The Hare and the Tortoise: Speed and Accuracy in Translation Retrieval

Timothy Baldwin
Department of Computer Science and Software Engineering, University of Melbourne, Victoria 3010, Australia. E-mail: tb@ldwin.net

Submitted: 28 January 2009 / Revised: 28 September 2009

Abstract. 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 using *n*-grams of various orders. To evaluate accuracy, we propose an automatic method which identifies the target-language string(s) which would lead to the optimal translation for a given input, based on analysis of the held-out translation and the current contents of the translation memory. Our results indicate that character-based indexing is superior to word-based indexing, and also that bag-of-words methods are equivalent to segment-order-sensitive methods in terms of accuracy but vastly superior in terms of retrieval speed, suggesting that word segmentation and segment order sensitivity are unnecessary luxuries for translation retrieval.

Keywords: translation memory, translation retrieval, character- and word-based indexingsegmentation

1. Introduction

Translation memories (TMs) are a well-established technology within the human and machine translation (MT) fraternities, due to the high translation precision they afford. Essentially, TMs are a list of translation records (source-language (SL) strings paired with a unique target-language (TL) translation), which the TM system accesses in suggesting a list of TL translation candidates for a given SL input (Trujillo, 1999; Somers, 2003b). In stand-alone TM systems, an arbitrary number of translation candidates of sufficient similarity with the overall input string are provided to the human user for manual manipulation in fashioning the final translation. TMs can also form a component of an MT system. TMs have been successfully combined with Statistical MT systems to boost translation accuracy over those inputs which have a good match in the TM (Marcu, 2001; Langlais and Simard, 2002). In example-based MT (EBMT), a TM is used to identify a set of translation records which correspond in part or whole with the input, and this set is then automatically adapted to form the final translation (Sato and Nagao, 1990; Nirenburg et al., 1993; Veale and Way, 1997; Somers, 2003a).

Translation retrieval is the process of retrieving a set of translation records from the TM which are calculated to be of potential use in translating the input string. At the time of retrieval, TM systems naturally have no way of accessing the TL translation of the SL input, and hence the list of TL translation candidates is determined based solely on SL similarity between the current input and translation examples within the TM. This is based on the integral assumption that SL inter-string structural and semantic similarities will be reflected in the TL translations. There is a design decision here as to just how linguistically robust the retrieval mechanism should be in determining SL similarity, and how hard the system should try to find a translation candidate in the case that no similar translation record is immediately evident. The most naive retrieval mechanism is full-string boolean match, which will naturally produce high-quality results in the case of a hit. For the greater part, however, we cannot reasonably expect to find a duplicate of the SL string in the TM, and must rely on some form of approximate match to retrieve similar but lexically differentiated translation candidates. Computational overhead tends to be at a premium with TM systems, as the user expectation is that the system should not impede the conventional translation process, but rather operate invisibly as an added extra in retrieving translation candidates and incrementally updating the TM on-stream. We are thus interested in getting the fastest possible access times, while maintaining near-optimal retrieval performance. This paper is primarily concerned with this tension.

In this paper, we choose to focus on retrieval performance over nonsegmenting languages, targeting Japanese as our SL for the greater part of the paper. Nonsegmenting languages are those which do not involve delimiters (i.e. spaces) between words, and include Japanese, Chinese and Thai. Our particular interest in nonsegmenting languages relates to the accuracy/speed trade-off, in determining the relative impact of the orthogonal parameters of segmentation, segment order and segment contiguity on retrieval performance. That is, we seek to determine whether the unavoidable overhead associated with presegmenting the input into words or morphemes is commensurate with the resultant retrieval performance. This is achieved through comparison of **character-based indexing** (where the string is naively segmented into its constituent characters or character chunks of fixed size) and **word-based indexing** (where a segmentation module is used to systematically segment the string into words), and compare their relative virtues. Similarly, we variously verify whether we gain anything by adopting more expensive segment-order-sensitive approaches, over treating each SL string as a “bag of words”, and performing a simple boolean match for each component segment. We also test the effects of explicitly modelling segment contiguity as compared to treating matches over contiguous segments identically to matches over displaced segments.

In order to evaluate different matching methods, indexing options and segment weighting schemes, we require some framework for empirical evaluation of the system output. Traditionally, TM system evaluation has taken the form of subjective evaluation of the “usefulness” of the top n ranking outputs, generally scored according to some discrete scale. We propose a fully automated, objective evaluation method, based on identification of the optimal translation candidate(s) within the TM, and comparison of these with the actual system output. By comparing the actual system output to these optimal translation candidates, we are able to empirically rate retrieval performance. This method both allows us to compare quantitatively the accuracies for different methods, and provides some insight into the degree of discrepancy in the case of nonoptimal output.

Our findings are as follows. Over a series of experiments we find that segmentation is detrimental to retrieval accuracy or, in other words, that it is better to retrieve over characters than over words. Furthermore, the bag-of-words methods we test are equivalent in translation accuracy to segment-order-sensitive methods, but superior in retrieval speed. Finally, a local model of segment contiguity is beneficial for retrieval over character-based segments, and effective to a lesser degree for word-based indexing. We thus provide clear evidence that naive (hare-like) methods are at least as good as, and in some cases, superior to more stringent (tortoise-like) retrieval methods. That is, translation retrieval is one task where “slow and supposedly steady” does not necessarily correlate with enhanced translation performance.

In the remainder of this paper, we first describe parameters we consider to affect translation retrieval accuracy, namely segmentation, segment order and segment type (Sect. 2). We then present a range of both bag-of-words and segment-order-sensitive string comparison methods in Sect. 3, and discuss methods for accelerating retrieval speed. In Sect. 4, we detail the evaluation methodology, before going on to evaluate the different methods with Japanese–English translation retrieval (Sect. 5) and English–Japanese translation retrieval (Sect. 6). Finally, we conclude the paper in Sect. 7.

2. Parameters Affecting Translation Retrieval Performance

In this section, we review three key parameters that we suggest impinge on retrieval performance, namely segmentation, segment order, and segment type.

2.1. Segmentation

Despite nonsegmenting languages such as Japanese not making use of segment delimiters, it is possible to partition off artificially a given string into constituent morphemes through the process of segmentation, using systems such as ChaSen[1] (Matsumoto et al., 1999), JUMAN[2] (Kurohashi and Nagao, 1998) and ALTJAWS.[3] The resultant segments constitute affixes, case markers and stand-alone words, which we will collectively term as "words" for the remainder of this paper. Using segmentation to divide strings into component words has the obvious advantage of clustering characters into semantic units, which in the case of logograph-based languages such as Japanese (in the form of kanji characters) and Chinese, generally disambiguates character meaning. The kanji character 弁 [ben], for example, can mean any of 'to discern/discriminate', 'to speak/argue' and 'a valve' in Japanese, but word context easily resolves such ambiguity. In this sense, our intuition is that segmented strings should produce better results than nonsegmented strings.

Looking to past research on string comparison methods for TM systems, almost all systems involving Japanese as the SL rely on segmentation (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:

- alleviate the computational overhead associated with segmentation modules;
- avoid the need to commit ourselves to a particular analysis type in the case of ambiguity or unknown words;
- avoid the need for stemming/lemmatisation;
- to a large extent get around problems related to the normalisation of lexical alternation.

All of these are issues which hit at the heart of morphological analysis.

The alleviation of the need to segment both the input and translation records in the TM (i.e. reliance on character-based indexing) could accelerate both the retrieval and TM update processes. Admittedly, for an on-line system, we can expect translation records in the TM to be presegmented, but there is still the need to segment each input, using the same segmentation method as was used to segment the TM in order to maintain consistency.

While word segmentation provides implicit disambiguation of kanji sense, treating each kanji character as an individual segment has the advantage of providing a primitive semantic class index. In technical domains in particular, it often occurs that a particular kanji occurs in only one of its semantic realisations, a fact which is borne out by our 弁 [ben] example, for which all 186 occurrences in the TM corpus used in evaluation are in the 'valve' sense.[4] Isolating kanji characters from their word contexts gives us direct access to this semantic class-type information, and gives a reasonable indication of similarity in the case of partial kanji overlap between two strings. This is particularly useful in cases where the same kanji is used in different words, which would not match under character-based indexing. With word-based indexing, we would have to rely on some form of higher-level semantic processing (e.g. linked to a thesaurus) to derive the same semantic link between the two words with kanji overlap.

Note that unless otherwise stated, all methods described in this paper 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 translation records that preserve the segment order observed in the input string will 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, posing the question of whether the cost is associated with a commensurate boost in accuracy.

As far as we are aware, there is no TM system that does not rely on word/segment/character order to some degree. Nakamura (1989) 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 noun phrases 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 stratified word order-sensitive metric 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. 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. Words in the local context of the pivot words in the original text are weighted according to a decay function over the distance from the pivot words. More recent work on translation retrieval in Japanese (Aramaki et al., 2005; Doi et al., 2005) – mainly in the context of EBMT – has tended to be syntax-based and hence operate over trees/graphs and use word-based indexing.

The greatest advantage in ignoring segment order is computational, in that we significantly reduce the search space. The arguments for segment order are largely based around the intuition that it should aid translation retrieval, a claim we look to verify. In evaluation, we analyse whether the gain in speed to bag-of-words methods outweighs any losses in retrieval accuracy over segment-order-sensitive methods. We also look to local models of segment order which partially supplement the segment order insensitivity of bag-of-words methods, thereby maintaining and potentially enhancing retrieval speed while allowing for limited segment order sensitivity.

Japanese is a relatively free-word-order language, which could possibly dilute the performance gains to sequential match methods. Having said this, while Japanese certainly has the scope for word-order variation (or strictly speaking, case-marked noun and adverb phrase permutation within a clause, as the predicate is always clause-final in written Japanese), it is rarely seen in formal or technical documents. In this sense, we can largely ignore the effects of Japanese word order variation for written language domains.

2.3. Segment Type

Japanese is made up of multiple script types, namely the native katakana and hiragana syllabries, and the kanji logographic system. Katakana and hiragana (collectively termed kana) are isomorphic and fully interreplaceable schematically, although there is a clear division of labour between the two, with katakana generally used for words of foreign origin, and hiragana for adverbs, functional words and conjugational affixes (Backhouse, 1993). Most nouns and verb and adjective stems are lexicalised in kanji, meaning that words with some kanji content are generally open class, and those made up entirely of hiragana are generally closed class.

2.4. Match stratification

One method which allows us to model multiple parameter types in parallel, is stratification of the match process (see Nirenburg et al. (1993), Planas and Furuse 1999, 2003, inter alia). Under stratified (multistratum) translation retrieval, string comparison takes place over multiple parallel levels of representation, possibly including the lexical, lemma, part-of-speech and semantic levels. By operating over multiple data types, we are able to pick up on subtle structural and conceptual correspondences not available under straight lexical comparison. This provides a valuable smoothing mechanism and a direct means of coping with effects such as lexical normalisation, by way of a simple extension to the existing matching apparatus. Clearly, however, we are faced with the same tradeoffs at each level as are discussed in this research. We thus choose to ignore stratification for the purposes of this research, and focus on comparison of the different parameter types.

3. String Comparison Methods

Due to our interest in the effects of segment order, segmentation and segment type, we must have a selection of string-comparison methods which are compatible with the various permutations of these three parameter types. We choose to look at a number of bag-of-words[5] and sequential segment matching methods which are compatible with both character-based and word-based indexing, and test different segment weighting schemes within each such system configuration.

It is possible to plot different string comparison techniques on a continuum of match stringency, as illustrated in Fig. 1. The bag-of-words, or boolean segment, matching mechanism is suggested to lie toward the computationally lightweight end of this continuum, outdone in simplicity only by full-string boolean match. Sequential segment match is computationally weightier than boolean segment match, as multiple sequential segment correspondences can exist between the two strings in question. Perhaps the most stringent matching method available is contiguous sequential segment match, where segment contiguity and sequentiality are balanced off against each other.

