Balancing up Efficiency and Accuracy in Translation Retrieval

Timothy Baldwin[†] and Hozumi Tanaka[†]

This research looks at the effects of segment order and segmentation on translation retrieval performance for an experimental Japanese-English translation memory system. We implement a number of both bag-of-words and segment order-sensitive string comparison methods, and test each over character-based and word-based indexing. The translation retrieval performance of each system configuration is evaluated empirically through the notion of segment edit distance between the translation output and model translation. Our results indicate that character-based indexing is consistently superior to word-based indexing in terms of raw accuracy, although segmentation does have an accelerating effect on TM search times in combination with a number of retrieval optimisation techniques. Segment order-sensitive approaches are demonstrated to generally outperform bag-of-words methods, with 3-operation edit distance proving the most effective comparison method. We additionally reproduced the same basic results over alphabetised data as for lexically differentiated data containing kanji characters.

KeyWords: *translation memory, translation retrieval, segmentation, segment order*

1 Introduction

Translation memories (TMs) are a well-established technology within the human and machine translation fraternities, due to the high translation precision they afford. Essentially, 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 **translation candidates** which may be helpful to the translator in translating a given source language input.[1]

Naturally, TM systems have no way of accessing the target language (L2) equivalent of the source language (L1) input, and hence the list of *target language* translation candidates is determined based on *source language* similarity between the current input and translation examples within the TM, with translation equivalent(s) of maximally similar L1 string(s) given as the translation candidate(s). This is based on the assumption that structural and semantic similarities between L2 translations will be reflected in the original L1 equivalents.

† Tokyo Institute of Technology, Department of Computer Science

1 See Planas (1998) for a thorough review of commercial TM systems.

One reason for the popularity of TMs is the low operational burden they pose to the user, in that translation pairs are largely acquired automatically from observation of the incremental translation process, and translation candidates can be produced on demand almost instantaneously. To support this low overhead, TM systems must allow fast access into the potentially large-scale TM, but at the same time be able to predict translation similarity with high accuracy. Here, there is clearly a trade-off between **access/retrieval speed** and **predictive accuracy** of the retrieval mechanism. Traditionally, research on TM retrieval methods has focused on speed, with little cross-evaluation of the accuracy of different methods. We prefer to focus on accuracy, and present empirical data evidencing the relative predictive potential of different string comparison methods over different parameterisations.

In this paper, we focus on comparison of different retrieval algorithms for non-segmenting languages, based around a TM system from Japanese to English. Non-segmenting languages are those which do not involve delimiters (e.g. spaces) between words, and include Japanese, Chinese and Thai. We are particularly interested in the part the orthogonal parameters of segmentation and segment order play in the speed/accuracy trade-off. That is, by doing away with segmentation in relying solely on character-level comparison (**character-based indexing**), do we significantly degrade match performance, as compared to word-level comparison (**word-based indexing**)? Similarly, by ignoring segment order and treating each L1 string as a "bag of words", do we genuinely lose out over segment order-sensitive approaches? The main objective of this research is thus to determine whether the computational overhead associated with more stringent approaches (i.e. word-based indexing and segment order-sensitive approaches) is commensurate with the performance gains they offer.

To preempt what follows, the major contributions of this research are: (a) empirical evaluation of different comparison methods over actual Japanese-English TM data, focusing on four orthogonal retrieval paradigms; (b) the finding that, over the target data, character-based indexing is consistently superior to word-based indexing in identifying the translation candidate most similar to the optimal translation for a given input; (c) verification of this result over fully alphabetised input, suggesting ramifications for glyph-based non-segmenting languages such as Thai; and (d) empirical verification of the supremacy of segment order-sensitive string comparison methods over boolean match methods.

In the following sections we discuss the effects of segmentation and segment order (Section 2) and present a number of both bag-of-words and segment order-sensitive string comparison methods (Section 3), before going on to evaluate the different methods with character-based and word-based indexing (Section 4). We then conclude the paper in Section 5.

2 Segmentation and segment order

Using **segmentation** to divide strings into component words or morphemes has the obvious advantage of clustering characters into semantic units, which in the case of ideogram-based languages such as Japanese (in the form of kanji characters) and Chinese, generally disambiguates character meaning. The kanji character '弁', for example, can be used to mean any of "to discern/discriminate", "to speak/argue" and "a valve", but word context easily resolves such ambiguity. In this sense, our intuition is that segmented strings should produce better results than non-segmented strings.

Looking to past research on string comparison methods for TM systems, almost all systems involving Japanese as the source language rely on segmentation (e.g. (Nakamura 1989; Sumita and Tsutsumi 1991; Kitamura and Yamamoto 1996; Tanaka 1997)), with Sato (1992) and Sato and Kawase (1994) providing rare instances of character-based systems.

