Translation Memory Engines: A Look under the Hood and Road Test*

Timothy Baldwin
CSLI
Stanford University
Stanford, CA 94305 USA
tbaldwin@csli.stanford.edu

Abstract

In this paper, we compare the relative effects of segment order, segmentation and segment contiguity on the retrieval performance of a translation memory system. We take a selection of both bag-of-words and segment order-sensitive string comparison methods, and run each over both character- and word-segmented data, in combination with a range of local segment contiguity models (in the form of N-grams). Over two distinct datasets, we find that indexing according to simple character bigrams produces a retrieval accuracy superior to any of the tested word N-gram models. Further, in their optimal configuration, bag-of-words methods are shown to be equivalent to segment order-sensitive methods in terms of retrieval accuracy, but much faster. We also provide evidence that our findings scale over larger-sized translation memories.

1 Introduction

Translation memories (TMs) are a list of **translation records** (source language strings paired with a unique target language translation), which the TM system accesses in suggesting a list of target language (L2) **translation candidates** for a given source language (L1) input (Trujillo 1999; Planas 1998). For example, if a translator is attempting to translate a given Japanese document into English (i.e. L1 = Japanese, L2 = English), the TM system will take each string in the original Japanese document and attempt to locate similar Japanese string(s) in a database of previous Japanese–English translation data (the TM). In the case that some set of suitably similar strings is located in the TM, the translations for each such string is returned to the translator.

Translation retrieval (TR) is a description of the process of selecting from the TM a set of translation records (TRecs) of maximum L1 similarity to a given input. Traditionally, the TM system selects an arbitrary number of translation candidates falling within a preset corridor of similarity with the input string, and simply outputs these for manual manipulation by the user in fashioning the final translation.

The process of TR and intrinsic utility of TMs is based on three basic assumptions: (1) L1 documents are to some degree repetitive in their string composition or that they overlap to some degree with the translation data contained in the TM, and that the TM system is thus able to propose translation candidates for some subset of the strings contained in the original document; (2) a given string will be translated consistently irrespective of document context, such that if a good match is found for a given L1 string, the associated translation candidate will be a near-miss translation for the input string; and (3) L1 similarly is directly proportional to L2 translation similarity, such that TRecs which are more similar to the input will have translations which correspond more closely to the translation

*This paper is based on Baldwin (2001).

of the input. Given that these assumptions hold (as is generally the case in technical domains, for example), TMs provide a means to recycle translation data and save time in the translation process.

One key concern in TM systems is invisibility, in terms of their integration into the translation environment (e.g. into the translator's word processing software of choice) and system responsiveness. Essentially, a TM system should take nothing away from the translator in terms of translation utility, speed or accuracy, and should operate in such a way that the translator can easily ignore the TM system output if they feel that the translation candidate(s) are not suitable base material in translating a given input. For the purposes of this paper, we will ignore the integration issues, and focus instead on the TM back-end in terms of responsiveness (i.e. speed) and accuracy.

A key assumption surrounding the bulk of past TR research has been that the greater the match stringency/linguistic awareness of the retrieval mechanism, the greater the final retrieval accuracy will become. Naturally, any appreciation in retrieval complexity comes at a price in terms of computational overhead, potentially impacting upon system responsiveness. We thus follow the lead of Baldwin & Tanaka (2000) and Baldwin (2001) in asking the question: what is the empirical effect on retrieval performance of different match approaches? Here, retrieval performance is defined as the combination of retrieval speed and accuracy, with the ideal method offering fast response times at high accuracy.

In this paper, we choose to focus on retrieval performance within a Japanese–English TR context. One key area of interest with Japanese is the effect that *segmentation* has on retrieval performance. As Japanese is a non-segmenting language (does not explicitly delimit words orthographically), we can take the brute-force approach in treating each string as a sequence of characters (**character-based indexing**), or alternatively call upon segmentation technology in partitioning each string into words (**word-based indexing**). The string にわか雨 [*niwakaame*] "rain shower", e.g., would be segmented up into に · わ · か · 雨[1] under character-based indexing but treated as a single segment under word-based indexing.

Orthogonal to this is the question of sensitivity to *segment order*. That is, should our match mechanism treat each string as an unorganised multiset of terms (the **bag-of-words** approach), or attempt to find the match that best preserves the original segment order in the input (the **segment order-sensitive** approach)? In other words, should we treat 夏 · の · 雨 [*natu no ame*] "summer rain" and 雨 · の · 夏 [*ame no natu*] "a rainy summer" identically on account of them being made up of the same segments, or is the difference in segment order relevant in the context of TR? We tackle this issue by implementing a sample of representative bag-of-words and segment order-sensitive methods and testing the retrieval performance of each.

As a third orthogonal parameter, we consider the effects of *segment contiguity*. That is, do matches over contiguous segments provide closer overall translation correspondence than matches over non-contiguous segments? That is, should we consider 雨 · の· 多い ·冬 [*ame no ōi huyu*] "a rainy winter" more similar to 雨 · の · 冬 [*ame no huyu*] "a rainy winter" than 雨· マーク ·の· 多い ·冬[*ame māku no ōi huyu*] "a winter with many days marked as rainy", on account of the two strings having the same basic segment overlap in the same order, but the segments being more contiguous in the first instance? Segment contiguity is either explicitly modelled within the string match mechanism, or provided as an add-in in the form of segment N-grams (see below).

The major finding of this paper is that character-based indexing is shown to be superior to word-based indexing over a series of experiments and datasets. Furthermore, the bag-of-words methods we test are equivalent in retrieval accuracy to the more expensive segment order-sensitive methods, but superior in retrieval speed. Finally, segment contiguity models provide benefits in terms of both

[1]Segment boundaries are indicated by "·" throughout the paper.

retrieval accuracy and retrieval speed, particularly when coupled with character-based indexing. We thus provide clear evidence that high-performance TR is achievable with naive methods, and moreover that such methods outperform more intricate, expensive methods. That is, counter to intuition, TR methods which seemingly ignore the linguistic structure of strings are superior to methods which attempt to (at least superficially) model this linguistic structure.

Below, we review the orthogonal parameters of segmentation, segment order and segment contiguity (§ 2). We then present a range of both bag-of-words and segment order-sensitive string comparison methods (§ 3) and detail the evaluation methodology (§ 4). Finally, we evaluate the different methods in a Japanese–English TR context (§ 5), before concluding the paper (§ 6).

2 Basic Parameters

In this section, we review three parameter types that we suggest impinge on TR performance, namely segmentation, segment order, and segment contiguity.

2.1 Segmentation

Despite non-segmenting languages such as Japanese not making use of explicit segment delimiters such as whitespace, it is possible to artificially partition off a given string into constituent morphemes through a process known as **segmentation**.

Looking at past research on string comparison methods for TM systems, almost all systems involving Japanese as the source language rely on segmentation (Nakamura 1989; Sumita & Tsutsumi 1991; Kitamura & Yamamoto 1996; Tanaka 1997), with Sato (1992) and Sato & Kawase (1994) providing rare instances of character-based systems. This is despite Fujii & Croft (1993) providing evidence from Japanese information retrieval that character-based indexing performs comparably to word-based indexing. In analogous research, Baldwin & Tanaka (2000) compared character- and word-based indexing within a Japanese–English TR context and found character-based indexing to hold a slight empirical advantage.