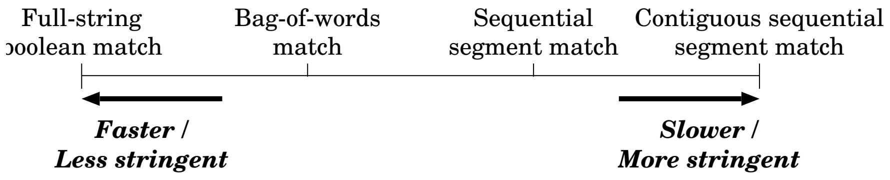

Figure 1. The continuum of match stringency vs. speed.

The particular bag-of-word approaches we target are the vector space model (VSM) (in the form of cosine similarity (Manning and Schütze, 1999, 300)) and **token intersection**, a simple ratio-based segment comparison method. For segment-order-sensitive approaches, we test two varieties of edit distance and similarity, and also **weighted sequential correspondence** (WSC).

All of the comparison techniques other than edit distance empirically describe the similarity $sim \in [0, 1]$ between two strings $S = s_1 s_2 ... s_m$ and $T = t_1 t_2 ... t_n$.[6] The different implementations of edit distance determine the distance rather than similarity between arbitrary strings in terms of the number of primitive edit operations required to transform one string into the other. For the similarity-based methods, *greater* values signify closer correspondence between the strings, whereas with edit distance, *smaller* values signify closer correspondence. It is a trivial process, however, to transform edit distance into edit similarity (see Sect. 3.2).

One feature of all comparison methods given here is that they define a partial ordering of translation records according to their relative similarity to the input. This was a deliberate design decision, and aimed at system customisab ility, i.e. the ability to adjust the system to output reliably n translations of maximum similarity to the input, or alternatively use the scores returned by the various methods to threshold over a user-defined cut-off of desired similarity. It also allows us to pinpoint a single translation record of maximum similarity (setting the issue of ties aside for the time being). 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, a selection of which are useful.

All methods are formulated to operate over an arbitrary *sweight* schema, i.e. predefined set of weights for different segment types which defines the relative reward/penalty for (mis)matches over different segments. In Sect. 5, we perform experiments to determine the relative impact of different *sweight* schema on retrieval performance.

3.1. Bag-of-Words String Comparison

Here, we describe our adopted implementations of the VSM and token intersection, the two bag-of-words methods. We illustrate the scoring process for both bag-of-words and segment-order-sensitive methods according to the task of translation retrieval for string (1), from the toy TM made up of the strings in (2) (segment delimiters given as '·'):[7]

(1) 冬·の·雨 *fuyu·no·ame* 'winter rain'

(2) a. 夏·の·雨 *natsu·no·ame* 'summer rain'
b. 雨·の·夏 *ame·no·natsu* 'a rainy summer'
c. 雨·の·冬 *ame·no·fuyu* 'a rainy winter'
d. 真·冬·の·雨 *ma·fuyu·no·ame* 'mid-winter rain'

For the purposes of illustration, we assume a unit *sweight* for all segments . Based on the TL similarity between strings (2) and (1) (i.e. the similarity in the English translations), either (2a) or (2d) would be the best translation candidates for (1).

3.1.1. *Vector Space Model*

Within our implementation of the VSM, the segment content of each string is described as a vector, made up of a single dimension for each segment token occurring within S or T. The value of each vector component is given as the weighted frequency of that token according to its *sweight* value. The string similarity of S and T is then defined as the cosine of the angle between vectors $\vec{S}$ and $\vec{T}$, respectively, calculated as in (3).

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

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

The similarity for string (1) over both strings (2a) and (2b) is calculated as $(1\times0+1\times1+1\times1+0\times1)/(\sqrt{3}\sqrt{3}) = 0.667$. Sentence (2d) fares slightly better, at a similarity of $(1\times1+1\times1+1\times1+0\times1)/(\sqrt{3}\sqrt{4}) = \sqrt{3}/2 \approx 0.866$. The greatest similarity, however, is attained for (2c), at $(1\times1+1\times1+1\times1)/(\sqrt{3}\sqrt{3}) = 3/3 = 1$. That is, VSM considers strings (1) and (2c) to be identical, where (2d) is clearly the better translation candidate. This is an example of how the simplicity of the model can lead to suboptimal outputs.

3.1.2. *Token Intersection*

The **token intersection** of S and T is defined as the cumulative intersecting frequency of tokens appearing in each of the strings, normalised according to the combined segment lengths of S and T using the Dice coefficient. Formally, this equates to (4),

$$(4) \quad tint(S,T) = \frac{2 \times \sum_{e \in S,T} \min\big(freq_S(e), freq_T(e)\big)}{len(S) + len(T)}$$

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

Note that, once again, segment order does not take any part in calculation.

The similarity for string (1) over both strings (2a) and (2b) is calculated to be $(2 \times (0 + 1 + 1 + 0))/(3 + 3) = 4/6 = 0.667$, the same result as for VSM (see Sect. 3.1.1). Sentence (2d) scores higher at $(2 \times (1 + 1 + 1 + 0))/(3 + 4) = 6/7 = 0.857$, but (2c) again produces the highest score, at a perfect $(2 \times (1 + 1 + 1))/(3 + 3) = 3/3 = 1$.

3.2. Segment Order-sensitive String Comparison

Next, we give a brief overview of the segment-order-sensitive string comparison methods utilised in this research.

3.2.1. *Edit Distance and Similarity*

The segment-based **(4-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). Conventionally, edit operations are segment equality (segments s_i and t_j are identical), segment deletion (delete segment s_i), segment insertion (insert segment t_j after s_i in string S), and segment substitution (substitute segment s_i for segment t_j). 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. This produces the desired quality that deletion and insertion operations over the same segment types are equal in cost, hence maintaining commutativity. Edit distance over all four operations at unit cost for segment deletion, insertion and substitution (i.e. $sweight = 1$ for all segment types), leads to the Levenshtein distance between a given string pair.

Segment substitution can be seen to be a compound operation, first deleting a segment at a given position in string S, and then inserting a second segment into that same position. 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 SL input.

```
set array[0][0] = 0 #init top and left boundary values
for i = 1 .. m
  set array[i][0] = array[i-1][0] + sweight(s_i)
endfor
for j = 1 .. n
  set array[0][j] = array[0][j-1] + sweight(t_j)
endfor

for i = 1 .. m #iteratively calculate distances row-wise
  for j = 1 .. n
    if s_i = t_j
     set array[i][j] = min(array[i-1][j] + sweight(s_i),
                          array[i][j-1] + sweight(t_j),
                          array[i-1][j-1])
    endif
    else
      set array[i][j] =
            min(array[i-1][j] + sweight(s_i),
                array[i][j-1] + sweight(t_j),
                array[i-1][j-1] +
                        max(sweight(s_i),sweight(t_j)))
    endelse
  endfor
endfor

return array[m][n] #return overall edit distance
```

Figure 2. The 4-operation edit distance algorithm

Additionally, we are able to distinguish between segment substitutions for strings of equal length, and segment deletions/insertions for strings of differing length. We test whether this enhances retrieval accuracy, by evaluating both 4-operation edit distance and 3-operation edit distance (with the missing operation being substitution).

Despite the high match stringency of edit distance, it has benefited from extensive optimisation (Wagner and Fischer, 1974; Masek and Paterson, 1980; Kukich, 1992), culminating in efficient dynamic programming (DP) algorithms. One particularly accessible algorithm which has gained widespread acceptance is that of Wagner and Fischer (1974), which is an implementation of 4-operation edit distance and runs in $O(mn)$ time (as do 3-operation edit distance and WSC), where m and n are the segment lengths of the target strings. The algorithm can be formalised as detailed in Fig. 2, for target strings $S = s_1 s_2 ... s_m$ and $T = t_1 t_2 ... t_n$. In its presented form, the algorithm is applicable to any *sweight* schema made up of nonnegative values.

As a distance, smaller values indicate greater similarity for edit distance, and identical strings have edit distance 0. It is possible to normalise operation edit distance $edist_{4op}$ over the lengths $len(S)$ and $len(T)$ of target strings S and T, and further convert the normalised edit distance $edist^*{}_{4op}$ into **4-operation edit similarity** sim_{4op} by way of the transformation (5).

(5) $$sim_{4op}(S,T) = 1 - edist^*{}_{4op}(S,T) = 1 - \frac{edist_{4op}(S,T)}{\max(len(S),len(T))}$$

To return to our example strings from Sect. 3.1.1, 4-operation edit distance generates a distance of 1 (and similarity of 0.667) for (2a), a distance of 1 (and similarity of 0.750) for (2d), and a distance of 2 (and similarity of 0.333) for (2b) and (2c). For the first time, therefore, we are able to produce the desired result of (2a) and (2d) being most similar to (1) (or in the case of 4-operation edit similarity, (2d) providing the closest match), due to the segment-order-sensitivity of this method. Note that the edit distances of these two strings are identical, despite them being due to a substitution in the first case and a deletion in the second. With 4-operation edit similarity, on the other hand, normalisation according to the segment lengths of the two strings discriminates between these two cases. In this sense, normalisation of 4-operation edit distance into a scaled similarity offers some means for discrimination between segment substitution and deletion/insertion.

The simplified **3-operation edit distance** algorithm differs from the 4-operation edit distance algorithm presented in Fig. 2 in one line only: line 19 is removed. That is, the third element of the min operator in the case of a segment mismatch is removed, to disallow substitutions. Again, as presented, the algorithm can operate over any *sweight* schema.

As for 4-operation edit distance, it is possible to normalise the final 3-operation edit distance $edist_{3op}$ over the lengths $len(S)$ and $len(T)$ of target strings S and T, and use this to generate **3-operation edit similarity** sim_{3op} as in (6).

(6) $$sim_{3op}(S,T) = 1 - edist^*{}_{3op}(S,T) = 1 - \frac{edist_{3op}(S,T)}{len(S)+len(T)}$$

3-operation edit similarity computed in this fashion is identical to the "sequential correspondence" method of Baldwin and Tanaka (2000).

To return to our example strings from Sect. 3.1.1, 3-operation edit distance generates a distance of 1 (and similarity of 0.857) for (2d), a distance of 2 (and similarity of 0.667) for (2a), and a distance of 4 (and similarity of 0.333) for both (2b) and (2c). Here, for both edit distance and similarity, (2d) is a clear winner over (2a), which is better than each of (2b) and (2c). This mirrors exactly our ranking of translation utility of the four strings.

3.2.2. *Weighted Sequential Correspondence*

Weighted sequential correspondence (WSC) (originally proposed in Baldwin and Tanaka (2000)) – the last of the segment-order-sensitive methods – goes one step further than edit distance in analysing not only segment sequentiality, but also the contiguity of matching segments. Given the input $\alpha_1\alpha_2\alpha_3\alpha_4$, the edit distance and similarity methods would suggest $\alpha_1\beta_1\alpha_2\beta_2\alpha_3\beta_3\alpha_4$ and $\alpha_1\alpha_2\alpha_3\alpha_4\beta_1\beta_2\beta_3$ as equally close matches, for example, despite the second of these being more likely to produce a translation at least partially resembling that of the input string.

We get around this by associating an incremental weight (orthogonal to our *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.[8] This weighting up of contiguous matches is facilitated through the DP algorithm given in Fig. 3. The essence of the method is to use array `m` to keep track of the truncated number of contiguous segment matches leading up to the current segment (calculated according to $\min(k, Max)$), and weight the current segment match based on the match contiguity score pre-stored in `m[i][j]`.

The final similarity is determined as in (7),

$$(7) \quad wseq^*(S,T) = \frac{2 \times wseq(S,T)}{len(S) + len(T)}$$

where $wseq(S,T)$ is the maximum WSC between S and T, as determined by the algorithm in Fig. 3.