By avoiding the need to segment text, we: (a) alleviate computational overhead; (b) avoid the need to commit ourselves to a particular analysis type in the case of ambiguity; (c) avoid the issue of how to deal with unknown words; (d) avoid the need for stemming/lemmatisation; and (e) to a large extent get around problems related to the normalisation of lexical alternation (see Baldwin and Tanaka (1999) for a discussion of problems related to lexical alternation in Japanese). Additionally, we can use the commonly ambiguous nature of individual kanji characters to our advantage, in modelling semantic similarity between related words with character overlap. With word-based indexing, this would only be possible with the aid of a thesaurus.

Similarly for **segment order**, we would expect that translation records that preserve the segment (word) order observed in the input string would provide closer-matching translations than translation records containing those same segments in a different order. Naturally, enforcing preservation of segment order is going to place a significant burden on the matching mechanism, in that a number of different substring match schemata are inevitably going to be produced between any two strings, each of which must be considered on its own merits.

To the authors' knowledge, 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 translation records which contain those same content words in the same order and preferably the same segment distance apart. Nakamura (1989) similarly gives preference to translation records 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 translation records which do not preserve the original word order. Sumita and Tsutsumi (1991) take the opposite tack in iteratively filtering out NPs and adverbs to leave only functional words and matrix-level predicates, and find translation records which contain those same key words in the same ordering, preferably with the same segment types between them in the same numbers. Nirenburg et al. (1993) propose a segment order-sensitive method based on "string composition discrepancy", and incrementally relax the restriction on the quality of match required to include word lemmata, word synonyms and then word hypernyms, increasing the match penalty as they go. Sato and Kawase (1994) employ a more local model of *character* order in modelling similarity according to N-grams fashioned from the original string.

The greatest advantage in ignoring word/segment order is computational, in that we significantly reduce the search space and require only a single overall comparison per string pair. Below, we analyse whether this gain in speed outweighs any losses in retrieval performance.

3 String comparison methods

Due to our interest in the effects of both segment order and segmentation, we must have a selection of string comparison methods compatible with the various permutations of these two parameter types. We choose to look at a number of bag-of-words and segment order-sensitive methods which are compatible with both character-based and word-based indexing, and vary the input to model the effects of the two indexing paradigms. The particular bag-of-word approaches we target are the vector space model (Manning and Schütze 1999, p 300) and "token intersection", a simple ratio-based similarity method. For segment order-sensitive approaches, we test 3-operation and 4-operation edit distance and similarity, and also "weighted sequential correspondence".

All methods describe the degree of correspondence between two input strings TM_i and IN,[2] where we define TM_i as an L1 string taken from the TM and IN as the input string. For the edit distance methods, this correspondence takes the form of a distance, with more similar strings having smaller distances separating them and identical strings having an edit distance of 0. All other methods generate scaled similarities in the range $[0, 1]$, with identical strings having similarity 1.

One feature of all string comparison methods given here is that they have fine-grained discriminatory potential and are able to narrow down the final set of translation candidates to a handful of, and in most cases one, output. This was a deliberate design decision, and

2 Note that the ordering here is arbitrary, and that all the similarity methods described herein are commutative for the given implementations.

aimed at example-based machine translation applications, where human judgement cannot be relied upon to single out the most appropriate translation from multiple system outputs. In this, we set ourselves apart from the research of Sumita and Tsutsumi (1991), for example, who judge the system to have been successful if there are a total of 100 or less outputs, and a fair proportion of useful translations are contained within them. Note that it would be a relatively simple procedure to fan out the number of outputs to n in our case, by taking the top n ranking outputs.

For all string comparison methods, we weight different Japanese segment types according to their expected impact on translation, in the form of the *sweight* function:

Segment type	*sweight*
punctuation	0
other segments	1

3.1 String comparison methods used in this research

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 token occurring within TM_i or IN. The value of each vector component is given as the weighted frequency of that token according to its *sweight* value, such that any number of a given punctuation mark will produce a frequency of 0. The string similarity of TM_i and IN is then defined as the cosine of the angle between vectors $\vec{TM_i}$ and $\vec{IN}$, respectively, calculated as:

$$\cos(\vec{TM_i}, \vec{IN}) = \frac{\vec{TM_i} \cdot \vec{IN}}{|\vec{TM_i}||\vec{IN}|} \tag{1}$$

where dot product and vector length coincide with the standard definitions.

The strings TM_i of maximal similarity to IN are those which produce the maximum value for the vector cosine.

Note that VSM considers only segment frequency and is insensitive to segment order.

Token intersection

The token intersection of TM_i and IN is defined as the cumulative intersecting frequency of tokens appearing in each of the strings, normalised according to the combined segment lengths of TM_i and IN. Normalisation is by way of Dice's coefficient:

$$tint(TM_i, IN) = \frac{2 \times \sum_t \min\left(freq_{TM_i}(t), freq_{IN}(t)\right)}{len(TM_i) + len(IN)} \tag{2}$$

where each t is a segment token occurring in either TM_i or IN, $freq_S(t)$ is defined as the *sweight*-based frequency of token t 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.