The most obvious advantage of character-based indexing over word-based indexing is that there is no pre-processing overhead. Other arguments for character-based indexing over word-based indexing are that we: (a) avoid the need to commit ourselves to a particular analysis type in the case of ambiguity or unknown words (e.g. does 東京都 correspond to 東京 · 都 [*Tōkyō to*] “Tokyo prefecture” or 東 · 京都 [*higashi Kyōto*] “east Kyoto”?); (b) avoid the need for stemming/lemmatisation (e.g. recognising that お食べになる [*otabeninaru*] “eat (subject honorific)” and 食べます [*tabemasu*] “eat” both correspond to the verb lemma 食べる); and (c) to a large extent get around problems related to the normalisation of lexical alternation (e.g. differences in vowel length, such as between コンピュータ [*konpyūta*] “computer” and コンピューター [*konpyūtā*] “computer”).

Note that all methods described below are applicable to both word- and character-based indexing. To avoid confusion between the two lexeme types, we will collectively refer to the elements of indexing as **segments**.

2.2 Segment Order

Our expectation is that TRecs that preserve the segment order observed in the input string will provide closer-matching translations than TRecs containing those same segments in a different order.

As far as we are aware, there is no TM system operating from Japanese that does not rely on word/segment/character order to some degree. Tanaka (1997) uses pivotal content words identified by the user to search through the TM and locate TRecs which contain those same content words in the same order and preferably the same segment distance apart. Nakamura (1989) similarly gives preference to TRecs in which the content words contained in the original input occur in the same linear order, although there is the scope to back off to TRecs which do not preserve the original word order. Sumita & Tsutsumi (1991) take the opposite tack in iteratively filtering out NPs and adverbs to leave only functional words and matrix-level predicates, and find TRecs which contain those same key words in the same ordering, preferably with the same segment types between them in the same numbers. Sato & Kawase (1994) employ a more local model of *character* order in modelling similarity according to N-grams fashioned from the original string.

2.3 Segment contiguity

The intuition that contiguous segment matches indicate higher string similar than non-contiguous segment matches is captured either by embedding some contiguity weighting facility within the string match mechanism (such as weighted sequential correspondence — see below), or providing an independent model of segment contiguity in the form of segment N-grams. By N-gram, we simply mean a string of N contiguous segments. E.g., 夏 · の · 雨 [*natu no ame*] "summer rain" contains a total of two 2-grams (夏の and の雨), which are formed by taking all contiguous pairings of segments in the string.

The particular N-gram orders we test are simple unigrams (1-grams), pure bigrams (2-grams), and mixed unigrams/bigrams. These N-gram models are implemented as a pre-processing stage, following segmentation (where applicable). All this involves is mutating the original strings into N-grams of the desired order, while preserving the original segment order and segmentation schema. From the Japanese string 夏 · の · 雨 *natu·no·ame* "summer rain", for example, we would generate the following variants (common to both character- and word-based indexing):

1-gram:	夏 · の · 雨
2-gram:	夏の · の雨
Mixed 1/2-gram:	夏 · 夏の · の · の雨 · 雨

3 String Comparison Methods

As the starting point for evaluation of the three parameter types targeted in this research, we take two bag-of-words (segment order-oblivious) and three segment order-sensitive methods, thereby modelling the effects of segment order (un)awareness. We then run each method over both segmented and unsegmented data in combination with the various N-gram models proposed above, to capture the full range of parameter settings.

The particular bag-of-word approaches we target are the vector space model (Manning & Schütze 1999:p300) and "token intersection". For segment order-sensitive approaches, we test 3-operation edit distance and similarity, and also "weighted sequential correspondence".

We will illustrate each string comparison method by way of the following mock-up TM:

(1a) 夏の雨 [*natu no ame*] "summer rain"
(1b) 雨の夏 [*ame no natu*] "a rainy summer"
(1c) 雨の冬 [*ame no huyu*] "a rainy winter"
(1d) 真冬の雨 [*ma huyu no ame*] "mid-winter rain"
(1e) にわか雨 [*niwakaame*] "rain shower"

	TRec for comparison with (2) = 冬・の・雨				
	(1a) 夏・の・雨	(1b) 雨・の・夏	(1c) 雨・の・冬	(1d) 真・冬・の・雨	(1e) に・わ・か・雨
sim_{VSM}	$\frac{2}{3}$	$\frac{2}{3}$	**1**	$\frac{3}{2\sqrt{3}}$	$\frac{1}{2\sqrt{3}}$
sim_{TINT}	$\frac{4}{6}$	$\frac{2}{3}$	**1**	$\frac{6}{7}$	$\frac{2}{7}$
$dist_{3op}$	2	4	4	**1**	6
sim_{3op}	$\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{6}{7}}$	$\frac{1}{7}$
sim_{WSC}	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{6}$	$\mathbf{\frac{4}{5}}$	$\frac{2}{15}$

Table 1: Similarity values for each string pairing from the TM example

and the input:

(2) 冬の雨 [*huyu no ame*] "winter rain"

The similarity of (2) to each string in the TM under the different string comparison methods is presented in Table 1, with the best match for each method indicated in **boldface**.

All methods are formulated to operate over an arbitrary $sweight$ schemata, although in L1 string comparison throughout this paper, we assume that any segment made up entirely of punctuation is given an $sweight$ of 0, and any other segment an $sweight$ of 1.

All methods are subject to a threshold on **translation utility**, and in the case that the threshold is not achieved, the null string is returned. The various thresholds are as follows:

Comparison method	*Threshold*
Vector space model	0.5
Token intersection	0.4
3-operation edit distance	$len(IN)$
3-operation edit similarity	0.4
Weighted seq. correspondence	0.2

where IN is the input string, and len is the conventional segment length operator.

Various optimisations were made to each string comparison method to reduce retrieval time, of the type described by Baldwin & Tanaka (2000). While the details are beyond the scope of this paper, suffice to say that the segment order-sensitive methods benefited from the greatest optimisation, and that little was done to accelerate the already quick bag-of-words methods.

3.1 Bag-of-Words Methods

Vector Space Model

Within our implementation of the **vector space model** (VSM), the segment content of each string is described as a vector, made up of a single dimension for each segment type occurring within S or

T. A more intuitive way to visualise this is as a table, with each column representing the different segment types occurring in the two strings we are comparing, and two rows: one for each of the two strings we are comparing. E.g., the table used to calculate the similarity between (1a) and (2) under a 1-gram character-based model is as follows:

	freq(夏)	*freq*(冬)	*freq*(の)	*freq*(雨)
(2) = 夏 · の · 雨	1	0	1	1
(1a) = 冬 · の · 雨	0	1	1	1

The value of each vector component is given as the weighted frequency of that type according to its $sweight$ value. The string similarity of S (made up of segments $s_1, s_2, ...$) and T (made up of segments $t_1, t_2, ...$) is then defined as the cosine of the angle between vectors $\vec{S}$ and $\vec{T}$, respectively, calculated as:

$$sim_{VSM}(S,T) = \cos(\vec{S},\vec{T}) = \frac{\vec{S}\cdot\vec{T}}{|\vec{S}||\vec{T}|} = \frac{\sum_j s_j t_j}{\sqrt{\sum_j s_j^2}\sqrt{\sum_j t_j^2}}$$

For (2) and (1a), therefore, the cosine value is $\frac{1\times0+0\times1+1\times1+1\times1}{\sqrt{1^2+0^2+1^2+1^2}\times\sqrt{0^2+1^2+1^2+1^2}} = \frac{2}{3}$. The best match in the TM is with (1b), at a value of $\frac{3}{3} = 1$, as its segment content is identical to that for (2).