One key area in which our algorithm differs from that of Sato (1992) is that we reset `m[i][j]` to zero in the case of a nonmatching segment, whereas Sato carries the degree of contiguity over from `m[i-1][j-1]` (in the manner indicated in line 21 of the algorithm description). Thus, if we have a contiguously matching substring of length Max or more early on in the match process, any subsequent segment-level matches will be treated as if they were contiguous with the original matching substring. In our implementation, we reset the match counter to zero whenever a nonmatching segment is encountered, and thereby measure the true contiguity of each matching substring. In evaluation, we analyse the implications of this difference on retrieval accuracy.

```
for i = 0 .. m #init top and left boundary values
  set m[i][0] = 0
  set array[i][0] = 0
endfor
for j = 1 .. n
  set m[0][j] = 0
  set array[0][j] = 0
endfor

for i = 1 .. m #iteratively calculate scores row-wise
  for j = 1 .. n
    if s_i = t_j
      set m[i][j] = min(m[i-1][j-1]+1,MAX)
      set array[i][j] =
             max(array[i-1][j], array[i][j-1],
                 array[i-1][j-1] + m[i][j]*sweight(s_i))
    endif
    else
      set m[i][j] = 0
      set m[i][j] = m[i-1][j-1]  #Sato(92)
      set array[i][j] = max(array[i-1][j],
                            array[i][j-1],
                            array[i-1][j-1])
    endelse
  endfor
endfor

return array[m][n] #return overall correspondence score
```

Figure 3. The substring match scoring algorithm for WSC

Turning once again to our example TM, the similarity for (2a) is $(2\times3)/(6+6) = 0.500$, the similarity for (2d) is $(2\times6)/(6+10) = 0.375$, and that for (2b) and (2c) is $(2\times1)/(6+6) = 0.167$. In this way, weighted sequential correspondence generates the same ranking of translation records as 3-operation edit distance.

3.3. Enhanced segment order sensitivity: n-gram methods

All of our string comparison methods can be run over any type of ordered, segmented data, noting that the segment order is ignored by the bag-of-words methods. By combining adjacent segments together into n-grams, it is possible to generate an explicit model of local segment context. This is particularly useful for the bag-of-words methods as it provides a crude model of both segment order and contiguity, but is also valuable for the edit distance methods as it complements the implicit modelling of segment order with local segment contiguity. Here, n-grams can be fashioned either from individual characters or words, and are thus orthogonal to the two indexing paradigms.

N-gram models have seen widespread usage in NLP, information retrieval and speech recognition research, due to their easy applicability and high versatility. The main failing documented in past research has been that smaller values of n often do not produce sufficient segment discrimination and are unable to model long-distance segment dependency, whereas larger values of n quickly run into problems with data sparseness (Brown et al., 1992; Chen and Goodman, 1996; Manning and Schütze, 1999; Zhai and Lafferty, 2004). One commonly-used method to combine the relative advantages of different n-gram orders is to use some form of smoothing or back-off. In our case, rather than introducing explicit smoothing, we adjust the input to the different string comparison methods, in the form of simple unigram, pure bigram, and mixed unigram/bigram modelling. In the mixed unigram/bigram method, we interleave bigrams and unigrams to preserve the original segment ordering. From string (2a), for example, we would generate the variants (common to both character- and word-based indexing) in (8).

(8) Unigram version: 夏·の·雨
Bigram version: 夏の·の雨
Mixed unigram/bigram version: 夏·夏の·の·の雨·雨

3.4. Retrieval Speed Optimisation

In their existing forms, the bag-of-words methods are both $O(m+n)$ (linear) while the segment-order-sensitive methods are all $O(mn)$ (polynominal), where m and n are the segment lengths of the two strings being compared. In terms of the computational cost of a single comparison, therefore, the bag-of-words methods are at a distinct advantage over the segment-order-sensitive methods, an effect which is exacerbated as m and n get bigger. All is not lost, however, as we are able to employ a branch-and-bound approach to reduce the number of string comparisons required to determine the best match, under the assumption that we are after only the best-scoring match(es) and not a full ranking of all translation candidates. This is achieved through analysis of the segment length of translation candidates (segment-order-

String comparison	String length:		Halting
method	Upper bound	Lower bound	condition
4-op edit distance	$\mathit{len}(IN) + \alpha$	$\mathit{len}(IN) - \alpha$	$\beta > \alpha$
4-op edit similarity	$\frac{\mathit{len}(IN)}{\alpha}$	$\alpha \mathit{len}(IN)$	$1 - \frac{\beta}{\max(\mathit{len}(IN),\mathit{len}(TM_i))} < \alpha$
3-op edit distance	$\mathit{len}(IN) + \alpha$	$\mathit{len}(IN) - \alpha$	$\beta > \alpha$
3-op edit similarity	$\frac{(2-\alpha)\mathit{len}(IN)}{\alpha}$	$\frac{\alpha \mathit{len}(IN)}{2-\alpha}$	$1 - \frac{\beta}{\mathit{len}(IN)+\mathit{len}(TM_i)} < \alpha$
WSC	$\frac{(2-\alpha)\mathit{len}(IN)}{\alpha}$	$\frac{\alpha \mathit{len}(IN)}{2-\alpha}$	$\frac{2\beta}{\mathit{len}(IN)+\mathit{len}(TM_i)} < \alpha$

Figure 4. Bounds on string lengths and halting conditions for the different methods. Key: β is the segment overlap between the input and current translation record, α is the current best matching score, and $\mathit{len}(IN)$ and $\mathit{len}(TM_i)$ are the lengths of the input and current translation record, respectively.

sensitive methods only), and segment overlap with the input (all methods). There is also scope for minimal savings with the edit distance and similarity methods, in prematurely exiting out of suboptimal string matches. All acceleration methods discussed here are lossless.

First, by precomputing and storing the weighted segment length of each string (weighted according to both the *sweight* values and the sequentiality increment in the case of WSC), 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. While the bounds can be derived trivially from the basic scoring functions for each method, in the interests of completeness, we present the various bounds for each method in Figure 4.

Clearly, we will not be able to rely on previous matches in setting α (the score for the current best match) for the first iteration. For the (unnormalised) edit distance methods, we get around this by initialising α to $\mathit{len}(IN)$, which is equivalent to setting the empty string as the closest match. For the remaining similarity-based methods, however, we apply the bounds only after we have calculated the similarity for the first string, and can use the resultant α value. Note that α for the straight edit-distance methods is an (unnormalised) edit distance, whereas that for all other methods it is a scaled similarity in the range $[0, 1]$.

It is important to realise that all bounds are static for a given α and $\mathit{len}(IN)$, and need be recomputed only when the current best match is improved upon. Note also that the string lengths are fixed and can be cached for as long as the *sweight* schema remains unchanged. For static segment weighting methods, therefore, this dependence on length poses no additional burden, and the database of translation record lengths can be updated incrementally whenever additions of translation records are made.

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 (Moffat and Zobel, 1996). Despite the minimalist nature of the inverted file (for example, it does not contain information about segment order[9]), it allows us to exclude all translation records with no common segment component with the input, from the search process; and it provides an immediate indication of the quality of match possible with each translation record overlapping in segment content with the input. All of this is possible within a compact, easy-to-maintain format.

For the bag-of-words approaches, the inverted file supplies data to plug directly into the scoring functions, in combination with the weight for each segment. For the segment-order-sensitive approaches, on the other hand, we determine the optimal match through the combination of the scoring function and the segment overlap between each translation record and the input; this is attained only in the case that the overlapping segments occur in identical order in the two strings. The method for computing the optimal score for each translation record differs according to the string-comparison method, and is given on the left-hand side of the various inequalities in the “halting condition” column of Figure 4.

In this way, we can order translation records according to the degree of match potentiality, and halt when the highest-ranking translation record yet to be processed has an optimal score inferior to the best-scoring match to that point, in the manner presented in the “halting condition” column of Figure 4.

While this method reduces the seach space considerably for most methods, there is a significant computational overhead to establishing an optimal score for each translation record sharing some segment content with the input, and subsequently sorting the translation records according to their optimal score. Over the small-scale TMs used in evaluation, this overhead was not probitively expensive, but it could prove too great for larger-scale TMs (see Sect. 5.5 for further discussion of this point). One lossy method of reducing the burden of preprocessing, could be to do a beam search over the translation records with the greatest segment overlap with the input (irrespective of the correlation of this overlap to the method-specific scoring system), thus partially eliminating expensive floating-point operations. Alternatively, a beam search could be employed over the n segments with the highest *sweight* values, reducing the search space further with little danger of disallowing optimal matches. By imposing tighter (but lossy) string length bounds and halting conditions, further reductions in the search space would be possible.

Local optimisations to each method are also possible. For the edit-distance methods, for example, we can build into the algorithms a mechanism to determine iteratively whether the current translation record has the potential to better the current best match. This is achieved by caching the minimum edit distance for each substring pair `i` and `j`, along the row `array[i][0..j]` and column `array[0..i][j]`. If the combined minimum edit distance along these edges exceeds the current optimum edit distance, then we halt the matching process. Our motivation here is that all edit operations are additive, and it is hence not possible for any reduction in the minimum local edit distance to occur over the remaining components of the target strings. If a local suboptimal match is detected, then we exit the matching process, and return an arbitrarily high edit distance (at least as high as the edit distance to that point), to ensure that the TM system discards that translation candidate. More localised optimisation is also possible using this same basic method, in triangularising the edit-distance matrix to avoid local computations over regions which have no chance of contributing to an overall best analysis – see Planas and Furuse (2000, 2003) for a more thorough description of this general method.

4. Evaluation Specifications

In this section, we describe the evaluation methodology. In proceeding sections, we go on to apply this methodology in evaluating the various string comparison methods, under the two indexing algorithms and in combination with the various models of segment contiguity. We additionally test various *sweight* schemata, based on character type.

4.1. Details of the Dataset

As our main dataset, we used 3,033 unique Japanese–English translation records relating to the servicing of construction machinery.[10] In the dataset, translation records vary in size from single-word glossary terms to multiple-sentence strings, at an average Japanese segment length of 14.4 and character length of 27.7, and an average English word length of 13.3. Note that our dataset constitutes a controlled language, that is, a given word will tend to be translated identically when used in a particular sense (i.e. there is no literary flair), and only a limited range of syntactic constructions are employed. We discuss the impact of this fact on retrieval performance in Sect. 4.3.1.

In Sect. 5.5, we use a second dataset of expanded size, from the economic domain. Details of the dataset and its use can be found there.

The translation records making up the dataset were generated from a glossary of technical terms and a collection of technical reports. Each line in the original glossary consisted of a Japanese technical term and a list of corresponding translations, from which a single translation record was generated by taking the first translation candidate. The technical reports posed a more serious alignment problem, as descriptions were given in full sentence form, and the Japanese and English versions were found in independent files. Fortunately, individual technical reports were labelled with the same index across languages, and the same logical structure was preserved between corresponding technical reports. It was thus possible to align reports at the logical segment level ("Title", "Problem description", "Solution", etc.). The number of sentences in aligned logical segments was then counted, and in the case of an identical count, individual sentences were extracted and sequentially aligned between Japanese and English (similarly to the approach of Veale and Way (1997)). Failing this and in the case that one of the languages contained a single sentence, the multiple sentences in the opposing language were aligned to that single sentence. Only in the case that a divergent, nonsingular number of sentences was obtained for the two languages, was manual alignment called upon. We did not attempt to align sentences automatically in this final case, as the number of such cases was minimal (around 50). Instead, we manually aligned this residue, a process which was completed within one person hour. Naturally, we recognise the worth of automatic alignment methods for larger texts with less logical structure, and refer the reader to the work of Gale and Church (1993), Chen (1993), Wu (1994), Véronis (2000), *inter alia*. Such methods would prove their worth in on-line applications where TM updates are fully automated.

Demarcation of translation records was assumed in evaluation, and no further effort was made to subdivide partitions. For real-world TM systems, the automation of this process comprises an important component of the overall system, integrally linked to translation retrieval. While acknowledging the importance of this step and its interaction with retrieval performance (Nirenburg et al., 1993), we choose to sidestep it for the purposes of this paper, and leave it for future research.

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 and Nagao, 1998) was also used as a control. A third system, ALTJAWS, was used to test the effects of lexical normalisation on retrieval accuracy. For all three segmenters, no attempt was made to postedit the segmented output, in interests of maintaining consistency in the data. We do, however, rate the segmentation performance of the different systems in Sect. 5.2 in order to determine the source of any disparity in retrieval performance.