As for VSM, the string(s) TM_i most similar to IN are those which generate the maximum value for $tint(TM_i, IN)$, and segment order does not take any part in calculation.

3- and 4-operation edit distance

The first of the segment order-sensitive methods is edit distance (Wagner and Fisher 1974; Planas and Furuse 1999). Essentially, the segment-based edit distance between strings TM_i and IN is the minimum number of primitive edit operations on segment units required to transform TM_i into IN (and vice versa). With 3-operation edit distance, we use the operations of *segment equality* (segment s_i in string S and segment t_j in string T are identical), *segment deletion* (delete segment s_i from string S) and *segment insertion* (insert segment a into a given position in string S); with 4-operation edit distance, *segment substitution* (substitute segment s_i in string S for segment a) makes up the fourth operation type. The cost associated with each operation over segments s_i and a is defined as:[3]

Operation	*Cost*
segment equality	0
segment deletion	$sweight(s_i)$
segment insertion	$sweight(a)$
segment substitution	$\max(sweight(s_i), sweight(a))$

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 and Fisher (1974). For 4-operation edit distance, the edit distance between strings $S = s_1 s_2 ... s_m$ and $T = t_1 t_2 ... t_n$ is defined as $D_{4op}(S, T)$:

$$D_{4op}(S, T) \quad = \quad d_4(m, n) \tag{3}$$

3 Note that the costs for deletion and insertion must be equal to maintain commutativity.

$$d_4(i,j) = \begin{cases} 0 & \textit{if}\, i = 0 \wedge j = 0 \\ d_4(0, j-1) + \textit{sweight}(t_j) & \textit{if}\, i = 0 \wedge j \neq 0 \\ d_4(i-1, 0) + \textit{sweight}(s_i) & \textit{if}\, i \neq 0 \wedge j = 0 \\ \min \begin{pmatrix} d_4(i-1, j) + \textit{sweight}(s_i), \\ d_4(i, j-1) + \textit{sweight}(t_j), \\ d_4(i-1, j-1) + m_4(i,j) \end{pmatrix} & \textit{otherwise} \end{cases} \tag{4}$$

$$m_4(i,j) = \begin{cases} 0 & \textit{if}\, s_i = s_j \\ \max(\textit{sweight}(s_i), \textit{sweight}(t_j)) & \textit{otherwise} \end{cases} \tag{5}$$

We modify this slightly to determine 3-operation edit distance, formalised over S and T as:

$$D_{3op}(S,T) = d_3(m,n) \tag{6}$$

$$d_3(i,j) = \begin{cases} 0 & \textit{if}\, i = 0 \wedge j = 0 \\ d_3(0, j-1) + \textit{sweight}(t_j) & \textit{if}\, i = 0 \wedge j \neq 0 \\ d_3(i-1, 0) + \textit{sweight}(s_i) & \textit{if}\, i \neq 0 \wedge j = 0 \\ \min \begin{pmatrix} d_3(i-1, j) + \textit{sweight}(s_i), \\ d_3(i, j-1) + \textit{sweight}(t_j), \\ m_3(i,j) \end{pmatrix} & \textit{otherwise} \end{cases} \tag{7}$$

$$m_3(i,j) = \begin{cases} d_3(i-1, j-1) & \textit{if}\, s_i = s_j \\ \infty & \textit{otherwise} \end{cases} \tag{8}$$

The reason that we distinguish between 3- and 4-operation edit distance is that the segment substitution operator is a compound operator, simultaneously involving a deletion and insertion operation. By maintaining segment deletion and insertion as separate operations, our intuition is that we should get a stronger sense of the true effort required to coerce an arbitrary string pair together, as a translator would have to do in adapting the final translation candidate to the needs of the original L1 input.

3- and 4-operation edit similarity

Above, we suggested the use of 3- and 4-operation edit distance as is without normalisation. This is possible due to them both explicitly modelling the degree of segment *disparity* between a given string pair, and hence capturing the degree of dissimilarity of the strings, relative to the minimum edit distance of zero. All other methods targeted herein model string overlap, and must be normalised in order to weight off the actual degree of overlap against the maximum potential overlap, in the form of the segment lengths of the target strings. While such normalisation is not obligatory for edit distance, it is certainly possible to normalise edit distance values to edit similarity values, scaled to the range $[0,1]$ as for other methods, a possibility we look to here.

The 3-operation edit distance between strings S and T can be translated into scaled 3-operation edit similarity by way of the following equation:

$$sim_{3op}(S,T) = 1 - \frac{D_{3op}(S,T)}{len(S) + len(T)} \tag{9}$$

Note that 3-operation edit similarity computed in this fashion is identical to the "sequential correspondence" method of Baldwin and Tanaka (2000), which determines the maximum sequential substring match between two strings.