Token Intersection

The **token intersection** of S and T is defined as the cumulative intersecting frequency of segment types appearing in each of the strings, normalised according to the combined segment lengths of S and T using the Dice coefficient. That is, we count the number of segments which occur in both strings, and compare this to the length of the two strings. Formally, this equates to:

$$sim_{TINT}(S,T) \quad = \quad \frac{2\times\sum_{e\in S,T}\min\left(freq_S(e), freq_T(e)\right)}{len(S)+len(T)}$$

where each e is a segment occurring in either S or T, $freq_S(e)$ is defined as the $sweight$-based frequency of segment type e occurring in string S, and $len(S)$ is the segment length of string S, that is the $sweight$-based count of segments contained in S (similarly for T). Returning to the example of (2) and (1a), the similarity is $\frac{4}{6} = \frac{2}{3}$, an identical result to the vector space model. The best match in the TM is, once again, with (1b), with a similarity score of $\frac{6}{6} = 1$. Note that while the similarity values generated by token intersection and the vector space model happen to be identical over these two string pairs, there are significant divergences in the results generated by the two methods, particularly when there are segments with frequency greater than 1 in either of the strings in question.

3.2 Segment Order-sensitive Methods

3-op Edit Distance and Similarity

Essentially, the segment-based **3-operation edit distance** between strings S and T is the minimum number of primitive edit operations on single segments required to transform S into T (and vice versa). The three edit operations are *segment equality* (segments s_i and t_j are identical), *segment deletion* (delete segment s_i) and *segment insertion* (insert segment a into a given position in string S). The cost associated with each operation is determined by the $sweight$ values of the operand segments, with the exception of segment equality which is defined to have a fixed cost of 0. For the strings (2) and (1a), therefore, the 3-operation edit distance is $1 + 1 = 2$, as the most efficient way to generate 夏・の・雨 from 冬・の・雨 is to delete the single segment 冬 and insert the segment 夏.

Dynamic programming (DP) techniques are used to determine the minimum edit distance between a given string pair, following the classic 4-operation edit distance formulation of Wagner & Fischer (1974).[2] For 3-operation edit distance, the edit distance between strings $S = s_1s_2...s_m$ and $T = t_1t_2...t_n$ is defined as $dist_{3op}(S,T)$:

$$dist_{3op}(S,T) \quad = \quad d_3(m,n)$$

$$d_3(i,j) = \begin{cases} 0 & \mathit{if}\ i = 0 \wedge j = 0 \\ d_3(0, j-1) + \mathit{sweight}(t_j) & \mathit{if}\ i = 0 \wedge j \neq 0 \\ d_3(i-1, 0) + \mathit{sweight}(s_i) & \mathit{if}\ i \neq 0 \wedge j = 0 \\ \min \begin{pmatrix} d_3(i-1,j) + \mathit{sweight}(s_i), \\ d_3(i,j-1) + \mathit{sweight}(t_j), \\ m_3(i,j) \end{pmatrix} & \mathit{otherwise} \end{cases}$$

$$m_3(i,j) = \begin{cases} d_3(i-1, j-1) & \mathit{if}\ s_i = s_j \\ \infty & \mathit{otherwise} \end{cases}$$

It is possible to normalise operation edit distance $dist_{3op}$ into **3-operation edit similarity** sim_{3op} by way of:

$$sim_{3op}(S,T) \quad = \quad 1 - \frac{dist_{3op}(S,T)}{len(S) + len(T)}$$

The principal effect of normalisation is to prefer shorter strings over longer strings. E.g., in the instance that two strings are at edit distance 1 from the input string S, as a result of a single deletion and insertion operation, respectively, the shorter string will be preferred under normalisation as the denominator will be smaller ($2len(S) - 1$ vs. $2len(S) + 1$).

Weighted Sequential Correspondence

Weighted sequential correspondence (originally proposed in Baldwin & Tanaka (2000)) goes one step further than edit distance in analysing not only segment sequentiality, but also the contiguity of matching segments.

Weighted sequential correspondence associates an incremental weight (orthogonal to our $\mathit{sweight}$ weights) with each matching segment assessing the contiguity of left-neighbouring segments, in the manner described by Sato (1992) for character-based matching. Namely, the kth segment of a matched substring is given the multiplicative weight $\min(k, Max)$, where Max is a positive integer.

This weighting up of contiguous matches is facilitated through the DP algorithm given below:

$$dist_{WSC}(S,T) \quad = \quad s(m,n)$$

[2]The fourth operator in 4-operation edit distance is segment substitution.

$$s(i,j) = \begin{cases} 0 & \mathit{if}\ i = 0 \vee j = 0 \\ \max \begin{pmatrix} s(i-1,j), \\ s(i,j-1), \\ s(i-1,j-1) + m_w(i,j) \end{pmatrix} & \mathit{otherwise} \end{cases}$$

$$m_w(i,j) = \begin{cases} cm(i,j) \times \mathit{sweight}(i) & \mathit{if}\ s_i = s_j \\ 0 & \mathit{otherwise} \end{cases}$$

$$cm(i,j) = \begin{cases} 0 & \mathit{if}\ i = 0 \vee j = 0 \vee s_i \neq t_j \\ \min(Max, cm(i-1,j-1)+1) & \mathit{otherwise} \end{cases}$$

The final similarity is determined as:

$$\mathit{sim}_{WSC}(S,T) \quad = \quad \frac{2 \times \mathit{dist}_{WSC}(S,T)}{\mathit{len}_{WSC}(S) + \mathit{len}_{WSC}(T)}$$

where $\mathit{len}_{WSC}(S)$ is the weighted length of S, defined as:

$$\mathit{len}_{WSC}(S) \quad = \quad \sum_{i=1}^{m} \mathit{sweight}(s_i) \times \min(Max, i)$$

For the strings (2) and (1a), the similarity is $\frac{2\times(1+2)}{(1+2+3)+(1+2+3)} = \frac{1}{2}$.

3.3 Retrieval Speed Optimisation

Retrieval speed optimisation was carried out through analysis of the segment length of translation candidates (segment order-sensitive methods only), and segment overlap with the input (all methods). Minor local enhancements were also made to the 3-operation edit distance and similarity methods.

First, by pre-computing and storing the weighted segment length of each string (weighted according to both the $\mathit{sweight}$ values and the sequentiality increment in the case of weighted sequential correspondence), we can use the current top-ranking score α and weighted segment length of the input $\mathit{len}(IN)$, to determine upper and lower bounds on string lengths that could possibly better that score for the segment order-sensitive methods.

