2.2.3.tar.Z.

16. S. Itahashi, “Research Report on Standardization of Office Automation Equipment,” Japan Electronic Industry Development Association (JEIDA) — Special Group on Speech I/O, pp. 183, March 1992 (in Japanese).

17. T. Fujisaki, “Segmentation and Hiragana Conversion of Mixed Kanji-Hiragana Notation with Dynamic Programming,” *Technical Report of the Information Processing Society of Japan on Natural Language Processing*, no. 28-5, November 1981.

18. K. Takeda and T. Fujisaki, “Automatic Decomposition of Kanji Compound Words Using Stochastic Estimation,” *Transactions of the Information Processing Society of Japan*, vol. 28, no. 9, pp. 952-961, September 1987.

19. S. Kaizuka, I. Hujino, and S. Ono, *Chinese Character to Japanese Dictionary*, Kadokawa Book Company, Tokyo, Japan, 1991.

20. Y. Koine, et. al., (eds.), *Eiwa: The Electronic Book Version*, Kenkyusha Ltd., Tokyo, Japan, 1990 (available on CD-ROM in the Sony Electronic Book format: ISBN4-7674-3500-5 C0582 P6200E).

21. I. Shinmura (editor), *Kojien: The Electronic Book Edition, Third Edition*, Iwanami Books, Tokyo, Japan, 1990 (available on CD-ROM in the Sony Electronic Book format: ISBN4-001311-7 C0800 P7725E).

22. J.R. Deller, J.G. Proakis, J.H.L. Hansen, *Discrete Time Processing of Speech Signals*, MacMillian Publishing Co., New York, New York, USA, 1993.

23. J. Picone, K.M. Goudie-Marshall, G.R. Doddington, and W.M. Fisher, “Automatic Text Alignment for Speech System Evaluation,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-34, no. 4, pp. 780-784, August 1986.

24. C.S. Meyers and L.R. Rabiner, “A Level-Building Dynamic Time Warping Algorithm For Connected Word Recognition”, *IEEE Transactions On Acoustics, Speech, and Signal Processing*, vol. 29, no. 2, pp. 284-296, April 1981.

25. K. Shudo, T. Narahara, and S. Yoshida, “Morphological Aspects of Japanese Language Processing,” in *Proceedings of the 8th International Conference on Computational Linguistics*, Tokyo, Japan, pp. 1-8, October 1980.

List of Tables:

N-Gram Order	Number of Entries	Number of Active Entries
1	6,506	5,378 (83%)
2	68,407	43,070 (63%)
3	37,116	18,972 (51%)
4	21,242	11,221 (53%)
5	10,316	3,780 (37%)
6	1,737	744 (43%)
7	311	186 (60%)
8	96	62 (65%)
9	22	14 (64%)
Total	145,753	83,427 (57%)

Table 1. A summary of the current dictionary sizes. These dictionaries have been designed to cover text restricted to Jōyō kanji and some common EUC characters. The last column shows the number of entries actually used when processing a 910,000 sentence database.

Source	Summary	Number of Sentences	Jōyō Errors	EUC Errors
Language of America	Conversational English	14,214	1 (0%)	0 (0%)
Asahi Tensei Jingo	Newspaper Articles	78,730	17 (0%)	0 (0%)
Conversational Dictionary	Useful Phrases	3,476	0 (0%)	0 (0%)
Eiwa	Dictionary	29,954	6 (0%)	10 (0%)
Gendai	Dictionary	25,312	8 (0%)	298 (0%)
Sample	8 Misc. Text Sources	64,708	5 (0%)	1,092 (2%)
13 Easle	Mystery Game	695	0 (0%)	0 (0%)
Daijirin	Dictionary	100,473	1,607 (2%)	2,370 (0%)
Kojien	Dictionary	166,780	6,695 (4%)	13,239 (8%)
Britannica Shokomoku	Encyclopedia Britannica	382,643	27,684 (7%)	34,751 (9%)
Hyakuning Isshu	Old Poetry and Songs	1,555	195 (13%)	289 (19%)
FJ News	Internet Network News	38,735	991 (3%)	1,099 (3%)
Total		907,275	37,209 (4%)	53,148 (6%)

Table 2. An overview of the Electronic Book (EB) text database used to develop the *n*-gram algorithm. A summary of each text source is given, along with the number of sentences in the database. The last two columns contain the number of sentences that the *n*-gram algorithm currently cannot correctly parse due to an unknown Jōyō or EUC character combination. This data can be used to estimate the difficulty of the kanji contained in the text material. Many of these errors are attributable to proper nouns that simply require a single dictionary entry to fix. The first five sources have been used extensively to train the *n*-gram algorithm. Hence, the error rates are lowest on these sources.