Similarly, 4-operation edit similarity is derived from 4-operation edit distance by:

$$sim_{4op}(S,T) = 1 - \frac{D_{4op}(S,T)}{\max(len(S), len(T))} \tag{10}$$

Weighted sequential correspondence

Weighted sequential correspondence—the last of the segment order-sensitive methods—takes into account not only the sequentiality but also the contiguity of match. This is achieved by associating an incremental weight with each matching segment, assessing the contiguity of left-neighbouring segments, similarly to the character-based matching method of Sato (1992). Namely, the kth segment of a matched substring is given the multiplicative weight $\min(Max, k)$.

$$S_W(S,T) = s(m,n) \tag{11}$$

$$s(i,j) = \begin{cases} 0 & \textit{if}\, i = 0 \vee j = 0 \\ \max\begin{pmatrix} s(i-1,j), \\ s(i,j-1), \\ s(i-1,j-1) + m_W(i,j) \end{pmatrix} & \textit{otherwise} \end{cases} \tag{12}$$

$$m_W(i,j) = \begin{cases} cm(i,j) \times sweight(i) & \textit{if}\, s_i = s_j \\ 0 & \textit{otherwise} \end{cases} \tag{13}$$

$$cm(i,j) = \begin{cases} 0 & \textit{if}\, i = 0 \vee j = 0 \vee s_i \neq t_j \\ \min(Max, cm(i-1,j-1)+1) & \textit{otherwise} \end{cases} \tag{14}$$

This raw similarity is then normalised according to Dice's coefficient, similarly to token intersection:

$$sim_W(TM_i, IN) = \frac{2 \times S_W(TM_i, IN)}{len_W(TM_i) + len_W(IN)} \tag{15}$$

where len_W is defined for a string $S = s_1 s_2 ... s_m$ as:

$$len_W(S) = \sum_{j=1}^{m} sweight(s_j) \times \min(Max, j) \tag{16}$$

3.2 Retrieval speed optimisation

While this paper is mainly concerned with accuracy, we take a moment out here to discuss the potential to accelerate the proposed methods, to get a feel for their relative speeds in actual retrieval.

First, an "inverted file" can be used to gain an insight into the optimal attainable match for a given string pair. An inverted file is simply a list of each segment type contained in the TM, and an index of those translation records containing that token (including a frequency count for each). By determining the token frequency for each segment type contained in the input, we can plug the data from the inverted file straight into the equations for the bag-of-words methods, and simply return the translation record(s) which produced the highest score. For the segment order-sensitive methods, on the other hand, the inverted file allows us to determine the optimal match achievable with each translation record, by assuming that overlapping segments occur in identical order in the two target strings. By then working through the translation records in descending order of optimal score, we can halt the search process once the optimal score for the top-ranking translation not yet processed, falls below the best score actually observed to that point. For both indexing paradigms, this method also allows us to completely rule out strings with no segment overlap with IN, greatly reducing the string search space.

One further mechanism we can rely on with the segment order-sensitive methods, is to use the current top-ranking score in establishing upper and lower bounds on the segment length of strings which have the potential to better that score. For both edit distance methods, for example, we make the observation that for a current minimum edit distance of α, the following inequality over $len(TM_i)$ must be satisfied for TM_i to have a chance of bettering α:

$$len(IN) - \alpha \leq len(TM_i) \leq len(IN) + \alpha \tag{17}$$

Through these two methods, we were able to greatly speed up the string comparison process for word-based indexing and all methods other than weighted sequential correspondence (due to artificially high optimal match scores for translation records, under the assumption of full contiguity). The degree of reduction for character-based indexing was not as marked, due to the greater numbers of translation records sharing some character content with IN.

4 Evaluation

4.1 Evaluation specifications

Evaluation was partitioned off into character-based and word-based indexing for the various string comparison methods. For word-based indexing, segmentation was carried out with ChaSen v2.0b (Matsumoto et al. 1999). No attempt was made to post-edit the segmented output, in interests of maintaining consistency in the data. Segmented and non-segmented strings were tested using a single program, with segment length set to a single character for non-segmented strings.

As our dataset, we used 3043 unique translation records deriving from technical field reports on construction machinery manually translated from Japanese into English.[4] Translation records varied in size from single-word technical terms taken from a technical glossary, to multiple-sentence strings, at an average Japanese word length of 14.4 and character length of 27.7, and average English word length of 13.3. All Japanese strings of length 6 characters or more (a total of 2512 strings) were extracted from the dataset, leaving a residue glossary of technical terms (531 strings) as we would not expect to find useful matches in the TM. The retrieval accuracy over the 2502 full-length strings was then verified by 10-fold semi-stratified cross validation, including the glossary in the TM data on each iteration. By 10-fold semi-stratified cross-validation, we mean that the dataset was partitioned into 10 equally-sized subsets of roughly equivalent L1 segment length distribution. The TM system was then run over 10 iterations, taking one partition as the held-out input set, and the remaining 9 partitions as the TM data on each iteration.

Note that the test data was pre-partitioned into single technical terms, single sentences or sentence clusters, each constituting a single translation record. Partitions were taken as given in evaluation, whereas for real-world TM systems, the automation of this process comprises an important component of the overall system, preceding translation retrieval. While acknowledging the importance of this step and its interaction with retrieval performance, we choose to sidestep it for the purposes of this paper, and leave it for future research.