We can also limit the search space for all methods by using an "inverted file" description of the TM to identify those translation records with some segment overlap with the input. An inverted file is simply a list of all segments realised within the TM, and for those translation candidates containing a given segment, a description of segment frequency. The use of an inverted file (1) allows us to exclude from the search process all TRecs with no common segment component with the input, and (2) provides an immediate indication of the quality of match possible with each TRec overlapping in segment content with the input. For the bag-of-words methods, the data from the inverted file is plugged directly into the scoring functions, in combination with the weight for each segment. For the segment order-sensitive approaches, on the other hand, the segment overlap between each TRec and the input, combines with the scoring function to determine the optimal match score. We then order translation records according to the degree of match potentiality, and stop matching when the highest-ranking TRec not yet processed has an optimal score inferior to the best-scoring match to that point.

4 Evaluation Specifications

In this section, we outline the procedure used to evaluate the performance of different combinations of indexing methods, string comparison methods and N-gram models.

4.1 Details of the Dataset

As our main dataset, we used 3033 unique Japanese–English TRecs extracted from construction machinery field reports. Most TRecs comprise a single sentence, with an average Japanese character length of 27.7 and English word length of 13.3. Importantly, our dataset constitutes a controlled language, that is, a given word will tend to be translated identically across all usages, and only a limited range of syntactic constructions are employed.

In secondary evaluation of retrieval performance over differing data sizes, we extracted 61,236 Japanese–English TRecs from the JEIDA parallel corpus (Isahara 1998), which is made up of government white papers. The alignment granularity of this second corpus is much coarser than for the first corpus, with a single TRec often extending over multiple sentences. The average Japanese character length of each TRec is 76.3, and the average English word length is 35.7. The language used in the JEIDA corpus is highly constrained, although not as controlled as that in the first corpus.

The construction of TRecs from both corpora was based on existing alignment data, and no further effort was made to subdivide partitions (i.e. to attain sub-sentential alignment).

For Japanese word-based indexing, segmentation was carried out primarily with ChaSen v2.0 (Matsumoto et al. 1999), and where specifically mentioned, JUMAN v3.5 (Kurohashi & Nagao 1998) and ALTJAWS[3] were also used.

4.2 Semi-stratified Cross Validation

Retrieval accuracy was determined by way of 10-fold semi-stratified cross validation over the dataset. As part of this, all Japanese strings of length 5 characters or less were extracted from the dataset, and cross validation was performed over the remainder of the data, including the shorter strings in the training data (i.e. TM) on each iteration.

In N-fold stratified cross validation, the dataset is divided into N equally-sized partitions of uniform class distribution. Evaluation is then carried out N times, taking each partition as the held-out test data, and the remaining partitions as the training data on each iteration; the overall accuracy is averaged over the N data configurations. As our dataset is not pre-classified according to a discrete class description, we are not able to perform true data stratification over the class distribution. Instead, we carry out "semi-stratification" over the L1 segment lengths of the TRecs.

4.3 Evaluation of the Output

Evaluation of retrieval accuracy is carried out according to a modified version of the method proposed by Baldwin & Tanaka (2000). The first step in this process is to determine the set of "optimal"

[3]`http://www.kecl.ntt.co.jp/icl/mtg/resources/altjaws.html`

translations by way of the same basic TR procedure as described above, except that we use the held-out translation for each input to search through the L2 component of the TM. As for L1 TR, a threshold on translation utility is then applied to ascertain whether the optimal translations are similar enough to the model translation to be of use, and in the case that this threshold is not achieved, the empty string is returned as the sole optimal translation.

Next, we proceed to ascertain whether the actual system output coincides with one of the optimal translations, and rate the accuracy of each method according to the proportion of optimal outputs. If multiple outputs are produced, we select from among them randomly. This guarantees a unique translation output and differs from the methodology of Baldwin & Tanaka (2000), who judged the system output to be "correct" if the potentially multiple set of top-ranking outputs contained an optimal translation, placing methods with greater fan-out of outputs at an advantage.

So as to filter out any bias towards a given string comparison method in TR, we determine translation optimality based on both 3-operation edit distance (operating over English word bigrams) and weighted sequential correspondence (operating over English word unigrams). We then derive the final translation accuracy as the average of the accuracies from the respective evaluation sets. Here again, our approach differs from that of Baldwin & Tanaka (2000), who based determination of translation optimality exclusively on 3-operation edit distance (operating over word unigrams), a method which we found to produce a strong bias toward 3-operation edit distance in L1 TR.

In determining translation optimality, all punctuation and stop words were first filtered out of each L2 (English) string, and all remaining segments scored at a $sweight$ of 1. Stop words are defined as those contained within the SMART (Salton 1971) stop word list.[4]

Perhaps the main drawback of our approach to evaluation is that we assume a unique model translation for each input, where in fact, multiple translations of equivalent quality could reasonably be expected to exist. In our case, however, both corpora represent relatively controlled languages and language use is hence highly predictable.

5 Results and Supporting Evidence

5.1 Basic evaluation

In this section, we test our five string comparison methods over the construction machinery corpus, under both character- and word-based indexing, and with each of unigrams, bigrams and mixed unigrams/bigrams. The retrieval accuracies and times for the different string comparison methods are presented in Figures 1 and 2, respectively. Here and in subsequent graphs, "VSM" refers to the vector space model, "TINT" to token intersection, "3opD" to 3-op edit distance, "3opS" to 3-op edit similarity, and "WSC" to weighted sequential correspondence; the bag-of-words methods are labelled in italics and the segment order-sensitive methods in bold. In Figures 1 and 2, results for the three N-gram models are presented separately, within each of which, the data is sectioned off into the different string comparison methods. Weighted sequential correspondence was tested with a unigram model only, due to its inbuilt modelling of segment contiguity. Bars marked with an asterisk indicate a statistically significant[5] gain over the corresponding indexing paradigm (i.e. character-based indexing vs. word-based indexing for a given string comparison method and N-gram order). Times in Figure 2 are calibrated relative to 3-operation edit distance with word unigrams, and plotted against a logarithmic time axis.

[4]`ftp://ftp.cornell.cs.edu/pub/smart/english.stop`

[5]As determined by the paired *t* test ($p < 0.05$).