Algorithm	Electronic Book (Training) Database			
	Combined Sentence Error Rate	Sentence Substitution Error Rate	Sentence Rejection Error Rate	Kanji Character Substitution Error Rate
JUMAN	44.1%	30.2%	13.9%	12.9%
Wnn	60.5%	47.3%	13.2%	34.2%
KAKASI	23.3%	23.0%	0.3%	4.6%
N-Gram	2.0%	2.0%	0.0%	0.3%

Table 3. A summary of the results of evaluations on a 1,000 sentence training database that is a subset of the EB text database. The *n*-gram algorithm, in closed-set testing, is shown to provide significantly better performance than its public domain counterparts.

Algorithm	FJ News Database			
	Combined Sentence Error Rate	Sentence Substitution Error Rate	Sentence Rejection Error Rate	Kanji Character Substitution Error Rate
JUMAN	39.2%	30.3%	8.9%	9.0%
Wnn	52.2%	20.1%	32.1%	21.8%
KAKASI	11.1%	11.0%	0.1%	1.8%
N-Gram	3.6%	3.6%	0.0%	0.5%

Table 4. A summary of the results of evaluations on the FJ News 1,000 sentence database. The *n*-gram algorithm is again superior in a closed set test. The remaining errors produced by the *n*-gram system on this database cannot be corrected by augmenting the dictionaries. The FJ News database contains a more restricted range of kanji than the EB database, so the results are overall somewhat better.

List of Figures:

Figure 9. A demonstration of the complexity of the n-gram algorithm. CPU time on a Sun Sparcstation SS 10/30 is shown as a function of the length of the input. Because the DP grid dimensions are a function of the maximum n-gram order and the input length, CPU time is linearly proportional to the length of the input.

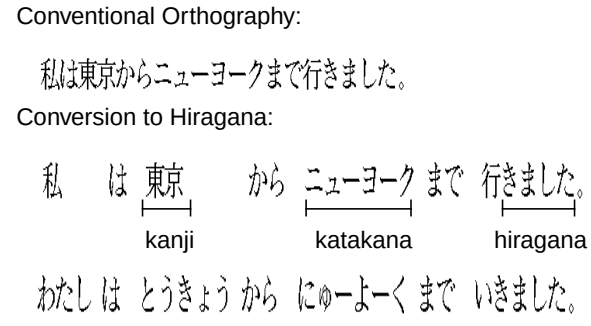

Figure 1. An example is given demonstrating the process of converting conventional orthography to hiragana. A romaji transcription (transliteration of Japanese words into roman letters) and an English translation are also shown.