While the different methods are generally capable of focusing in on a small set of translation candidates for a given input, we enforce the constraint that a unique translation candidate (possibly the empty string – see below) must be generated for each input, in order to avoid any bias to methods with high output fan-out. This is done by breaking ties in translation potential, by randomly selecting one translation candidate from the set of outputs.

4 A superset of the dataset used by Baldwin and Tanaka (2000).

In an effort to make evaluation as objective and empirical as possible, appropriateness of the final translation candidate proposed by the different methods was evaluated according to the 3-operation edit distance between the translation candidate and the unique model translation. In this, we transferred the 3-operation edit distance method described above directly across to L2 (English), with segments as words and the following experimentally-validated *sweight* schema:

Segment type	*sweight*
punctuation	0
stop words	0.01
other words	1

Stop words are defined as those contained within the SMART (Salton 1971) stop word list.[5] The (unique) system output was judged to be correct if it was optimally close to the model translation, i.e. that there was no other translation candidate closer to the model translation in terms of 3-operation edit distance; the average optimal 3-operation edit distance from the model translation was 3.72.

We set the additional criterion that the different methods should be able to determine whether the top-ranking translation candidate is likely to be useful to the translator, and that no output should be given if the closest matching translation record was outside a certain range of "translation usefulness". In practice, this was set to the 3-operation edit distance between the model translation and the empty string (i.e. the edit cost of creating the model translation from scratch). This cutoff point was realised for the different string comparison methods by thresholding over the respective scores. The different thresholds settled upon experimentally for all string comparison methods are given in brackets in the second column of Table 1, with the threshold for edit distance methods dynamically set to the edit distance between the input and the empty string.

We set ourselves apart from conventional research on TM retrieval performance in adopting this objective numerical evaluation method. Traditionally, retrieval performance has been gauged by the subjective usefulness of the closest matching element of the system output (as judged by a human), and described by way of a discrete set of translation quality descriptors (e.g. (Nakamura 1989; Sumita and Tsutsumi 1991; Sato 1992)). Perhaps the closest evaluation attempts to what we propose are those of Planas and Furuse (1999) in setting a mechanical cutoff for "translation usability" as the ability to generate the model translation from a given

5 ftp://ftp.cornell.cs.edu/pub/smart/english.stop

translation candidate by editing less than half the component words, and Nirenburg et al. (1993) in calculating the weighted number of key strokes required to convert the system output into an appropriate translation for the original input. The method of Nirenburg et al. (1993) is certainly more indicative of true L2 usefulness, but is dependent on the competence of the translator editing the TM system output, and not automated to the degree our method is.

	Method	*Accuracy*	*Edit discrep.*	*Ave. outputs*	*Ave. time*
Char-Based Indexing	Vector space model (0.5)	51.56	0.85	1.03 (0.97)	1.72
	Token intersection (0.4)	51.44	0.75	1.07 (0.94)	2.39
	3-op edit distance ($len(IN)$)	58.19	0.50	1.36 (0.81)	3.01
	3-op edit similarity (0.4)	53.31	0.60	1.08 (0.95)	11.49
	4-op edit distance ($len(IN)$)	50.24	0.66	1.54 (0.79)	20.02
	4-op edit similarity (0.3)	51.56	0.59	1.17 (0.90)	30.83
	Weighted seq. corr, $Max = 2$ (0.2)	55.67	0.46	1.06 (0.95)	64.02
	Weighted seq. corr, $Max = 4$ (0.2)	53.48	0.66	1.06 (0.95)	137.83
Word-Based Indexing	Vector space model (0.5)	50.84 (−1.4%)	0.77	1.10 (0.93)	0.68
	Token intersection (0.4)	51.40 (−0.1%)	0.71	1.17 (0.89)	0.91
	3-op edit distance ($len(IN)$)	54.67 (−6.0%)	0.56	1.78 (0.72)	1.00
	3-op edit similarity (0.4)	51.92 (−2.6%)	0.62	1.17 (0.89)	1.80
	4-op edit distance ($len(IN)$)	48.08 (−4.3%)	0.76	2.51 (0.66)	3.39
	4-op edit similarity (0.3)	49.32 (−4.3%)	0.64	1.40 (0.84)	4.45
	Weighted seq. corr, $Max = 2$ (0.2)	52.44 (−5.8%)	0.50	1.15 (0.91)	12.85
	Weighted seq. corr, $Max = 4$ (0.2)	50.08 (−6.4%)	0.65	1.13 (0.93)	31.28

Table 1 Results for the different string comparison methods under character-based and word-based indexing

4.2 Results

The results for the different string comparison methods with character-based and word-based indexing are given in Table 1, with the two bag-of-words approaches partitioned off from the five segment order-sensitive approaches for each indexing paradigm (weighted sequential correspondence was tested twice, with varying values of the variable cutoff *Max*). "Accuracy" is an indication of the proportion of inputs for which an optimal translation was produced; character-based indexing accuracies in bold indicate a significant[6] advantage over the corresponding word-based indexing accuracy, and figures in brackets for word-based indexing indicate the relative performance gain over the corresponding character-based indexing con-