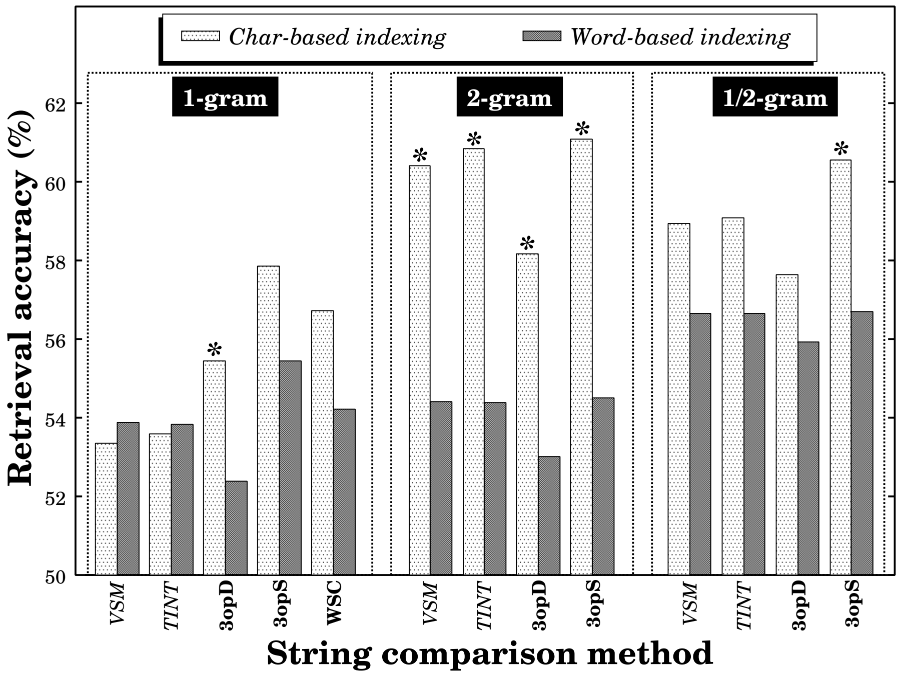

Figure 1: Basic retrieval accuracies

The results can be summarised as follows:

- Character-based indexing is consistently superior to word-based indexing, particularly when combined with bigrams or mixed unigrams/bigrams.
- In terms of raw translation accuracy, there is very little to separate the best of the bag-of-words methods from the best of the segment order-sensitive methods. [6]
- With character-based indexing, bigrams offer tangible gains in translation accuracy at the same time as greatly accelerating the retrieval process. With word-based indexing, mixed unigrams/bigrams offer the best balance of translation accuracy and computational cost.
- Weighted sequential correspondence is moderately successful in terms of accuracy, but grossly expensive.

Based on the above results, we judge bigrams to be the best segment contiguity model for character-based indexing, and mixed unigrams/bigrams to be the best segment contiguity model for word-based indexing, and for the remainder of this paper, present only these two sets of results.

It is important to note that Sato (1994) achieved sizeable speed improvements for the weighted sequential correspondence method using a massively parallel machine. However, given the disappointing accuracy of the method, this avenue of research does not seem worth pursuing for our purposes.

[6]With the possible exception of unigrams, where the bag-of-words methods were generally inferior to the segment order-sensitive methods. The significance of this result is diminished, however, by the gains in retrieval accuracy afforded to all methods by bigrams and mixed unigrams/bigrams.

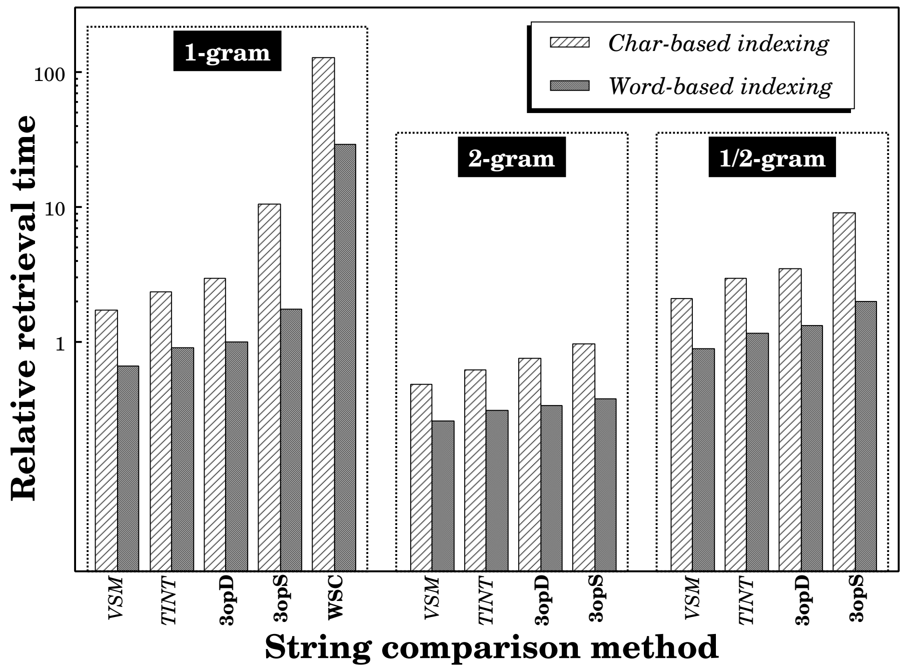

Figure 2: Basic unit retrieval times

While we have been able to confirm the finding of Baldwin & Tanaka (2000) that character-based indexing is superior to word-based indexing, we are no closer to determining why this should be the case. In the following sections we look to shed some light on this issue by considering each of: (i) the retrieval accuracy for other segmentation systems, (ii) the effects of lexical normalisation, (iii) the effects of kanji, and (iv) the scalability and reproducibility of the given results over different datasets. Finally, we present a brief qualitative explanation for the overall results.

5.2 The effects of segmentation and lexical normalisation

Above, we observed that segmentation consistently brought about a degradation in translation retrieval for the given dataset. Automated segmentation inevitably leads to errors, which could possibly impinge on the accuracy of word-based indexing. Alternatively, the performance drop could simply be caused somehow by our particular choice of segmentation module, that is ChaSen.

First, we used JUMAN to segment the construction machinery corpus, and evaluated the resultant dataset in the exact same manner as for the ChaSen output. Similarly, we ran a development version of ALTJAWS over the same corpus to produce two datasets, the first simply segmented and the second both segmented and lexically normalised. By lexical normalisation, we mean that each word is converted to its canonical form. The main segment types that normalisation has an effect on are verbs and adjectives (conjugating words), and also loan-word nouns with an optional long final vowel (e.g. モニター *monitā* "monitor" ⇒ モニタ *monita*) and words with multiple inter-replaceable kanji realisations (e.g. 充分 *zyūbuN* "sufficient" ⇒ 十分).

The retrieval accuracies for JUMAN, and ALTJAWS with and without lexical normalisation are presented in Figure 3, juxtaposed against the retrieval accuracies for character-based indexing (bigrams) and also ChaSen (mixed unigrams/bigrams) from Section 5.1. Asterisked bars indicate a