4.2. Semi-stratified Cross Validation

Retrieval accuracy was determined by way of 10-fold semistratified cross validation over the dataset. As part of this, all Japanese strings of length 5 characters or less (a total of 531 translation records) were extracted from the dataset, leaving a residue of 2,502 translation records over which cross validation was performed. Our motivation in setting aside the shorter strings was that they originated from the glossary, and are not realistic inputs for translation. That is not to say that they could not provide high-quality partial matches for other translation records, however, and they were included 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 and other evaluative data is appropriately sampled over the n data configurations.

For our purposes, the test data equates to a set of held-out inputs, and the training data (plus the glossary) the TM. As our dataset is not preclassified 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 SL segment lengths of the translation records. The same partitioning of data was used throughout evaluation unless otherwise specified.

4.3. Evaluation of the Output

We evaluate the appropriateness of the translation candidate(s) returned for a given input by taking the held-out translation for the input as the model translation and analysing whether the translation candidate(s) are optimally close to the model translation, given the current TM content. By treating the held-out translation for each input as the model translation, it is possible to apply the same string comparison methods as are used in translation retrieval, to determine that set of translations most closely corresponding to the actual translation. Determination of the set of "optimal" translations, therefore, consists of using the held-out translation as the input in searching through the

TL component of the TM, and using an arbitrary string comparison method to judge which translations in the TM most closely resemble the model translation. That is, translation "optimality" is defined numerically according to TL similarity or distance from the model translation, as determined by a given string comparison method.

4.3.1. *Methodology*

First, we use a string comparison method to determine the closest-matching translation record(s) to the (unique) model translation. In this, we consider only those translation records in the training TM associated with each input. This methodology correlates to pinpointing the "most useful" translation(s) the TM system could possibly return from the current TM, given that this is the only translation data available to the TM system.

Having identified the translation candidates most similar to the model translation, we next consider whether the effort required to transform any one of these candidates into the model translation will be greater than the effort required to translate the input from scratch, that is whether their "translation utility" will be sufficiently high. This is achieved in a model-specific manner, either by thresholding over a static value (see Sect. 4.3.2), or in the case of the edit-distance methods, by thresholding over the edit distance between the input and the empty string (i.e. the segment length of the input). The set of optimal translation candidates for a given TM and model translation IN^{TL} can thus be formalised as in (9) in the case of a distance-based string comparison method.[11]

$$(9)\quad \begin{aligned}&S^{TL}|\langle S^{SL}, S^{TL}\rangle \in TM \cup \{\langle \epsilon^{SL}, \epsilon^{TL}\rangle\}\wedge\\ &\mathit{dist}(S^{TL}, IN^{TL}) \leq \mathit{len}(IN^{TL})\wedge\\ &\forall \langle T^{SL}, T^{TL}\rangle \in TM - \langle S^{SL}, S^{TL}\rangle :\\ &\mathit{dist}(S^{TL}, IN^{TL}) \leq \mathit{dist}(T^{TL}, IN^{TL})\}\end{aligned}$$

In the case that no translation candidate is below the translation utility cut-off, we return the empty string; in the case that the optimal translation candidate(s) score at the same level as the cut-off, the empty string is returned along with the best-scoring translation candidate(s).

Having established the optimal TM system output for each input, we proceed to ascertain whether the actual system output coincides with one of the optimal outputs, and rate the accuracy of each method according to the proportion of optimal outputs. In order to factor out the effects of differing degrees of multiplicity of output, we break ties randomly. This guarantees a unique translation output. Our primary motivation in this was to make the final results for each method directly comparable. This differs from the methodology of Baldwin and Tanaka

(2000) who consider the system to be "correct" if an optimal translation candidate is contained in the potentially multiple set of top-ranking outputs.

The choice of string comparison method used to evaluate the TL output is independent of the choice of string comparison method used in translation retrieval. So as to filter out any bias towards a given string comparison method in translation retrieval, we determine the optimal set of translation candidates via: (a) 3-operation edit distance (operating over bigrams); and (b) WSC (operating over unigrams). We calculate the accuracy of a given method relative to each of the evaluation sets returned by the two methods, and average them to estimate the final translation accuracy.

We additionally evaluate the margin of error in the case that the translation output is nonoptimal, i.e. that it does not correspond to one of the optimal translation candidates. This is achieved by determining the minimum distance/maximum similarity between the unique output and the set of optimal outputs, hence modelling the relative degree of error in suboptimal outputs.

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 utility 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 (Nakamura, 1989; Sumita and Tsutsumi, 1991; Sato, 1992). Perhaps the closest evaluation attempts to what we propose are those of Planas and Furuse (1999), who modelled "translation usability" by the ability to generate the model translation from a given translation candidate by editing less than half the component words; and Nirenburg et al. (1993), who calculated 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 target language utility, but is dependent on the competence of the translator editing the TM system output, and not automated to the degree our method is.

One drawback to our evaluation methodology is that we assume a unique model translation for each input, whereas in fact multiple translations of equivalent quality, differentiated only stylistically, may exist for each SL string. This is not a significant concern in the experiments described here, as our primary dataset of technical field reports is in a relatively controlled language with very few occurrences of competing translations for a given SL string. For free text domains, the allowance of multiple model translations would go some way towards modelling stylistic variation, although this would leave the issue of how the TM

Language	Segment type	*sweight*
English	punctuation	0
	stop words	0
	other words	1
Japanese	punctuation	0
	other segments	1

Figure 5. The *sweight* schema used for TL evaluation of English (top half) and Japanese (bottom half)

system should choose between multiple model translations for a given SL string, in positing translation candidates from the TM.

An alternative approach to evaluation would be to adopt automatic MT evaluation metrics such as Bleu (Papineni et al., 2002) and METEOR (Banerjee and Lavie, 2005). A major consideration in deciding against this was that for the majority of our inputs we had only one reference translation, and automatic MT evaluation metrics rely on the availability of multiple reference translations to operate reliably (Culy and Riehemann, 2003). If a large-scale Japanese–English dataset with multiple reference translations were to become available in the future, we would hope to perform an empirical comparison of our evaluation methodology with mainstream MT evaluation metrics.

4.3.2. *Parameter settings*

Both of the string comparison methods used in evaluation (namely 3-operation edit distance and WSC) rely on the definition of an *sweight* schema. For English, we adopted the TL *sweight* schema presented in the top half of Figure 5. Stop words are defined as those contained within the SMART (Salton, 1971) stop-word list.[12] For Japanese, we adopted the TL *sweight* schema presented in the bottom half of Figure 5.

The thresholds on "translation utility" are those given in Figure 6, where IN is the input string, and *len* is the conventional segment length operator.

5. Japanese–English Translation Retrieval Results

In this section, we explore a number of questions through a series of Japanese–English translation retrieval experiments. Namely, we evalu-

Comparison method	Threshold
Vector space model	0.5
Token intersection	0.4
3-operation edit distance	$len(IN)$
3-operation edit similarity	0.4
4-operation edit distance	$len(IN)$
4-operation edit similarity	0.4
WSC	0.2
Sato (1992)	0.2

Figure 6. Thresholds on translation utility for each of the string comparison methods

ate the different string comparison methods over character- and word-based indexing for varying n-gram orders. We then go on to investigate the impact of segmentation accuracy, different *sweight* settings and kanji on retrieval performance. Finally, we determine the scalability of the different methods over TMs of increasing size.

5.1. Experiment I: basic results

First, we look at Japanese–English translation retrieval for the basic SL *sweight* schema given in the lower half of Figure 5.

5.1.1. *Results*

The results for the different string comparison methods under character-based and word-based indexing are given in Figures 7 and 8, respectively. In each table, the bag-of words methods are partitioned off from the segment-order-sensitive methods, "Sato92" to the Sato (1992) variant of WSC. The bag-of-words methods and edit-distance and similarity methods were variously tested under unigram, bigram, and mixed unigram/bigram models of segment contiguity,[13] whereas WSC and Sato92 were tested only with the unigram model, due to their implicit modelling of segment contiguity. "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[14] advantage over the corresponding word-based indexing accuracy (same string comparison method and n-gram model), and the underlined figure for each indexing paradigm indicates the highest achieved accuracy. "Nonopt dist" refers to the mean minimum 3-operation edit distance between the translation candidate and optimal translation(s) in the

	Method	Accuracy	Nonopt edit dist	Unique outputs	Av'ge time
Unigram	VSM	53.35	10.75	0.97	1.73
	Token int	53.59	10.12	0.94	2.37
	3-op edit dist	**55.45**	8.00	0.81	2.98
	3-op edit sim	57.87	8.61	0.95	10.55
	4-op edit dist	**50.76**	9.50	0.79	17.40
	4-op edit sim	56.49	8.52	0.90	26.28
	WSC	56.73	9.18	0.96	127.56
	Sato92	55.15	9.58	0.95	132.77
Bigram	VSM	**60.41**	7.95	0.97	0.49
	Token int	**60.85**	7.88	0.95	0.62
	3-op edit dist	**58.17**	7.61	0.82	0.76
	3-op edit sim	**61.09**	7.83	0.96	0.97
	4-op edit dist	**51.66**	9.32	0.77	1.67
	4-op edit sim	**59.55**	7.82	0.94	1.98
Mixed	VSM	58.95	9.24	0.97	2.11
	Token int	59.09	8.63	0.97	2.96
	3-op edit dist	57.65	7.77	0.85	3.50
	3-op edit sim	**60.57**	8.15	0.96	9.03
	4-op edit dist	**51.42**	9.38	0.84	20.89
	4-op edit sim	**59.05**	8.07	0.95	32.22

Figure 7. Results for the different comparison methods under **character**-based indexing, using a unigram, bigram and mixed unigram/bigram segment contiguity model, as evaluated using the combined 3-operation edit distance and WSC methods

case of the translation candidate being nonoptimal (the edit distance is not calculated in the case of an optimal translation output); as for accuracy, the best-achieved nonoptimal edit distance for each indexing paradigm is underlined. “Unique outputs” describes the proportion of inputs for which a unique translation candidate was produced. “Av'ge time” describes the average time taken to determine the translation candidate(s) for a single input, relative to the time taken for word unigram-based 3-operation edit-distance retrieval. We use this particular system configuration, as timed over the construction machinery dataset, as our touchstone for time calculation throughout this paper.

Perhaps the most striking result is that character-based indexing produces a superior match accuracy to word-based indexing for *all* comparison methods operating over *all* n-gram models of local context, to a level of statistical significance in all cases except for the VSM and

	Method	Accuracy		Nonopt edit dist	Unique outputs	Av'ge time
Unigram	VSM	53.89	(+1.0%)	10.08	0.93	<u>0.66</u>
	Token int	53.83	(+0.4%)	9.60	0.89	0.91
	3-op edit dist	52.38	(−5.5%)	<u>8.51</u>	0.72	1.00
	3-op edit sim	<u>55.45</u>	(−4.2%)	8.75	0.90	1.75
	4-op edit dist	47.72	(−6.0%)	10.30	0.66	3.09
	4-op edit sim	53.71	(−4.9%)	8.93	0.84	4.04
	WSC	54.23	(−4.4%)	9.00	0.93	28.99
	Sato92	52.97	(−3.9%)	9.72	0.92	29.61
Bigram	VSM	54.41	(−9.9%)	<u>7.57</u>	0.95	<u>0.26</u>
	Token int	54.39	(−10.6%)	7.61	0.93	0.31
	3-op edit dist	53.01	(−8.9%)	7.75	0.83	0.34
	3-op edit sim	<u>54.51</u>	(−10.8%)	7.61	0.93	0.38
	4-op edit dist	47.60	(−7.9%)	10.16	0.70	0.59
	4-op edit sim	54.23	(−8.9%)	7.74	0.92	0.55
Mixed	VSM	56.65	(−3.9%)	<u>7.95</u>	0.96	<u>0.90</u>
	Token int	56.65	(−4.1%)	8.07	0.94	1.16
	3-op edit dist	55.93	(−3.0%)	8.08	0.83	1.32
	3-op edit sim	<u>56.69</u>	(−6.4%)	8.00	0.94	1.99
	4-op edit dist	47.82	(−7.0%)	10.30	0.73	3.65
	4-op edit sim	54.91	(−7.0%)	8.23	0.91	4.68

Figure 8. Results for the different comparison methods under **word**-based indexing, using a unigram, bigram, and mixed unigram/bigram segment contiguity model, as evaluated using the combined 3-operation edit distance and WSC methods

token intersection, when coupled with a unigram model. The relative gain in accuracy due to character-based indexing, averaged over the different methods and n-gram models, is 9.2%.