6 As determined by the paired t test ($p < 0.05$).

figuration. "Edit discrep." refers to the mean 3-operation edit distance discrepancy between the translation candidate and optimal translation(s) in the case of the translation candidate being sub-optimal. "Ave. outputs" describes the average number of translation candidates output by the system, with the figure in brackets being the proportion of inputs for which a unique translation candidate was produced; recall that a unique translation candidate is randomly selected for final evaluation purposes in the case of multiple outputs. "Ave. time" describes the average time taken to determine the translation candidate(s) for a single output, relative to the time taken for word-based 3-operation edit distance retrieval; note that the figures for word-based indexing include the times for on-line segmentation of the input. The best result in each column for each of character- and word-based indexing, is underlined.

Perhaps the most striking result is that character-based indexing produces a superior match accuracy to word-based indexing for *all* string comparison methods, although we must qualify this in saying that none of the gains were found to be statistically significant. While this finding is perhaps counterintuitive, it concurs with the results of Baldwin and Tanaka (2000) for an analogous TM system and also Fujii and Croft (1993) for information retrieval.

Looking to segment order, we see that 3-operation edit distance outperforms all other methods for both character- and word-based indexing, peaking at just over 58% for character-based indexing.[7] The relative performance of the remaining methods is variable, with the two bag-of-words methods being superior to or roughly equivalent to all segment order-sensitive methods other than 3-operation edit distance for word-based indexing, but the relative gain for the segment order-based methods under character-based indexing tending to exceed that for the bag-of-words methods. It is thus difficult to draw any hard and fast conclusion as to the relative merits of segment order-based versus bag-of-words methods, other than to say that 3-operation edit distance would appear to have a clear advantage over other methods.

The figures for edit discrepancy in the case of non-optimal translation candidate(s) are equally interesting, and suggest that on the whole, the various methods err more conservatively for character-based than word-based indexing. The most robust method is weighted sequential correspondence ($Max = 2$), at an edit discrepancy of 0.46 and 0.50 for character-based and word-based indexing, respectively.

All methods were able to produce just over one translation candidate on average, with all other than the edit distance methods returning a unique translation candidate around 90% of the time or better. Theoretically, it should be possible to generate slightly higher accuracies for methods with higher numbers of outputs (most notably the edit distance methods) through

7 All accuracies are well up on those quoted in Baldwin and Tanaka (2000) due to the modified English *sweight* schema and an enlarged dataset.

more careful selection of the final translation candidate. Preliminary testing of the scope for improvement here, suggests that there is certainly a correlation between the average number of outputs and the potential for improvement in both raw accuracy and edit discrepancy, a point we leave for future research.

Turning to speed, word-based indexing was found to be faster than character-based indexing across the board, for the simple reason that the number of character segments is always going to be greater than or equal to the number of word segments. The average segment lengths quoted above (27.7 characters vs. 14.4 words) indicate that we generally have twice as many characters as words in a given string. Additionally, the inverted file-based acceleration technique has a greater effect for word-based indexing than character-based indexing, accentuating any speed disparity. The exceptionally slow speeds for weighted sequential correspondence under character-based indexing and for higher values of *Max* in particular, are worrying. Indeed, despite the marginal lead of weighted sequential correspondence over other methods in terms of edit discrepancy, we suggest that its sluggish nature makes it inappropriate for on-line tasks, or that at best, any effort expended in speeding it up would be better invested in enhancing one of the other methods.

One interesting point to come out of the presented figures is that 3-operation edit distance is superior to 4-operation edit distance and similarity in all respects, and also that we lose out by normalising 3-operation edit distance to a similarity.

In summary, segmentation degrades retrieval accuracy but enhances access speed. The best justification for segmentation thus comes in terms of acceleration when used in combination with the proposed retrieval optimisation techniques, and if we are solely interested in accuracy and edit discrepancy, then our method of choice is 3-operation edit distance operating under character-based indexing.

4.3 The impact of kanji on the results

An immediate explanation for character-based indexing's empirical edge over word-based indexing is the semantic smoothing effects of individual kanji characters, alluded to above (Section 2). To take an example, the single-segment nouns 操作 [*sōsa*] and 作動 [*sadō*] are synonyms and both translated as "operation" in the given domain, but would not match under word-based indexing. Character-based indexing, on the other hand, would recognise the overlap in character content, and in the process pick up on the semantic correspondence between the two words.