a	i	u	e	o	ya	yu	ye	yo	wa	wi	we	wo
ア	イ	ウ	エ	オ	ヤ	ユ	イェ	ヨ	ワ	ウィ	ウェ	ウォ
ka	ki	ku	ke	ko	kya	kyu	kye	kyo	kwa	kwi	kwe	kwo
カ	キ	ク	ケ	コ	キャ	キュ	キェ	キョ	クァ	クィ	クェ	クォ
sa	shi	su	se	so	sha	shu	she	sho	swa	swi	swe	swo
サ	シ	ス	セ	ソ	シャ	シュ	シェ	ショ	スァ	スィ	スェ	スォ
ta	chi	tsu	te	to	cha	chu	che	cho	tsa	tsi	tse	tso
タ	チ	ツ	テ	ト	チャ	チュ	チェ	チョ	ツァ	ツィ	ツェ	ツォ
na	ni	nu	ne	no	nya	nyu	nye	nyo	nwa	nwi	nwe	nwo
ナ	ニ	ヌ	ネ	ノ	ニャ	ニュ	ニェ	ニョ	ヌァ	ヌィ	ヌェ	ヌォ
ha	hi	hu	he	ho	hya	hyu	hye	hyo	fa	fi	fe	fo
ハ	ヒ	フ	ヘ	ホ	ヒャ	ヒュ	ヒェ	ヒョ	ファ	フィ	フェ	フォ
ma	mi	mu	me	mo	mya	myu	mye	myo	mwa	mwi	mwe	mwo
マ	ミ	ム	メ	モ	ミャ	ミュ	ミェ	ミョ	ムァ	ムィ	ムェ	ムォ
ra	ri	ru	re	ro	rya	ryu	rye	ryo	rwa	rwi	rwe	rwo
ラ	リ	ル	レ	ロ	リャ	リュ	リェ	リョ	ルァ	ルィ	ルェ	ルォ
ga	gi	gu	ge	go	gya	gyu	gye	gyo	gwa	gwi	gwe	gwo
ガ	ギ	グ	ゲ	ゴ	ギャ	ギュ	ギェ	ギョ	グァ	グィ	グェ	グォ
za	ji	zu	ze	zo	ja	ju	je	jo	zwa	zwi	zwe	zwo
ザ	ジ	ズ	ゼ	ゾ	ジャ	ジュ	ジェ	ジョ	ズァ	ズィ	ズェ	ズォ
da	di	du	de	do	dya	dyu	dye	dyo	dwa	dwi	dwe	dwo
ダ	ディ	ドゥ	デ	ド	デャ	デュ	デェ	デョ	ドゥァ	ドゥィ	ドゥェ	ドゥォ
ba	bi	bu	be	bo	bya	byu	bye	byo	bwa	bwi	bwe	bwo
バ	ビ	ブ	ベ	ボ	ビャ	ビュ	ビェ	ビョ	ブァ	ブィ	ブェ	ブォ
pa	pi	pu	pe	po	pya	pyu	pye	pyo	pwa	pwi	pwe	pwo
パ	ピ	プ	ペ	ポ	ピャ	ピュ	ピェ	ピョ	プァ	プィ	プェ	プォ
	ti	tu			tya	tyu	tye	tyo	twa	twi	twe	two
	ティ	トゥ			テャ	テュ	テェ	テョ	トゥァ	トゥィ	トゥェ	トゥォ
va	vi	vu	ve	vo	vya	vyu	vye	vyo	vwa	vwi	vwe	vwo
ヴァ	ヴィ	ヴ	ヴェ	ヴォ	ヴャ	ヴュ	ヴェ	ヴョ	ヴゥァ	ヴゥィ	ヴゥェ	ヴゥォ
		fu			fya	fyu	fye	fyo				
		フ			フャ	フュ	フィェ	フョ				
zi	dji	dzu	ng	oo	q							
ズィ	ヂ	ヅ	ン	ヲ	ッ							

Note: Each symbol ending in a vowel has a long vowel counterpart. For example, “o” has a corresponding entry “o@” which is output for the long vowel ō.

Figure 2. A list of output symbols used in kanji to hiragana conversion is displayed. Each row containing a set of ASCII symbols is followed by a row containing the definition of the symbol in katakana (use of katakana in such a table is more standard). A dictionary entry contains three fields: a kanji *n*-gram sequence, a sequence of these ASCII symbols representing its reading, and a weight.

n-gram order = 1:

あ	a	1.00
い	i	1.00
ア	a	1.00
イ	i	1.00
学	ga ku	1.00
円	e ng	1.00
宴	e ng	1.00
川	ka wa	1.00
吏	JYO	0.01
浪	JYO	0.01
娃	EUC	0.01
阿	EUC	0.01

n-gram order = 2:

英雄	e i yu@	2.01
会社	ka i sha	2.01
書く	ka ku	2.01

n-gram order = 3:

歌舞伎	ka bu ki	3.02
食べる	ta be ru	3.02
面白い	o mo shi ro i	3.02

Figure 3. Excerpts from the 1-gram, 2-gram, and 3-gram dictionaries. A typical dictionary entry contains the kanji *n*-gram, the hiragana-like mapping, and a weight.

1-gram dictionary:

翼	tsu ba sa	1.00

2-gram dictionary:

右翼	u yo ku	2.01
左翼	sa yo ku	2.01

Figure 4. An example of a kanji character that requires entries in multiple dictionaries to completely define the possible contexts in which the character can occur. In this case, a default reading covering the most common interpretation is used in the 1-gram dictionary. The 2-gram dictionary is used to define the exceptions.

original text:

総代理店側は

"largest *n*-gram first" parse (incorrect):

総代　理店側　は

そうだい　りてんがわ　は

"largest *n*-gram first" parse using 1-grams (incorrect):

総代　理　店　側　は

そうだい　り　みせ　がわ　は

parse using exhaustive search (correct):

総　代理店　側　は

そう　だいりてん　がわ　は

Figure 5. An example of ambiguity in which a simple "largest *n*-gram first" parsing strategy fails. The kanji text is shown along with its hiragana representation. The first two characters represent a valid 2-gram. However, if this is chosen, then the next characters will not parse correctly. In our current system they would be incorrectly converted as 1-grams. The correct parse is shown as consisting of a 1-gram, a 3-gram, and two 1-grams (identification of word boundaries is the significant difference between the first and third readings.)