3-operation edit distance outperforms all other methods for both character- and word-based indexing, peaking at around 61% for character bigrams. There is a marked drop-off in accuracy between 3-operation edit distance and 3-operation edit similarity for both character- and word-based indexing, which was found to be statistically significant in most cases. Similarly, 3-operation edit distance has a clear advantage over 4-operation edit distance, supporting our prediction that the introduction of the substitution operator with 4-operation edit distance leads to a loss of discriminatory granularity. Interestingly, 4-operation edit similarity fared slightly better than 4-operation edit distance, due to the normalisation effect observed in Sect. 3.2. WSC and the Sato92 method performed at roughly the same level as 4-operation edit similarity.

An equally important component of evaluation is nonoptimal edit distance, i.e. the magnitude of error in the case of a nonoptimal output. Here again, 3-operation edit distance was found to err most conservatively in all cases. We can thus state that, of all the methods tested, 3-operation edit distance is superior in terms of both raw accuracy and the margin of error in the case of a nonoptimal output, within the confines of the given evaluation framework. Looking to the other methods, the bag-of-words methods tended to produce a high nonoptimal edit distance when operating over unigrams, but were equal to or better than the nonoptimal edit distance of the segment-order-sensitive methods for the bigram and mixed bigram models.

Of the n-gram orders tested, bigrams were found to perform best under character-based indexing, nosing out mixed unigrams/bigrams, and mixed unigrams/bigrams were found to be superior to straight unigrams and bigrams for word-based indexing. Bigrams produced a marked acceleration in retrieval time (i.e. they reduce the number of translation records required to be processed before settling on the final translation candidate set), whereas mixed unigrams/bigrams were marginally slower than simple unigrams (predictably given the extended lengths of the two resulting strings, m and n).

All similarity-based methods were able to produce a unique translation candidate over 90% of the time, while the edit distance methods were less successful at predicting a single translation candidate, returning unique translation candidates around 85% of the time. Note that these figures are simply a reflection of the discriminatory potential of the different methods, and that a unique translation candidate was randomly selected from the set of outputs for use in evaluating accuracy and edit discrepancy.

Looking to the relative speeds of the different methods, word-based indexing was found to be faster than character-based indexing across the board for all n-gram orders, largely because the number of character n-grams per string is always going to be greater than or equal to the number of word n-grams. Additionally, word-based indexing produces greater segment discrimination than character-based indexing for the same n-gram order, by way of increasing the number of segment types. This, coupled with the acceleration methods described in Sect. 3.4, reduces the number of string comparisons required to settle on the final translation candidate set (i.e. prunes the string search space). Note that the running times include the time required to segment the input on-line. Due to the smaller number of segments under word-based indexing, the overhead for doing the segmentation is to some degree made up for by the gain in computation for the string comparison relative to characters. While recognising that segmentation has a part to play in shortening retrieval times, character bigrams were found to be faster than word unigrams. Word bigrams were faster again, but were associated with a sharp drop in retrieval accuracy.

5.1.2. *Summary*

Based on the results of this experiment, we judge bigrams to be the best segment of the n-gram orders for character-based indexing, and mixed unigrams/bigrams to be optimal for word-based indexing, and for the remainder of this paper, present only these two sets of results. In keeping with our comments on the incompatibility of WSC and Sato92 with anything other than a unigram model, we continue to run these two methods only on basic unigram data, but present the results for other methods with both (character) bigrams and mixed (word) unigrams/bigrams.

5.2. Experiment II: Segmentation accuracy

5.2.1. *Background*

In experiment I, 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. In this section, we look to clarify such issues primarily by evaluating the retrieval and segmentation performance of the ChaSen, JUMAN and ALTJAWS systems and considering the ramifications of system errors on overall retrieval accuracy. In conjunction with this, we test whether lexical normalisation has any impact on retrieval performance, using ALTJAWS to convert each segment into canonical form.

First, we used JUMAN to segment the original set of 3,033 translation records, and evaluated the resultant dataset in the exact same manner as for the ChaSen output. Note that in semistratified cross validation, a slightly different partition composition was generated for JUMAN than for ChaSen, due to relative differences in the number of segments returned for each string. Similarly, we ran a development version of ALTJAWS over the 3,033 translation records to produce two datasets, the first simply segmented and the second both segmented and lexically normalised.[15] Lexical normalisation took the form of taking each segment and converting it into canonical form according to the ALTJAWS analysis.[16] Instances where the canonical form differed from

	Method	Accuracy	Nonopt edit dist	Unique outputs	Av'ge time
Word-based	VSM	56.94 (+0.5%)	<u>7.84</u>	0.96	<u>0.76</u>
	Token int	<u>57.45</u> (+1.4%)	7.89	0.95	1.03
	3-op edit dist	56.19 (+0.5%)	7.92	0.84	1.19
	3-op edit sim	57.28 (+1.0%)	7.88	0.95	1.61
	4-op edit dist	47.64 (−0.4%)	10.18	0.73	2.76
	4-op edit sim	55.94 (+1.9%)	7.97	0.92	3.52
	WSC	55.04 (+1.5%)	8.64	0.92	23.50
	Sato92	53.58 (+1.1%)	9.07	0.91	23.39

Figure 9. Results for data segmented with JUMAN, with word-based indexing

the original orthography were predominantly verbs in nonregular form (e.g. 交換して [*kōkan-shite*] 'replace' ⇒ 交換する *kōkan-suru*), but also included 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 kanji realisations for the same basic sense (e.g. 充分 [*jūbun*] 'sufficient' ⇒ 十分). As for the JUMAN output, we evaluated the normalised output of ALTJAWS using semistratified cross validation.

5.2.2. *Results with the different segmenters*

The results for JUMAN are presented in Figure 9, and those for ALTJAWS with and without lexical normalisation are presented in Figure 10. We present only the word-based indexing results, as the results for character-based indexing are equivalent to those for ChaSen presented in Figure 10.[17] These results should be compared with those presented in Figure 8 for ChaSen.

Looking first to the results for JUMAN, there is a slight but appreciable gain in accuracy over ChaSen, for all string comparison methods other than 4-operation edit distance. While this step-up in performance was never found to be statistically significant, it was consistent on the whole, suggesting JUMAN as an equally plausible candidate segmentation module to ChaSen. We consider reasons for this performance discrepancy in Sect. 5.2.3.

With ALTJAWS, also, a consistent gain in performance is evident with simple segmentation, the degree of which is considerably higher than for JUMAN. The addition of lexical normalisation enhances this effect marginally, although not to a degree where we are able to draw any hard conclusions as to the effectiveness of lexical normalisation in itself. One area in which ALTJAWS has an edge over ChaSen and JUMAN

	Method	Accuracy	Nonopt edit dist	Unique outputs	Ave. time
Without normal'n	VSM	**58.69** (+3.6%)	7.61	0.96	<u>0.60</u>
	Token int	<u>**59.33**</u> (+4.7%)	7.68	0.95	0.87
	3-op edit dist	58.31 (+4.3%)	<u>7.61</u>	0.82	1.03
	3-op edit sim	**59.09** (+4.2%)	7.66	0.95	1.39
	4-op edit dist	49.44 (+3.4%)	9.68	0.72	2.43
	4-op edit sim	**57.59** (+4.9%)	7.77	0.93	2.70
	WSC	**57.24** (+5.5%)	8.35	0.93	20.43
	Sato92	**55.96** (+5.6%)	8.83	0.92	19.43
With normal'n	VSM	58.86 (+3.9%)	7.85	0.96	<u>0.57</u>
	Token int	<u>**59.60**</u> (+5.2%)	7.86	0.94	0.74
	3-op edit dist	59.06 (+5.6%)	<u>7.77</u>	0.83	0.85
	3-op edit sim	59.29 (+4.6%)	7.83	0.94	1.28
	4-op edit dist	**50.83** (+6.3%)	9.85	0.73	2.20
	4-op edit sim	**58.09** (+5.8%)	7.94	0.93	2.84
	WSC	**57.41** (+5.8%)	8.54	0.93	16.25
	Sato92	**56.52** (+6.7%)	9.05	0.92	17.25

Figure 10. Results for data segmented and optionally lexically normalised with ALTJAWS, with word-based indexing

is the handling of compound nouns (and particularly katakana words), an effect we analyse in more detail in Sect. 5.2.3.

In sum, we can state that the choice of segmentation system does have a modest impact on retrieval accuracy, and also that the effects of lexical normalisation are highly localised. Our segmentation module of choice, ChaSen, was found to be the worst performer of all three systems tested, prompting doubt as to our commitment to it. In response to this, we emphasise that the main interest of this paper is in verifying the relative merits of different methods and procedures in translation retrieval, and not in generating the absolute best integrated system possible. ChaSen simply provides a point of reference for this purpose. It is, however, worthwhile delving deeper into the expected impact of the segmentation module on final translation output, and in the following, we look to quantify the relationship between retrieval performance and segmentation accuracy.

5.2.3. *Analysis of the relation between segmenter performance and translation retrieval performance*

In the next step of evaluation, we took a random sample of 200 translation records from the original dataset, and ran each of ChaSen, JUMAN and ALTJAWS over the Japanese component of each translation record. We then manually evaluated the output in terms of segment precision and segment recall, defined respectively as in (10).

(10) a. $$\text{Segment precision} = \frac{\text{Number of correct segments}}{\text{Total number of segments}}$$

b. $$\text{Segment recall} = \frac{\text{Number of correct segments}}{\text{Total number of segments expected}}$$

We further calculated the average character and segment count per sentence, and average sentence segmentation accuracy (i.e. the proportion of sentences for which an overall correct analysis was obtained).

One slight complication in evaluating the output of the three systems is that they adopt incongruent models of conjugation. ChaSen and ALTJAWS consistently partition off the stem of (both conjugating and nonconjugating) verbs and adjectives, and also treat any verbal morphemes (e.g. the passive morpheme) and conjugating affixes as individual segments. JUMAN, on the other hand, segments conjugating lexemes only in the case that they are demarcated by verbal morphemes, choosing to consider cases of simple inflection as a single segment. There are also subtle differences between ChaSen and ALTJAWS, such as ChaSen tending to separate off each individual auxiliary verb, but ALTJAWS bundling complex sequences of auxiliaries into a single segment. The verb complex していた [*shite-ita*] 'DO/prog/past'was doing, for example, would be divided into three segments (the stem *shi* 'do', the stem of the progressive auxiliary *te-i*, and the past tense auxiliary *ta*) by ChaSen, two segments (the stem *shi* 'do' and a single segment *te-ita* for the two auxiliaries) by ALTJAWS, and a single segment *shite-ita* by JUMAN.

In evaluating the accuracy of the different systems, therefore, we had to make allowance for the idiosyncrasies of the particular segmentation paradigm adopted by each system. In practice, all errors for the three systems occurred in the segmentation of noun complexes, and slight variance in the treatment of single-entry verb complexes was permitted (e.g. both *sashi-kaeru* and *sashikaeru* were judged to be acceptable segmentations for 差し替える [*sashikaeru*] 'to swap over').

Any fall-off in segmentation performance should thus be seen as a reflection of the ability of each system to handle noun-phrase segmentation, and in particular the robustness of the system over compound nouns and unknown words.