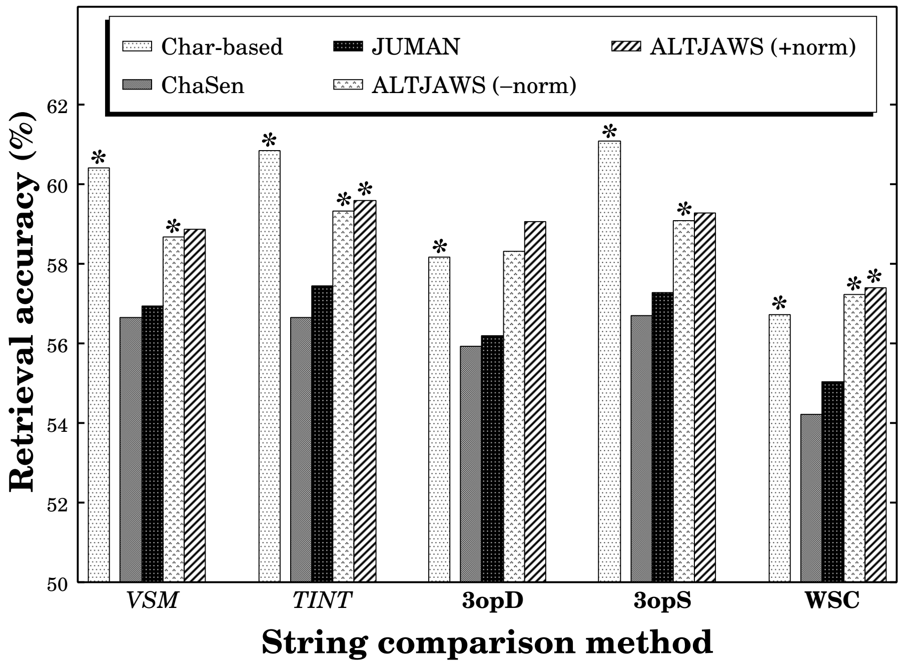

Figure 3: Results using different segmentation modules

statistically significant gain in accuracy over ChaSen.

Looking first to the results for JUMAN, there is a gain in accuracy over ChaSen for all string comparison methods. With ALTJAWS, also, a consistent gain in performance is evident with simple segmentation, the degree of which is significantly higher than for JUMAN. The addition of lexical normalisation enhances this effect marginally. Notice that character-based indexing (based on character bigrams) holds a clear advantage over the best of the word-based indexing results for all string comparison methods.

Based on the above, we can state that the choice of segmentation system does have a modest impact on retrieval accuracy, but that the effects of lexical normalisation are highly localised. In the following, we look to quantify the relationship between retrieval and segmentation accuracy.

In the next step of evaluation, we took a random sample of 200 TRecs from the original dataset, and ran each of ChaSen, JUMAN and ALTJAWS over the Japanese component of each. We then manually evaluated the output in terms of segment precision and recall, defined respectively as:

$$\text{Segment precision} = \frac{\text{\# correct segs in output}}{\text{Total \# segs in output}}$$

$$\text{Segment recall} = \frac{\text{\# correct segs in output}}{\text{Total \# segs in model data}}$$

One slight complication in evaluating the output of the three systems is that they adopt incompatible models of conjugation. We thus made allowance for variation in the analysis of verb and adjective complexes, and focused on the segmentation of noun complexes.

A performance breakdown for ChaSen, JUMAN and ALTJAWS is presented in Table 2. ALTJAWS was found to outperform the remaining two systems in terms of segment precision, while

	ChaSen	JUMAN	ALTJAWS
Average segments/TRec	13.0	12.0	11.7
Segment precision	98.3%	98.3%	98.6%
Segment recall	98.1%	96.2%	97.7%
Sentence accuracy	70.5%	59.0%	72.0%
Total segment types	650	656	634

Table 2: Segmentation performance

ChaSen and JUMAN performed at the exact same level of segment precision. Looking next to segment recall, ChaSen significantly outperformed both ALTJAWS and JUMAN. The source of almost all errors in recall, and roughly half of errors in precision for both ChaSen and JUMAN was katakana sequences such as *gēto-rokku-barubu* "gate-lock valve", transcribed from English. ALTJAWS, on the other hand, was remarkably successful at segmenting katakana word sequences, achieving a segment precision of 100% and segment recall approaching 99%. This is thought to have been the main cause for the disparity in retrieval accuracy for the three systems, aggravated by the fact that most katakana sequences were key technical terms.

To gain an insight into consistency in the case of error, we further calculated the total number of segment types in the output, expecting to find a core set of correctly-analysed segments, of relatively constant size across the different systems, plus an unpredictable component of segment errors, of variable size. The system generating the fewest segment types can thus be said to be the most consistent.

Based on the segment type counts in Table 2, ALTJAWS errs more consistently than the remaining two systems, and there is very little to separate ChaSen and JUMAN. This is thought to have had some impact on the inflated retrieval accuracy for ALTJAWS.

To summarise, there would seem to be a direct correlation between segmentation accuracy and retrieval performance, with segmentation accuracy on key terms (katakana sequences) having a particularly keen effect on translation retrieval. In this respect, ALTJAWS is superior to both ChaSen and JUMAN for the target domain. Additionally, complementing segmentation with lexical normalisation would seem to produce meager performance gains. Lastly, despite the slight gains to word-based indexing with the different segmentation systems, it is still significantly inferior to character-based indexing.

5.3 The effects of kanji

In this section, we look closer at the role that kanji play in translation retrieval, by analysing retrieval performance in the absence of kanji; this is achieved by replacing all kanji characters with their katakana (alphabetised) equivalents.

Our motivation in removing kanji is to study the semantic smoothing effects of individual kanji characters. To take an example, the single-segment nouns 操作 [*sōsa*] and 作動 [*sadō*] both translate into English as "operation" for the given domain, but would not match under word-based indexing. Character-based indexing, on the other hand, would recognise the partial overlap in character content, and in the process pick up on the semantic correspondence between the two words (assuming that the segment contiguity model includes unigrams).

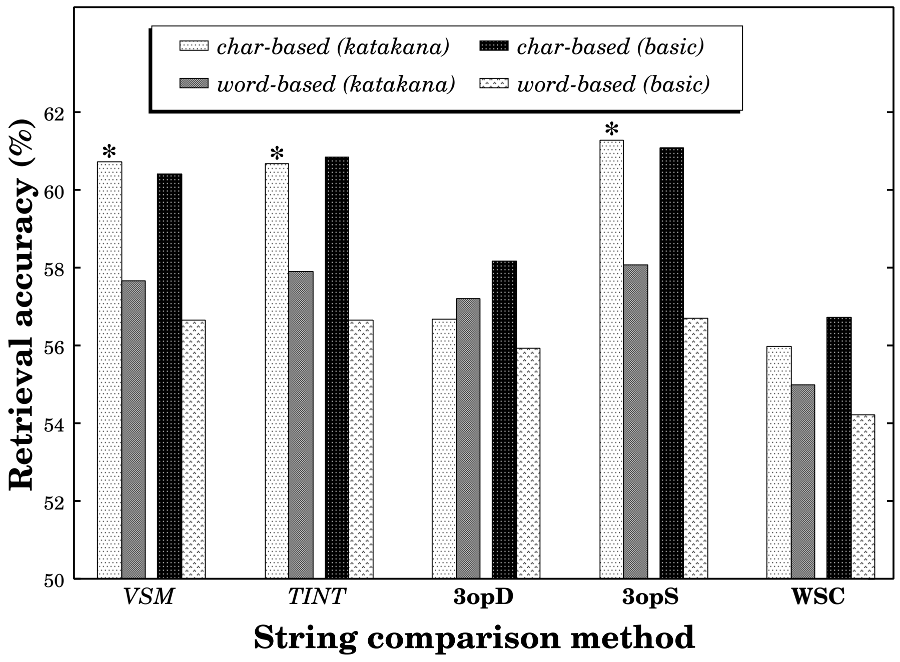

Figure 4: Results for fully alphabetised data

To test the effect of kanji characters (i.e. ideograms) on retrieval performance, we used ChaSen to convert all kanji and hiragana into katakana. In one version of this alphabetised data, the original segmentation was retained, and in a second version, each string was segmented off into individual katakana characters. The retrieval accuracies for character- and word-based indexing over the modified dataset are presented in Figure 4, alongside the results for character- and word-based indexing attained for fully lexically differentiated data (reproduced from Section 5.1).