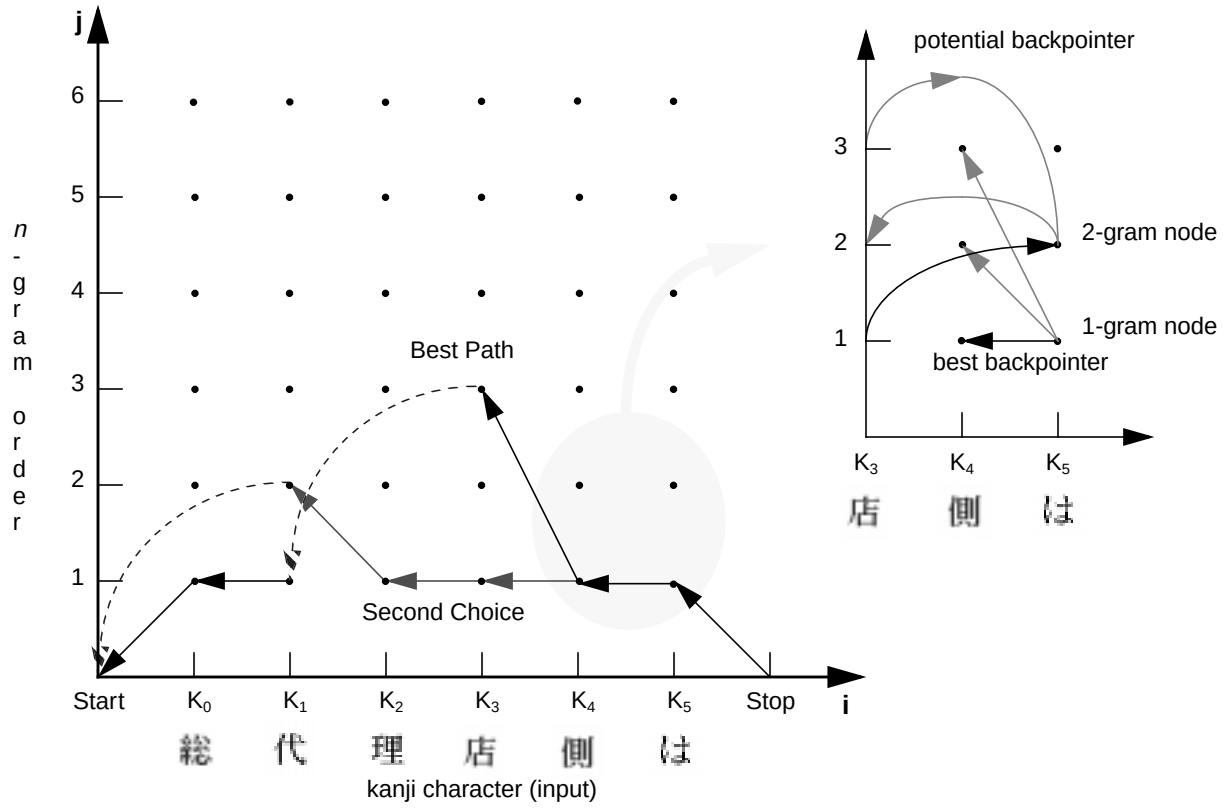

Figure 6. An example illustrating an efficient dynamic programming-based search for the best kanji reading. The horizontal axis corresponds to the input kanji characters. The vertical axis corresponds to the *n*-gram order. Node scores include the weight of an appropriate entry in the dictionary. Straight lines denote backpointers that point to the previous column; arcs denote backpointers that skip the previous column(s). Previous paths from a given node can extend backwards as far back as the maximum *n*-gram order. Theoretically, this would be prohibitively expensive in computational cost. In practice, there are very few competing paths that need to be considered. A diagram showing the first three allowable backpointers for a 1-gram and 2-gram node is shown to the right.

```
(Input)  総代理店側は
(Output) so u da i ri te ng ga wa ha
 Score: 6.02
 dict no. 01: [総] [so u]
 dict no. 03: [代理店] [da i ri te ng]
 dict no. 01: [側] [ga wa]
 dict no. 01: [は] [ha]
```

Figure 7. Example output from the *n*-gram algorithm for the example in Figure 6. The ASCII output is shown along with some debugging information indicating which dictionary entries were used.