A performance breakdown for JUMAN, ChaSen and ALTJAWS is presented in Figure 11, where segment precision and recall are defined as in (10), and sentence accuracy is an indication of the proportion of sentences which contained no errant segments. “Total segment types” is the total number of distinct segments found in the overall dataset, adjusted to account for differences in segment granularity. The finer segment granularity of ChaSen over JUMAN and ALTJAWS is borne out by its slightly higher average segment number. ALTJAWS was found to outperform the remaining two systems in terms of segment precision, at an error reduction rate of over 17%.[18] ChaSen and JUMAN performed at the exact same level of segment precision, although one must bear in mind that this statistic for ChaSen includes 200 $(= (13.0 - 12.0) \times 200)$ additional segments judged to be 100% correct (conjugating affixes, etc. attributable to the finer segmentation granularity of ChaSen); when these are factored out from calculations, the segment precision for ChaSen falls back to 98.2%, and JUMAN has a slight empirical advantage. Looking next to segment recall, ChaSen outperformed both ALTJAWS and JUMAN (operating at an error rate roughly 17% lower than ALTJAWS, and exactly half that of JUMAN). In practice, this difference was due to JUMAN being less adept at segmenting compound nouns, and tending to bundle multiword katakana word sequences such as ゲートロックバルブ [*gēto-rokku-barubu*] ‘gate-lock valve’ together into a single segment. ALTJAWS was remarkably successful at segmenting katakana word sequences, achieving a segment precision of 100% and segment recall approaching 99%. For the other two systems, on the other hand, this was the source of almost all false negatives, and roughly half of the false positives.[19] This is thought to have been the main cause for the different translation retrieval results over data segmented by the three systems. The reason for JUMAN’s slight edge over ChaSen is partially due to the higher number of verb morpheme segments contained in the ChaSen output. Under the weighting schema adopted at this point, all segments are treated equally, such that the higher proportion of verbal segments in ChaSen output has the potential to create a bias towards strings which share the same verbal complexes, and downplay the importance of nouns.

One point which is not immediately reflected in the precision/recall analysis is the level of consistency in case of error. For translation retrieval, segmentation consistency is almost as important as segmentation accuracy, in that we are mainly interested in producing a match between identical words or word clusters, irrespective of just how that match is reached. Indeed, this is a major factor in the high performance of character-based indexing. While consistency is guaranteed in the case of a correct analysis, we wish to gain an insight into the consistency of

	ChaSen	JUMAN	ALTJAWS
Average segments per translation record	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

Figure 11. Segmentation performance of ChaSen, JUMAN and ALTJAWS

the three systems in the case of error. To this end, 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, as what errors it makes are reproducible on alternate data.

Here again, we face difficulty in comparing the output of the three systems directly due to their heterogeneous segmentation output styles. To gain a truly indicative view of the degree of variation in segment types independent of the handling of predicates, we normalise the output of the three systems by configuring them to output the base form of each morpheme, and filtering any conjugational affixes and verbal morphemes from the output. This results in a total of 634 segment types for ALTJAWS, as compared to 650 for ChaSen and 656 for JUMAN. Based on this, ALTJAWS errs more consistently than the remaining two systems, and there is very little to separate ChaSen and JUMAN. Indeed, while ChaSen produced fewer segment types than JUMAN, it is important to realise that the recall gain to ChaSen exceeds this small disparity. Ostensibly what this means is that ChaSen is more erratic in case of error, but that its higher segment recall covers for this. In conclusion, therefore, ALTJAWS holds a marginal advantage over ChaSen and JUMAN, with little separating the latter two systems.

5.2.4. *Summary*

To return to evaluation of the translation retrieval task for the three systems, there would seem to be a direct correlation between segmentation accuracy and retrieval performance, with segmentation accuracy on key terms such as compound nouns 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 for each segment would seem to produce slight performance gains, although not to an immediately tangible level. It is important to compare the final results for ALTJAWS with lexical normalisation, back to the original results for character-based indexing, and realise that the absolute best word-based indexing system configuration is by and large inferior to character bigrams. Thus, for all the effort expended in tweaking the performance of word-based indexing, character-based indexing, coupled with a simple bigram model, maintains a performance advantage. Our original observation as to the superiority of character-based indexing over word-based indexing still holds, therefore.

5.3. Experiment III: The effects of different character type-based segment weighting schemata

5.3.1. *Background*

Having variously tested the relative merits of the different string comparison methods and the effects of segmentation, we next turn to methods of enhancing retrieval performance through segment weighting. The first such approach is segment weighting based on the character composition of that segment, through a static weighting schema.

To reiterate the claims of Sect. 2.3, for Japanese we predict that segments made up entirely of hiragana will have less bearing on retrieval performance than other segment types. It is also possible to take this a step further in suggesting that segments not containing any kanji characters will have less import on retrieval performance than those containing kanji. To test this hypothesis, we define the class of "light segments" to be those segments made up exclusively of hiragana, numerals and Roman characters, and stipulate a range of static *sweight* values for light and other segments to balance up the impact each will have on translation retrieval.[20] This is done in the form of the three scoring schemata in Figure 12, where "other segments" refers to segments containing kanji characters. With katakana, we experiment with the two options of: (a) including them in the category of light segments, and (b) including them in the category of other segments, along with kanji. The theoretical motivation here is that katakana segments make little semantic contribution (encode semantically peripheral words, etc), or that they make a major semantic contribution (encode technical terms, etc.), respectively.

5.3.2. *Methodology*

The translation accuracy over each weighting schema is tested and compared to the basic translation accuracy from Figures 7 and 8 (that is with the *sweight* for light segments set equal to that for other non-

Segment type	$W^{JE}{}_1$	$W^{JE}{}_2$	$W^{JE}{}_3$
Punctuation	0	0	0
Light segments	0.01	0.2	0.5
Other segments	1	1	1

Figure 12. The SL *sweight* schemata tested for Japanese (katakana are classified as either light segments [−k] or other segments [+k])

punctuation segments, at 1). Recall that the results in Figures 7 and 8 were based on an *sweight* schema where punctuation got a weight of 0 and all other segments got a weight of 1.

As an alternative to the hand-selected *sweight* schemata, we could attempt to learn dynamically a set of segment weights (Och and Ney, 2003). Given our relatively small datasets, it is doubtful that machine learning would improve results further, but we leave this exploration for future research.

5.3.3. *Results*

Final translation accuracies are given in Figure 13, including the average gain in accuracy over all methods. Each [+k] column indicates the case of katakana having been included in the definition of light segments, and each [−k] column the case of exclusion of katakana from the definition. Underlined accuracies represent the best performance for that string comparison method under the given indexing paradigm.

We first analyse the results for character-based indexing, where the respective *sweight* values translate across to weights on character bigrams, recalling that character-based indexing is applied exclusively to character bigrams at this stage of evaluation. There is a negative effect on translation accuracy for all three weighting schemata when katakana are included in the definition of light segments. The relative drop-off in accuracy lessens as the *sweight* for light segments increases, finally reaching the level of the basic method for an *sweight* value of 1 for light segments. The string comparison methods most keenly affected by a depleted *sweight* value for light segments were WSC and Sato92, probably because of katakana sequences being weighted down against kanji strings. When katakana characters are excluded from the definition of light segments, on the other hand, slight gains in translation accuracy are observable, peaking for $W^{JE}{}_2$[−k]. Combining these two trends together we can conclude that katakana characters have an important role to play in translation accuracy for the given domain, due to their predominant usage in technical terms. Our original hypothesis

	Method	W^{JE}_1 [+k]	W^{JE}_1 [−k]	W^{JE}_2 [+k]	W^{JE}_2 [−k]	W^{JE}_3 [+k]	W^{JE}_3 [−k]
Char-based	VSM	60.39	60.47	60.31	60.33	60.33	60.31
	Token int	60.83	61.11	60.99	60.91	60.95	60.91
	3-op edit dist	58.23	58.25	57.99	58.81	58.05	58.09
	3-op edit sim	61.13	61.17	61.29	61.39	61.17	61.15
	4-op edit dist	51.74	51.73	51.58	51.45	51.94	51.77
	4-op edit sim	59.51	59.73	59.59	59.69	59.61	59.49
	WSC	49.20	56.39	53.35	57.19	55.07	57.29
	Sato92	46.90	54.99	51.75	55.83	53.67	55.63
	Av'ge gain	−7.2%	+0.0%	−3.1%	+0.7%	−1.3%	+0.5%
Word-based	VSM	57.39	57.21	57.29	57.23	57.19	57.35
	Token int	58.29	58.39	58.33	58.11	57.41	57.45
	3-op edit dist	57.47	57.31	57.51	57.31	56.97	57.15
	3-op edit sim	58.29	58.17	58.21	57.85	57.53	57.47
	4-op edit dist	49.34	48.94	49.18	49.02	48.72	48.80
	4-op edit sim	56.83	56.69	56.61	57.01	56.01	56.27
	WSC	56.49	56.51	56.09	55.91	55.47	55.41
	Sato92	56.01	55.99	55.37	55.35	54.65	54.89
	Av'ge gain	+6.5%	+4.9%	+5.5%	+5.3%	+3.6%	+4.0%

Figure 13. The retrieval performance over different character-based segment weighting schemata

that hiragana characters are of lesser importance than either kanji or katakana is tentatively supported in the modest accuracy gains when katakana characters are treated on a par with kanji characters. While we do not present the results here, the various weighting schemata were also tested over other segment contiguity models. With character unigrams, the drop-off in accuracy when katakana were included in the definition of light segments, was even more accentuated than that for character bigrams.

Turning next to the accuracies for word-based indexing, we see a marked improvement over the relative gains for character-based indexing. Contrary to the results for character-based indexing, here the best results are achieved when katakana characters are included in the category of light segments, with the best individual accuracy gain of 6.5% witnessed for W^{JE}_1[+k]. The relative gain for the [+k] system configurations drops continuously as the *sweight* associated with light segments increases, again contrasting with the results for character-based indexing. With katakana excluded from our definition of light segments, on the other hand, the results mirror those for character-based indexing, peaking for $W^{JE}{}_2[-k]$ at a gain of 5.3%.

We have no immediate explanation for the contradictory results under character- and word-based indexing with respect to the proper treatment of katakana, other than to say that weighting based on character type has much greater impact on word-based indexing than character-based indexing. Note that while considerably higher gains in accuracy were observed for word-based indexing, the best results for word-based indexing were still inferior to the worst results for character-based indexing, with the exception of the WSC and Sato92 methods.

5.4. Experiment IV: The effects of kanji

5.4.1. *Background*

In Sect. 5.3, we observed that translation performance gains were possible through treating kanji-containing segments as "first-rate citizens" and weighting them up over other segment types. 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 by their katakana (alphabetised) equivalents.

An additional motivation for removing kanji is to study the semantic smoothing effects of individual kanji characters, alluded to in Sect. 2.1. 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 overlap in character content (namely 作), and in the process pick up on the semantic correspondence between the two words (assuming that the segment contiguity model includes unigrams).

5.4.2. *Methodology*

To test the effect of kanji characters (i.e. logographs) on translation retrieval performance, 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 katakana characters; the rough equivalent for English would be to delete all word boundaries and chunk the input into syllable chunks, with no differentiation between word and syllable boundaries (e.g. *char ac ter based in dex ing* for *character-based indexing*). We then ran the same methods over this modified input, using exactly the same technique as for the original

	Method	Accuracy	Nonopt edit dist	Unique outputs	Av'ge time
Char-based	VSM	**60.72**	8.42	0.97	1.03
	Token int	**60.68**	8.41	0.96	1.40
	3-op edit dist	56.67	7.92	0.84	1.64
	3-op edit sim	**61.28**	8.17	0.96	2.39
	4-op edit dist	**52.28**	9.37	0.80	4.49
	4-op edit sim	**59.96**	8.14	0.94	5.59
	WSC	55.97	9.54	0.97	199.91
	Sato92	53.69	10.10	0.97	216.71
Word-based	VSM	57.66 (−5.0%)	8.11	0.96	0.82
	Token int	57.90 (−4.6%)	8.32	0.94	1.08
	3-op edit dist	57.22 (+1.0%)	8.20	0.84	1.29
	3-op edit sim	58.08 (−5.2%)	8.24	0.94	1.77
	4-op edit dist	48.42 (−7.4%)	10.28	0.73	3.36
	4-op edit sim	55.79 (−7.0%)	8.36	0.91	4.41
	WSC	54.99 (−1.8%)	9.15	0.92	25.55
	Sato92	53.27 (−0.8%)	9.81	0.92	26.34

Figure 14. Results for fully alphabetised data

experiment (including bigrams for character-based indexing and mixed unigrams/bigrams for word-based indexing).