To test the effect of kanji characters (i.e. ideograms) on translation retrieval performance,

	Method	*Accuracy*	*Edit discrep.*	*Ave. outputs*	*Ave. time*
Char-Based Indexing	Vector space model (0.5)	48.22	1.22	1.03 (0.97)	3.28
	Token intersection (0.4)	49.02	0.97	1.05 (0.95)	4.53
	3-op edit distance ($len(IN)$)	55.47	0.53	1.35 (0.81)	42.84
	3-op edit similarity (0.4)	53.07	0.68	1.06 (0.95)	86.62
	4-op edit distance ($len(IN)$)	50.70	0.68	1.42 (0.81)	98.60
	4-op edit similarity (0.3)	51.30	0.65	1.17 (0.90)	139.32
	Weighted seq. corr, $Max = 2$ (0.2)	**55.35**	0.62	1.05 (0.96)	182.63
	Weighted seq. corr, $Max = 4$ (0.2)	54.35	0.69	1.04 (0.97)	218.93
Word-Based Indexing	Vector space model (0.5)	**52.54** (+9.0%)	0.80	1.10 (0.93)	0.63
	Token intersection (0.4)	**51.94** (+6.0%)	0.75	1.19 (0.89)	0.84
	3-op edit distance ($len(IN)$)	55.15 (−0.6%)	0.57	1.85 (0.74)	0.93
	3-op edit similarity (0.4)	52.30 (−1.4%)	0.64	1.19 (0.89)	1.65
	4-op edit distance ($len(IN)$)	47.93 (−5.5%)	0.75	2.44 (0.67)	2.98
	4-op edit similarity (0.3)	50.26 (−2.0%)	0.66	1.41 (0.84)	3.93
	Weighted seq. corr, $Max = 2$ (0.2)	52.26 (−5.6%)	0.65	1.17 (0.91)	12.90
	Weighted seq. corr, $Max = 4$ (0.2)	51.22 (−5.8%)	0.65	1.14 (0.92)	27.41

Table 2 Results for the different string comparison methods over alphabetised (katakana-transcribed) data

we used ChaSen to convert all kanji and hiragana into katakana, generating an essentially alphabetic version of each string, analogous to the case of Thai. In one version of this alphabetised data, the original segmentation was retained, and in a second version, each string was segmented off into individual characters. We then ran the same methods over this modified input, using exactly the same technique as for the original experiment. The results are presented in Table 2, with times calculated relative to 3-operation edit distance in the first experiment.

Here, we find that for word-based indexing, most methods performed marginally better over the alphabetised data than over the original data preserving the full heterogeneity of hiragana, katakana and kanji. As for the original experiment, the segment order-sensitive methods perform better under character-based indexing than word-based indexing, to a level of statistical significance for weighted sequential correspondence ($Max = 2$). This trend was reversed for the bag-of-words methods, with both VSM and token intersection suffering a significant degradation in accuracy under character-based indexing. Once again, 3-operation edit distance proved the clear victor on all fronts, with the only blemish being its running time under character-based indexing.

One immediate conclusion which can be drawn from this is that, in the absence of segment sensitivity, segmentation is required in order to maintain performance levels under character-based indexing, when operating over homogeneous alphabetised data. Perhaps more importantly, that the same smoothing effect was observed for character-based indexing when operating over both lexically differentiated and alphabetised script, would tend to suggest that smoothing is not tied to kanji characters to the degree we had originally predicted. Having said this, returning to our "operation" synonym example from above, we notice that the overlap in kanji is reflected in overlap in pronunciation, which is retained in the katakana version. In this sense, the smoothing effect for katakana-based input can be said to draw on the same basic mechanism as for the original lexically differentiated data.

It is possible to translate these results for Japanese across to other non-segmenting languages, notably Chinese and Thai. Due to its ideogrammatic founding, we would expect to get similar results to those for the first experiment with Chinese, that is segmentation would harm rather than aid translation retrieval accuracy and impinge only slightly on retrieval speed for most methods. For Thai, we can apply the results from the second experiment in postulating that the main advantage in segmenting strings would be computational in pruning the search space.

As an aside, it is important to pick up on the blowout in running times for character-based indexing in the second experiment, again suggesting that segmentation has a place in pruning the search space and reducing access times. The slowdown for the edit distance and similarity methods is particularly marked, and relates to the inverted file-based acceleration method failing to a large degree due to the homogeneity of the data. This does not occur for word-based indexing because segmentation produces segment differentiation. The relatively lesser degree of slowdown for weighted sequential correspondence when compared to the original experiment, is largely because the running time is bounded by the size of the TM; a large portion of the TM was searched over in the original experiment, such that the relative slowdown was buffered.

4.4 Miscellaneous reflections

One way in which ChaSen could conceivably have affected retrieval performance is that technical terms tended to be over-segmented. Experimentally combining recognised technical terms into a single segment (particularly in the case of contiguous katakana segments in the manner of Fujii and Croft (1993)), however, degraded rather than improved retrieval performance for both character-based and word-based indexing. As such, this side-effect of ChaSen would not appear to have impinged on retrieval accuracy. Additionally, over-segmentation was consistent on the whole, such that parts of the same whole could be matched together under word-based indexing as well as character-based indexing.