As for the original experiment, character-based indexing was found to be superior to word-based indexing, to a level of statistical significance in most cases. Recall that the system is operating over fully alphabetised Japanese data, such that with character-based indexing, retrieval is based on a jumble of what are essentially syllable pairs. Amazingly, the retrieval accuracy for the alphabetised data was in many cases higher than for the original data, such that kanji if anything complicate rather than bolster translation retrieval.

5.4 Scalability of performance

All results to date have arisen from evaluation over a single dataset of fixed size. In order to validate the basic findings from above and observe how increases in the data size affect retrieval performance, we next ran the string comparison methods over differing-sized subsets of the JEIDA corpus.

We simulate TMs of differing size by randomly splitting the JEIDA corpus into ten partitions, and running the various methods first over partition 1, then over the combined partitions 1 and 2, and so on until all ten partitions are combined together into the full corpus. We tested all string comparison methods other than weighted sequential correspondence over the ten subsets of the JEIDA corpus. Weighted sequential correspondence was excluded from evaluation due to its overall sub-standard retrieval performance. The translation accuracies for the different methods over the ten datasets of

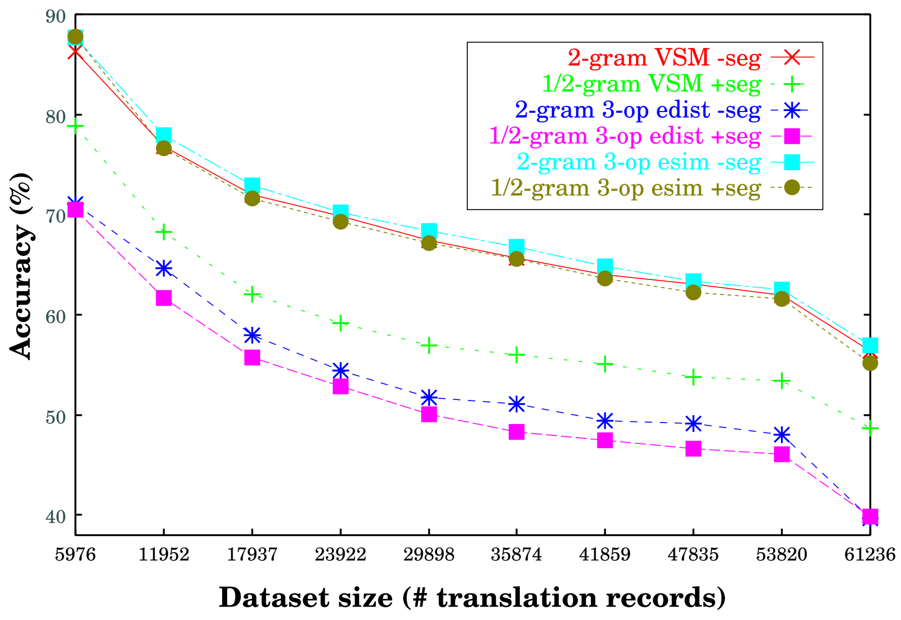

Figure 5: Retrieval accuracies over datasets of increasing size

varying size, are indicated in Figure 5, with each string comparison method tested under character bigrams ("2-gram −seg") and mixed word unigrams/bigrams ("1/2-gram +seg") as above. The results for token intersection have been omitted from the graph due to their being almost identical to those for VSM.

A striking feature of the graph is that it is right-decreasing, which is essentially an artifact of the inflated length of each TRec (see Section 4.1) and resultant data sparseness. That is, for smaller datasets, in the bulk of cases, no TRec in the TM is similar enough to the input to warrant consideration as a translation candidate (i.e. the translation utility threshold is generally not achieved). For larger datasets, on the other hand, we are having to make more subtle choices as to the final translation candidate.

One key trend in Figure 5 is the superiority of character- over word-based indexing for each of the three string comparison methods, at a relatively constant level as the TM size grows. Also of interest is the finding that there is very little to distinguish bag-of-words from segment order-sensitive methods in terms of retrieval accuracy in their respective best configurations.

As with the original dataset from above, 3-operation edit similarity was the strongest performer just nosing out (character bigram-based) VSM for line honours, with 3-operation edit distance lagging well behind.

Next, we turn to consider the mean unit retrieval times for each method, under the two indexing paradigms. Times are presented in Figure 6, plotted once again on a logarithmic scale in order to fit the full fan-out of retrieval times onto a single graph. VSM and 3-operation edit distance were the most consistent performers, both maintaining retrieval speeds in line with those for the original dataset at around or under 1.0 (i.e. the same retrieval time per input as 3-operation edit distance run over word unigrams for the construction machinery dataset). Most importantly, only minor increases in retrieval

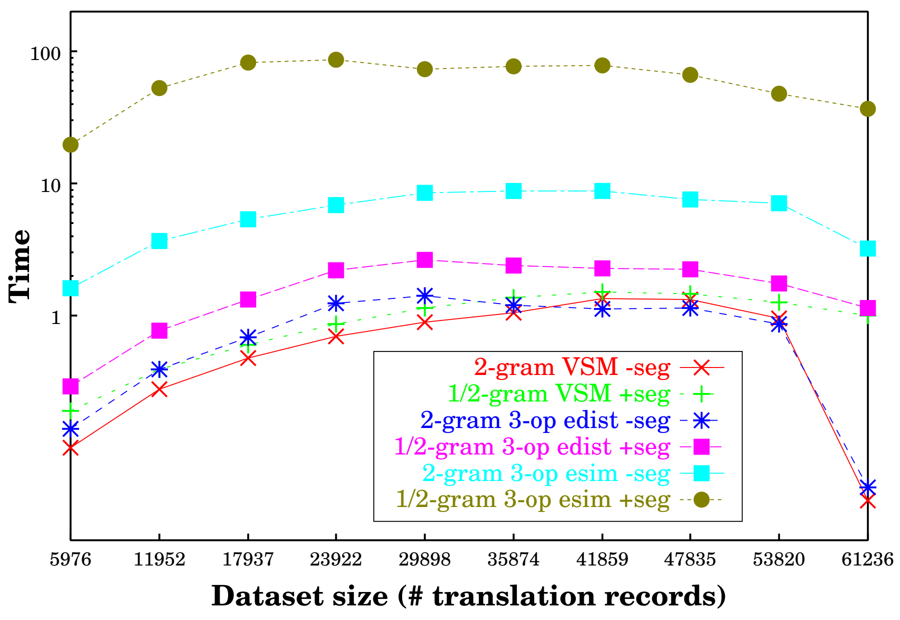

Figure 6: Relative unit retrieval times over datasets of increasing size

speed were evident as the TM size increased, which were then reversed for the larger datasets. All three string comparison methods displayed this convex shape, although the final running time for 3-operation edit similarity under character- and word-based indexing was, respectively, around 10 and 100 times slower than that for VSM or 3-operation edit distance over the same dataset.