5.4.3. *Results*

The results are presented in Figure 14, with times calculated relative to 3-operation edit distance over word unigrams from the first experiment.

As for the original experiment, character-based indexing was found to be superior to word-based indexing for the given n-gram models, at a level of statistical significance in most cases.[21] It is worth emphasising here that the system is operating over fully alphabetised Japanese data, such that with character-based indexing, we have only a jumble of what are essentially metaphoneme pairs to go on in deciding on an appropriate translation candidate for the given input. Even in these impoverished conditions, segmentation is by and large detrimental.

We originally removed kanji from the data to test whether the superiority of character-based indexing to word-based indexing was a side-effect of kanji-based semantic smoothing. That we were able to reproduce the same basic set of results as for fully lexically-discriminated data would suggest that it is not kanji that had bolstered the performance of character-based indexing. Having said this, returning to our synonym example from Sect. 5.4.1, in which the single-segment nouns 操作 [*sōsa*] and 作動 [*sadō*], which both translate into English as 'operation', the overlap in kanji 作 is reflected in overlap in alphabetisation (even though 作 is pronounced differently in the two words), which is retained in the katakana version. In fact, different readings for the same kanji will tend to have subtly or otherwise different meanings, such that by transposing the data into a near-phonetic representation, we are refining the synonym detection process for words sharing some kanji content. That is not to say, however, that words with similar phonetic content will generally be semantically associated in some way. In this sense, it is surprising that the fall-off in performance was as slight as it was. In light of this and the low level of drop-off in performance in the absence of kanji, we can postulate that the character bigram method in particular is highly robust and capable of handling a broad range of input types. This suggests ramifications for alphabet-based, nonsegmenting languages such as Thai, where our expectation would be for analogous results to be produced.

It is important to note the marked slowdown for WSC and the closely-related Sato92, largely due to them operating over unigrams. This does not occur to the same degree for word-based indexing because segmentation produces segment differentiation, and the remaining string comparison methods are relatively immune to slowdown under character-based indexing, due to them operating over highly-discriminated character bigrams.

5.5. Scalability of performance

5.5.1. *Background*

All results to date have been based on a single dataset of fixed size. While this has provided a valuable insight into a wide range of phenomena, we have no way of knowing how the results will scale with TMs of expanded size. It could well happen, for example, that the performance disparity that exists between character- and word-based indexing is narrowed and even reversed as the size of data increases. An orthogonal issue which we have tended to neglect to this point is the inherent speed of the different methods. Does, for example, a 10-times difference in retrieval speed for a given TM size translate into a 100-times difference for a TM 10 times the size, or is relative speed independent of TM size? While we have complexity results for each method, the speed gains resulting from our optimisations (Sect. 3.4) make it hard to predict accurately the actual running time for a given method over a given dataset. In order to address these questions, we employ a second parallel corpus of expanded size (61k sentences vs. the

3k sentences in the original dataset), and test the translation performance of a number of different system configurations over increasingly large subsets of the original corpus, to simulate TM size variation.

The particular parallel corpus we used was developed by JEIDA, and originates from government white papers on the Japanese economy, translated into English (Isahara, 1998). We extracted a total of 61,236 translation records from the JEIDA corpus,[22] over 20 times the number of translation records in the construction machinery corpus. One reservation with this dataset is that the granularity of alignment is fairly coarse, with a single segment often extending over multiple sentences. The average segment and character lengths of the Japanese component of each translation record are 45.3 and 76.3 respectively, vastly greater than the figures of 14.3 and 27.7 for the construction machinery corpus used for evaluation up to this point. On the English side, the average word length of a single translation record is 35.7, again representing a steep increase from the 13.3 word length of the original corpus. The implication of these inflated figures is that it will frequently occur that we cannot find a translation record similar enough to the input to be useful, and the bulk of translation candidates will simply be empty strings. Naturally, this trend will be lessened as the size of the TM increases and we gain access to a more diverse range of translation records.

5.5.2. *Methodology*

We simulate TMs of differing size by randomly splitting the JEIDA corpus into ten partitions, and running the various system configurations first over partition 1, then over combined partitions 1 and 2, and so on until all ten partitions are combined together into the full corpus. For each of the ten datasets produced in this manner, we computed the translation accuracy and retrieval speed, in the same manner as indicated in Sect. 5.1. Retrieval speed was calibrated based on the mean retrieval speed for word unigrams under 3-operation edit distance over the original corpus (i.e. relative speeds are directly comparable to those presented to this point).

With the JEIDA dataset, we restrict our attention to the three methods which performed particularly well over the original dataset, namely the VSM, 3-operation edit distance and 3-operation edit similarity.[23] That is, we tested one bag-of-words and two segment-order-sensitive methods, and one distance- and two similarity-based methods, selecting those methods which tended to perform best in each such category.

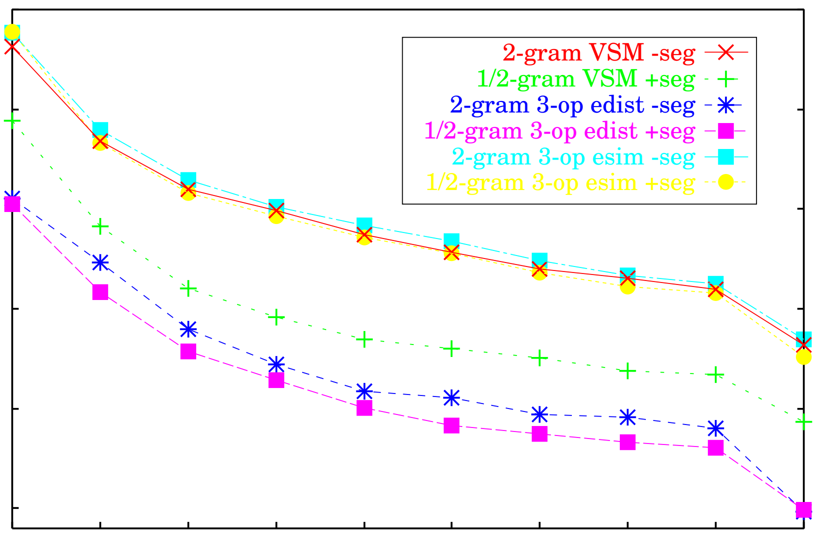

Figure 15. Translation accuracies of the VSM, 3-operation edit distance and 3-operation edit similarity over datasets of increasing size

5.5.3. *Results*

The transition in translation accuracy for the different methods over the ten datasets of varying size is indicated in Fig. 15, with each string comparison method tested under character bigrams ("2-gram −seg") and mixed word unigrams/bigrams ("1/2-gram +seg"). Note that while we do not present the results here, other models of segment contiguity were tested, in order to verify the superior performance of these methods over the two indexing paradigms. Results remarkably similar to those for the construction machinery dataset were obtained: character bigrams and word unigrams/bigrams were found to outperform other methods within the respective indexing paradigms.

A striking feature of the various graphs is that they are right-decreasing. This is an artifact of the size of the translation records and resultant data sparseness. That is, for smaller datasets, in the bulk of cases, no translation record in the TM is similar enough to the input to warrant consideration as a translation candidate. This is reflected in the ratio of optimal translation sets made up solely of the null string in Figure 16, for each dataset (averaged across the optimal translation data from WSC and bigram-based 3-operation edit distance). TM size is measured in translation records.

One key trend in Fig. 15 is the relative disparity in accuracy between character- and word-based indexing for each of the three string

TM size	% of total	Null-string proportion (%)
5976	10	82.3
11952	20	74.3
17937	30	69.4
23922	40	65.7
29898	50	62.7
35874	60	59.9
41859	70	57.3
47835	80	55.2
53820	90	53.4
61236	100	50.6

Figure 16. Relative proportion of singleton null-string optimal translations as the dataset grows

comparison methods. For all methods, the absolute margin between the character- and word-based implementations is maintained as the TM size grows, with character-based indexing coming out on top for all methods. This supports our claim that character-based indexing is superior to word-indexing, but also goes one step further in suggesting that we can expect this trend to be reproduced for all TM sizes and a variety of data types. It is interesting to note that the disparity between character- and word-based indexing is greatest for VSM, and that with the two segment-order-sensitive methods, the spread is much smaller (but still significant). This is contrary to our earlier results, where token intersection and VSM showed the smallest gains for character-based indexing. As with the original dataset, 3-operation edit distance outstripped 3-operation edit similarity in terms of raw accuracy. With character-based indexing, VSM approximately equalled the accuracy of character-based 3-operation edit similarity for all dataset sizes. Based on this result, we conclude that there is little to distinguish bag-of-words from segment-order-sensitive methods in terms of retrieval accuracy.

As with the original dataset, 3-operation edit similarity was the strongest performer, with VSM coming second and 3-operation edit distance a distant third. The sizeable disparity of around 20% between 3-operation edit similarity and 3-operation edit distance was unobserved in our original evaluation, suggesting its performance is brittle.

Next, we turn to consider the mean retrieval times for each method, under the two indexing paradigms. Times are presented in Fig. 17, cal-

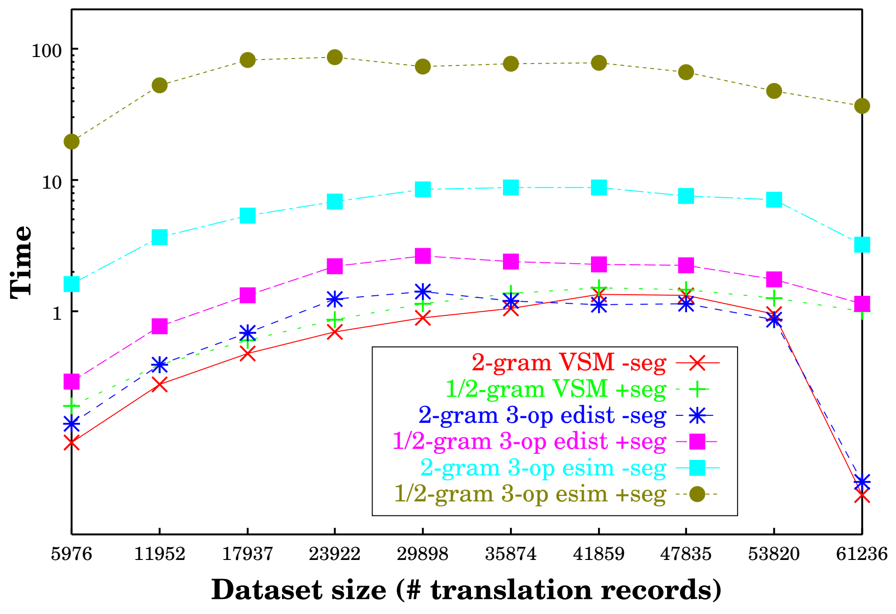

Figure 17. Relative retrieval speeds of the VSM, 3-operation edit distance and 3-operation edit similarity over datasets of increasing size

ibrated according to 3-operation edit distance run over word unigrams for the construction machinery dataset. The reader should note that a logarithmic scale has been used on the y-axis 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 time as 3-operation edit distance run over word unigrams for the construction machinery dataset). Most importantly, only minor increases in retrieval speed were evident as the TM size increased, which were then reversed as the retrieval time dropped back for the larger datasets. This same convex shape was observed for 3-operation edit similarity, although the slope of the curve was much steeper over the smaller datasets, with the final running time for mixed word unigrams/bigrams over 100 times that for VSM or 3-operation edit distance. For character bigrams also, the final retrieval time was markedly slower than those for the other two methods, at around one order of magnitude. Clearly, this blow-out in retrieval time for 3-operation edit similarity is not desirable if we wish our TM system to be scalable, and the coupling of 3-operation edit distance with lower-order n-gram models would be computationally inappropriate.

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.

5.6. Summary

To summarise the findings of this section, character-based indexing was found to be superior to word-based indexing once more, corroborating our findings over the original dataset. Perhaps more importantly, VSM and 3-operation edit distance returned almost constant mean retrieval times over datasets of varying size, demonstrating their scalability over large TMs. 3-operation edit similarity proved to be considerable slower, particularly for a mixed n-gram model, but the blow-out was at least found to be bounded, at around 10 times the speed of the other two methods for character-based indexing.