One other plausible reason for the unexpected results is that the dataset could have been in some way inherently better suited to character-based indexing than word-based indexing, although the fact that the results were cross-validated would tend to rule out this possibility.

Interestingly, weighted sequential correspondence consistently performed better with *Max* set to 2. This contradicts the finding of Sato (1992) that a setting of 4 was optimal for character-based indexing.

To return to the original question posed above of retrieval speed vs. accuracy, the segment order-sensitive edit distance approach would seem to hold a genuine edge over the other methods in terms of accuracy and edit discrepancy, to an order that would suggest the extra computational overhead is warranted, in both accuracy and translation discrepancy. It must be said that the TM used in evaluation was too small to get a genuine feel for the computational overhead that would be experienced in a real-world TM system context of potentially millions rather than thousands of translation records.

5 Concluding remarks

This research is concerned with the relative import of segment order and segmentation on translation retrieval performance for a TM system. We modelled the effects of segment order sensitivity vs. bag-of-words segment order insensitivity by implementing a total of seven string comparison methods: two bag-of-words approaches (the vector space model and "token intersection") and five segment order-sensitive approaches (3- and 4-operation edit distance and similarity, and "weighted sequential correspondence"). Each of these methods was then tested under character-based and word-based indexing, to determine what effect segmentation would have on retrieval performance. Empirical evaluation based around the L2 3-operation edit distance of proposed translation candidates revealed that character-based indexing consistently produced greater accuracy than word-based indexing, and that the segment order-sensitive 3-operation edit distance method clearly outperformed all other methods under both indexing paradigms. We then went on to analyse the effect of kanji ideograms on the superiority of character-based indexing, and concluded that while individual kanji characters may have some smoothing effect, a fully alphabetised context produces the same basic result. One unexpected benefit of segmentation was the speed-up of TM search times when used in combination with a number of optimisation techniques.

The main area in which we feel this research could be enhanced is to validate the findings of this paper in expanding evaluation to other domains and test sets, which we leave as an item for future research. We also skirted around the issue of translation record partitioning, and wish to investigate how different partitioning methods perform against each other.

Acknowledgements

Vital input into this research was received from Francis Bond (NTT) and Emmanuel Planas (NTT).

Reference

Baldwin, T., and Tanaka, H. (1999). "The applications of unsupervised learning to Japanese grapheme-phoneme alignment." In *Proc. of the ACL Workshop on Unsupervised Learning in Natural Language Processing*, pp. 9–16.

Baldwin, T., and Tanaka, H. (2000). "The Effects of Word Order and Segmentation on Translation Retrieval Performance." In *Proc. of the 18th International Conference on Computational Linguistics (COLING 2000)*, pp. 35–41.

Fujii, H., and Croft, W. (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.

Kitamura, E., and Yamamoto, H. (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).

Manning, C., and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* MIT Press.

Matsumoto, Y., Kitauchi, A., Yamashita, T., and Hirano, Y. (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).

Nirenburg, S., Domashnev, C., and Grannes, D. (1993). "Two Approaches to Matching in Example-Based Machine Translation." In *Proc. of the 5th International Conference on Theoretical and Methodological Issues in Machine Translation (TMI-93)*, pp. 47–57.

Planas, E. (1998). "*A Case Study on Memory Based Machine Translation Tools.*" PhD Fellow Working Paper, United Nations University.

Planas, E., and Furuse, O. (1999). "Formalizing Translation Memories." In *Proc. of Machine Translation Summit VII*, pp. 331–9.

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

Sato, S. (1992). "CTM: An Example-Based Translation Aid System." In *Proc. of the 14th International Conference on Computational Linguistics (COLING '92)*, pp. 1259–63.

Sato, S., and Kawase, T. (1994). "*A High-Speed Best Match Retrieval Method for Japanese Text.*" Tech. rep. IS-RR-94-9I, JAIST.

Sumita, E., and Tsutsumi, Y. (1991). "A Practical Method of Retrieving Similar Examples for Translation Aid." *Transactions of the IEICE*, *J74-D-II*(10), 1437–47. (In Japanese).

Tanaka, H. (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).

Wagner, A., and Fisher, M. (1974). "The String-to-String Correction Problem." *Journal of the ACM*, *21*(1), 168–73.

Timothy Baldwin: Timothy Baldwin received his BSc in computer science from the University of Melbourne in 1994, and his BA in linguistics and Japanese also from the University of Melbourne in 1995. He completed an MEng in computer science at the Tokyo Institute of Technology in 1998, and is currently working towards his PhD at that same institution. His current research interests include machine translation, computational lexical semantics, word sense disambiguation, machine learning and computer-aided language learning.

Hozumi Tanaka: Hozumi Tanaka received his BS in 1964 and MS in 1966, both from the Tokyo Institute of Technology. In 1966 he joined the Electro Technical Laboratories, Tsukuba. He received his DEng in 1980 from the Tokyo Institute of Technology, and has been a professor in the Department of Computer Science of that same university since 1983. His main research background is in artificial intelligence and natural language processing.