To combine the findings for accuracy and speed, VSM under character-based indexing suggests itself as the pick of the different system configurations, combining both speed and consistent accuracy. That is, it offers the best overall retrieval performance.

5.5 Qualitative evaluation

Above, we established that character-based indexing is superior to word-based indexing for distinct datasets and a range of segmentation modules, even when segmentation is coupled with lexical normalisation. Additionally, we provided evidence to the effect that bag-of-words methods offer superior translation retrieval performance to segment order-sensitive methods. We are still no closer, however, to determining why this should be the case. Here, we seek to provide an explanation for these intriguing results.

First comparing character- and word-based indexing, we found that the disparity in retrieval accuracy was largely related to the scoring of katakana words, which are significantly longer in character length than native Japanese words. For the construction machinery dataset as analysed with ChaSen, for example, the average character length of katakana words is 3.62, as compared to 2.05 overall. Under word-based indexing, all words are treated equally and character length does not enter into calculations. Thus a katakana word is treated identically to any other word type. Under character-based indexing, on the other hand, the longer the word, the more segments it generates, and a single matching katakana sequence thus tends to contribute more heavily to the final score than other words. Effectively, therefore, katakana sequences receive a higher score than kanji and other sequences, producing a preference for TRecs which incorporate the same katakana sequences as the input. As noted above, katakana sequences generally represent key technical terms, and such weighting thus tends to be beneficial to retrieval accuracy.

We next examine the reason for the high correlation in retrieval accuracy between bag-of-words and segment order-sensitive methods in their optimum configurations (i.e. when coupled with character bigrams). Essentially, the probability of a given segment set permuting in different string contexts diminishes as the number of co-occurring segments decreases. That is, for a given string pair, the greater the segment overlap between them (relative to the overall string lengths), the lower the probability that those segments are going to occur in different orderings. This is particularly the case when local segment contiguity is modelled within the segment description, as occurs for the character bigram and mixed word uni/bigram models. For high-scoring matches, therefore, segment order sensitivity becomes largely superfluous, and the slight edge in retrieval accuracy for segment order-sensitive methods tends to come for mid-scoring matches, in the vicinity of the translation utility threshold.

6 Conclusion

This research has been concerned with the relative import of segmentation, segment order and segment contiguity on translation retrieval performance. We simulated the effects of word order sensitivity vs. bag-of-words word order insensitivity by implementing a total of five comparison methods: two bag-of-words approaches and three word order-sensitive approaches. Each of these methods was then tested under character-based and word-based indexing and in combination with a range of N-gram models, and the relative performance of each such system configuration evaluated. Character-based indexing was found to be superior to word-based indexing, particularly when supplemented with a character bigram model.

We went on to discover a strong correlation between retrieval accuracy and segmentation accuracy/consistency, and that lexical normalisation produces marginal gains in retrieval performance. By transcribing all strings into katakana, significant performance gains were observed for both indexing models, such that kanji appear to potentially impact negatively on retrieval performance. We further tested the effects of incremental increases in data on retrieval performance, and confirmed our earlier finding that character-based indexing is superior to word-based indexing. At the same time, we discovered that in their best configurations, the retrieval accuracies of our bag-of-words and segment order sensitive string comparison methods are roughly equivalent, but that the computational overhead for bag-of-words methods to achieve that accuracy is considerably lower than that for segment order sensitive methods.

Acknowledgements

This research was supported in part by the Research Collaboration between NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation and CSLI, Stanford University.

References

Baldwin, Timothy: 2001, ‘Low-cost, high-performance translation retrieval: Dumber is better’, in *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics*, Toulouse, pp. 18–25.

Baldwin, Timothy & Hozumi Tanaka: 2000, ‘The effects of word order and segmentation on translation retrieval performance’, in *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany, pp. 35–41.

Fujii, Hideo & W. Bruce Croft: 1993, 'A comparison of indexing techniques for Japanese text retrieval', in *Proc. of 16th International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'93)*, pp. 237–46.

Isahara, Hitoshi: 1998, 'JEIDA's English–Japanese bilingual corpus project', in *Proc. of the 1st International Conference on Language Resources and Evaluation (LREC'98)*, pp. 471–81.

Kitamura, E. & H. Yamamoto: 1996, 'Translation retrieval system using alignment data from parallel texts', in *Proc. of the 53rd Annual Meeting of the IPSJ*, vol. 2, pp. 385–6, (In Japanese).

Kurohashi, Sadao & Makoto Nagao: 1998, '*Nihongo keitai-kaiseki sisutemu JUMAN* [Japanese morphological analysis system JUMAN] version 3.5', Tech. rep., Kyoto University, (In Japanese).

Manning, Christopher D. & Hinrich Schütze: 1999, *Foundations of Statistical Natural Language Processing*, Cambridge, USA: MIT Press.

Matsumoto, Yuji, Akira Kitauchi, Tatsuo Yamashita & Yoshitaka Hirano: 1999, '*Japanese Morphological Analysis System ChaSen Version 2.0 Manual*', Tech. Rep. NAIST-IS-TR99009, NAIST.

Nakamura, N.: 1989, 'Translation support by retrieving bilingual texts', in *Proc. of the 38th Annual Meeting of the IPSJ*, vol. 1, pp. 357–8, (In Japanese).

Planas, Emmanuel: 1998, '*A Case Study on Memory Based Machine Translation Tools*', PhD Fellow Working Paper, United Nations University.

Salton, Gerald: 1971, *The SMART Retrieval System: Experiments in Automatic Document Processing*, Prentice-Hall.

Sato, Satoshi: 1992, 'CTM: An example-based translation aid system', in *Proc. of the 14th International Conference on Computational Linguistics (COLING '92)*, Nantes, Frances, pp. 1259–63.

Sato, Satoshi: 1994, 'Example-based translation and its MIMD implementation', in H. Kitano & J. Hendler, eds., *Massively Parallel Artificial Intelligence*, AAAI/MIT Press, pp. 171–201.

Sato, Satoshi & Takeshi Kawase: 1994, '*A High-Speed Best Match Retrieval Method for Japanese Text*', Tech. Rep. IS-RR-94-9I, JAIST.

Sumita, Eiichiro & Y. Tsutsumi: 1991, 'A practical method of retrieving similar examples for translation aid', *Transactions of the IEICE*, **J74-D-II**(10): 1437–47, (In Japanese).

Tanaka, Hideki: 1997, 'An efficient way of gauging similarity between long Japanese expressions', in *Information Processing Society of Japan SIG Notes*, vol. 97, no. 85, pp. 69–74, (In Japanese).

Trujillo, Arturo: 1999, *Translation Engines: Techniques for Machine Translation*, Springer Verlag.

Wagner, Robert A. & Michael J. Fischer: 1974, 'The string-to-string correction problem', *Journal of the ACM*, **21**(1): 168–73.