Database:	Electronic Book		FJ News	
	Char.	# Errors	Char.	# Errors
JUMAN:	人	40	人	46
	物	13	良	29
	上	9	行	19
Wnn:	物	13	良	22
	上	9	話	20
	力	8	方	19
KAKASI:	方	7	方	17
	人	7	入	10
	国	6	後	5
N-Gram:	人	6	方	16
	後	2	様	4
	行	2	実	3

Figure 8. The top three most commonly misread characters are shown for each of the algorithms evaluated. Many of these characters have a strong alternate choice that can be disambiguated using a better model of the context. The characters misread by the *n*-gram algorithm are notoriously context dependent.

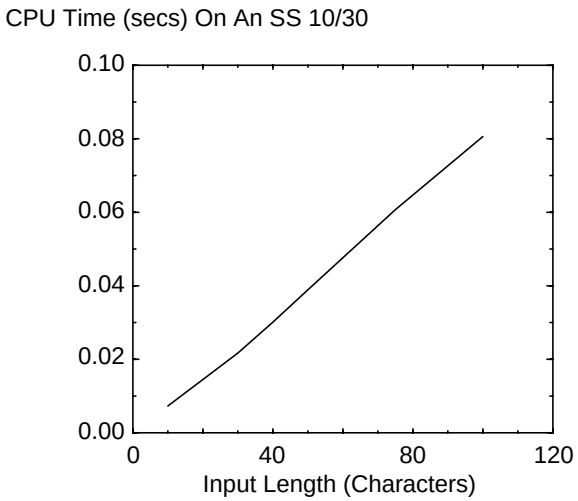

Figure 9. A demonstration of the $O(N)$ complexity of the *n*-gram algorithm. CPU time on a Sun Sparcstation SS 10/30 is shown as a function of the length of the input. Because the DP grid dimensions are a function of the maximum *n*-gram order and the input length, CPU time is linearly proportional to the length of the input.

(a) Choosing “onyomi” vs. “kunyomi” is difficult;

人の為になる仕事をする。

邦人の犠牲者はいません。

(b) counters need to be rule-driven;

十一人

一人

(c) extremely high-level context is often required.

私は最中が好きです。

私は仕事の最中です。

冷たい風が吹いています。

そんな風に言わないで下さい。

Figure 10. Three classes of problems for the *n*-gram algorithm that are beyond the reach of the current algorithm (details are provided above).

Joseph Picone received his Ph.D. in Electrical Engineering from Illinois Institute of Technology in 1983. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at Mississippi State University, where he also directs the Institute for Signal and Information Processing. He has previously been employed by Texas Instruments and AT&T Bell Laboratories. His primary research interest currently is the development of public domain speech recognition technology. Dr. Picone is a Senior Member of the IEEE and a registered Professional Engineer. He has also served in several capacities with the IEEE, including an associate editor of this journal. Dr. Picone has published more than 90 papers in the area of speech processing and has been awarded 8 patents.

Tom Staples earned his B.S.E.E at the University of Illinois at Chicago in 1990. From 1990 to 1992 he was a researcher at Nissan Motor Company's Central Engineering Laboratories in Yokosuka, Kanagawa Prefecture Japan. From 1992 to 1993 he was an engineer at the Tsukuba Research and Development Center of Texas Instruments in Tsukuba, Ibaraki Prefecture in Japan. From 1993 to 1998, Mr. Staples worked as a consultant to the Speech Research Branch of Texas Instruments in Dallas, Texas. He is currently with Nuance Communications, where he is developing telephone-based speech recognition technology.

Kazuhiro Kondo received the B. E. and M. E. degrees in electrical engineering from Waseda University in 1982 and 1984 respectively. From 1984 to 1992, he was with Central Research Laboratory, Hitachi Ltd., Tokyo, Japan. From 1992 to 1998, he was with Texas Instruments, where he conducted research into large vocabulary speech recognition. In 1998 he joined Conversational Computing Corp. His current research interests include conversational speech recognition and multimedia signal processing. He has been awarded 9 US patents in the field of speech signal processing. Mr. Kondo is a member of the IEEE, the Acoustical Society of Japan, and the Institute of Electronics, Information, and Communication Engineers of Japan.

Nozomi Arai was employed at the Tsukuba Research and Development Center of Texas Instruments in Tsukuba, Ibaraki Prefecture in Japan from 1991 to 1993, where she developed language engineering resources to support the development of Japanese speech recognition technology. She has previously been employed at Intel Corporation, and is currently employed by Hilti Japan Ltd.