6. English–Japanese Translation Retrieval Results

6.1. Background

Based on the combined results of Sect. 5, we concluded that character-based indexing is superior to word-based indexing, and also that there is little to separate bag-of-words and segment-order-sensitive methods in general. These findings were based on analysis of Japanese, however, and we have little way of knowing how they translate across to other languages. For this reason, in this section, we consider a new language direction for translation retrieval, namely English–Japanese, reusing the data we have at hand. Another reason that we are interested in English–Japanese translation retrieval is to justify the *sweight* schema chosen for the evaluation of translation optimality, and in particular our handling of stop words.

Naturally, words are explicitly segmented in English, making the question of whether to segment or not moot.[24] The bag-of-words vs. segment-order sensitivity issue is still pertinent, however, as is the n-gram order. We therefore consider the full range of string comparison methods, as well as each of word unigrams, word bigrams and mixed word unigrams/bigrams, as before.

6.2. Methodology

In evaluation of translation optimality, we employ the same method as described in Sect. 4.3.1 for Japanese–English translation retrieval

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

Figure 18. *sweight* schema for TL evaluation of Japanese

Segment type	$W^{EJ}{}_1$	$W^{EJ}{}_2$	$W^{EJ}{}_3$	$W^{EJ}{}_4$	$W^{EJ}{}_5$	$W^{EJ}{}_6$
punctuation	0	0	0	0	0	0
stop words	0	0.05	0.1	0.2	0.5	1
other words	1	1	1	1	1	1

Figure 19. *sweight* schemata experimented with for English

(combining the results for 3-operation edit distance and WSC), excepting that character-based was used in place of word-based indexing; we employed the TL *sweight* schema given in Figure 5 for Japanese, which we reproduce here as Figure 18.

As with Japanese–English translation retrieval, we tested the effects of different static *sweight* schemata, operating over word type. Principally, we distinguish between punctuation, stop words and any other words, with stop words defined as those contained within the SMART (Salton, 1971) stop-word list, as was the case with optimal translation determination for Japanese–English translation retrieval. The full range of *sweight* schemata evaluated is presented in Figure 19. Essentially, we vary the relative impact of stop words by varying the associated *sweight* between 0 and 1, in the manner adopted for character type-based weighting in Sect. 5.3. Note that $W^{EJ}{}_1$ (i.e. stop words filtered out of all strings) corresponds to the TL *sweight* schema utilised in Japanese–English translation retrieval. A stop word in the case of an n-gram model of order 2 or greater is interpreted as an n-gram of stop words. That is, a segment is given the indicated stop-word weight only in the case that it is made up entirely of stop words.

6.3. Results

The results for the three segment contiguity models, as tested under the different *sweight* settings, are presented in Figure 20. The first thing to note is the high correlation between these results and those for Japanese–English translation retrieval, namely the superiority of

	Method	$W^{EJ}{}_1$	$W^{EJ}{}_2$	$W^{EJ}{}_3$	$W^{EJ}{}_4$	$W^{EJ}{}_5$	$W^{EJ}{}_6$
Unigram	VSM	46.25	50.10	50.16	49.90	$\underline{50.36}$	46.46
	Token int	49.48	$\underline{50.60}$	50.32	49.96	49.70	47.06
	3-op edit dist	47.12	48.78	$\underline{49.56}$	48.92	48.94	46.90
	3-op edit sim	50.94	$\underline{51.16}$	$\underline{51.16}$	51.08	50.74	49.56
	4-op edit dist	42.83	43.80	43.86	$\underline{43.96}$	42.94	41.26
	4-op edit sim	49.24	49.88	$\underline{50.52}$	50.28	49.80	48.08
	WSC	50.58	51.08	$\underline{51.34}$	50.96	50.94	50.50
	Sato92	49.72	$\underline{50.08}$	49.70	49.62	49.52	46.98
	Av’ge gain	−2.2%	+0.2%	+0.5%	0.0%	−0.5%	−4.6%
Bigram	VSM	$\underline{51.60}$	51.40	$\underline{51.60}$	51.32	51.42	51.32
	Token int	52.02	52.02	51.86	$\underline{52.65}$	51.62	51.86
	3-op edit dist	$\underline{50.12}$	49.88	49.68	49.62	49.40	49.54
	3-op edit sim	51.68	$\underline{52.26}$	51.92	51.54	52.06	51.74
	4-op edit dist	43.94	43.62	43.56	43.98	43.86	$\underline{44.00}$
	4-op edit sim	$\underline{50.36}$	50.10	49.94	50.30	50.20	49.90
	Av’ge gain	+0.1%	−0.0%	−0.2%	0.0%	−0.2%	−0.3%
Mixed	VSM	49.59	52.98	$\underline{53.38}$	53.27	52.73	50.64
	Token int	$\underline{53.61}$	52.94	52.60	52.44	52.79	52.08
	3-op edit dist	$\underline{52.36}$	51.62	51.70	52.18	51.08	50.30
	3-op edit sim	$\underline{52.86}$	52.64	52.73	52.44	52.38	52.26
	4-op edit dist	43.72	43.20	$\underline{43.84}$	43.46	43.32	42.18
	4-op edit sim	50.54	$\underline{50.62}$	$\underline{50.62}$	50.54	50.54	49.76
	Av’ge gain	−0.4%	−0.1%	+0.2%	0.0%	−0.4%	−1.7%

Figure 20. Performance for English–Japanese translation retrieval with different character-based methods, using a unigram, bigram, and mixed unigram/bigram segment contiguity model

word bigrams and mixed unigrams/bigrams to unigrams, and also the relative performances of the different string comparison methods. Additionally, the bag-of-words methods perform at an equivalent level to the best of the segment-order-sensitive methods, with 3-operation edit similarity coming out as the overall method of choice. The strong performance of the bag-of-words methods is perhaps a little surprising, and discounts any suggestion that their strong showing when run over Japanese is related to its relatively free word order (see Sect. 2.2). These findings underline the language independence of the conclusions drawn for Japanese–English translation retrieval, although further language pairs must be tested to confirm the trend observed here.

The results highlight the fact that there is a strong case for treating stop words differently to other words, and that for word bigrams at least, it is possible to filter stop words (i.e. adjacent pairs of stop words) out of strings all together. For the remaining two models of segment contiguity, which contain a component of unigrams, both scoring stop words too low and scoring stop words too high was found to be detrimental, with $\mathrm{W^{EJ}}_3$ providing the optimal trade-off between the two.

Our choice of word bigrams for the determination of translation optimality in Japanese–English translation retrieval, despite mixed word unigrams/bigrams providing slight enhancements in accuracy, is rationalised through consideration of retrieval speed. That is, while the scales are tipped marginally in favour of mixed word unigrams/bigrams in terms of retrieval accuracy, they point unequivocally to word bigrams when retrieval time is taken into account, particularly for large-scale TMs such as the JEIDA dataset. The $\mathrm{W^{EJ}}_1$ *sweight* setting would also appear to be optimal for word bigrams in terms of accuracy, and again has advantages in terms of speed (i.e. we can completely filter our stop-word bigrams from the dataset, reducing both the search space and number of segment comparisons).

7. Discussion and Concluding Remarks

7.1. Discussion

We conclude by spending a moment reflecting on the combined results for Japanese–English translation retrieval. At the outset of evaluation, we chose to explore the following parameters: character- vs. word-based indexing; bag-of-words vs. segment-order-sensitive string comparison method; and the effectiveness of n-gram-based segment contiguity modelling. Lesser areas of interest included 3-operation edit distance/ similarity vs. 4-operation edit distance/similarity, and WSC vs. Sato92. Here, we review the findings and discuss related literature for each of these items.

First, character-based indexing was found to be superior to word-based indexing almost without exception, generally at a level of statistical significance. The disparity between the two indexing paradigms was particularly noticeable when character bigrams were compared with mixed word unigrams/bigrams, the best-performing models of segment contiguity for the respective indexing paradigms. Most research relating to translation retrieval for Japanese has taken it for granted that segmentation will enhance retrieval performance. That is, the overhead associated with segmentation was implicitly assumed to be justifiable in terms of producing greater retrieval robustness, or in other words a “slower but steadier” method. While we were prepared to question this standpoint, we were certainly not expecting hare-like character-based indexing to come out the clear leader. Looking to the related literature, there is limited support for our findings, particularly in relation to IR where Fujii and Croft (1993) established that character-based indexing can perform at comparable levels to word-based indexing in Japanese IR. In research laying the foundations for this paper, Baldwin and Tanaka (2000) and Baldwin (2001a) report similar findings for a Japanese–English translation retrieval task. To the author’s knowledge, there is no empirical evidence to suggest that segmentation enhances accuracy in an analogous retrieval context.

Considering next segment-order sensitivity in the string comparison method, the findings for retrieval accuracy were somewhat mixed, with bag-of-words methods coming out on top occasionally, but segment-order-sensitive methods more generally holding an advantage. All in all, however, there was very little to separate the two in terms of accuracy and the degree of error in the case of suboptimal output. In terms of speed, the bag-of-words methods were reliably fast, whereas depending on the segment-order-sensitive method, sizeable blow-outs in time were observed, particularly for WSC and Sato92, and to a lesser degree, the edit similarity methods. For real-world applications, speed is obviously a concern. Given that there is little to distinguish the two method types in terms of retrieval accuracy, bag-of-words methods must be recognised as the more attractive option. Once again, therefore, the sleek, seemingly fragile method comes out on top.

These findings contradict those of Baldwin and Tanaka (2000), who found segment-order-sensitive methods to excel over bag-of-words methods. The principal cause for this was that calculation of translation optimality was based exclusively on 3-operation TL edit distance, producing a bias towards 3-operation SL edit distance in retrieval and against the bag-of-words methods. We claim that our new results reflect more accurately the true performance of the different methods, due to evaluation being averaged across 3-operation edit distance and WSC, two heterogeneous methods picking up on distinct effects in the output.

Recall that our comparison of bag-of-words and segment-order-sensitive methods is based on what is suggested to be a representative selection of comparison methods in each category. At no point do we claim that the methods targeted herein are optimal, and significant improvement may well be possible with more sophisticated matching means. We do suggest, however, that bag-of-words methods have the greatest room for computational optimisation, as the only types of optimisation employed are an inverted file and cache of string segment lengths, unlike the segment-order-sensitive methods where much effort was put into getting the maximal possible performance from the basic method. Optimisation of the bag-of-words methods would thus open the way for more computationally expensive means of comparison, without sacrificing TM access times.

Segment contiguity modelling in the form of n-grams was found to be a valuable supplement to the various system configurations. Of unigrams, bigrams and mixed unigrams/bigrams, bigrams were found to perform best for character-based indexing, and mixed unigrams/ bigrams for word-based indexing. Despite segment contiguity modelling being orthogonal to both the choice of indexing paradigm and the type of string comparison method, the following correlations were observed: character bigrams are a computationally lightweight approximation of word unigrams, and bag-of-words methods coupled with higher-order n-gram models approach the accuracy of segment-order-sensitive methods through gaining access to local segment ordering. Our results show unequivocally that the simple n-gram chunking of characters is more effective than word segmentation, while retaining the same speed advantages as word-based indexing.

Turning to more specific issues, WSC was suggested as an enhancement over the edit distance and similarity methods, in that it fed off segment order at the same time as preferring contiguous matches over noncontiguous matches. In evaluation, this awareness of segment contiguity produced no real gain in retrieval accuracy, and simply made the method more cumbersome. In particular, it was found that by combining a model which is oblivious to segment contiguity with a higher-order n-gram model, we could both accelerate retrieval times and benefit from limited modelling of local contiguity. Given the gains achieved by edit distance and similarity over WSC when coupled with n-gram models, we feel confident in stating that WSC has little practical worth. Note that work has been done on buffering the inflated retrieval times seen for WSC, namely in the work of Sato (1994), where a massively parallel computer was used to speed up retrieval. Even when the speed factor is removed, however, the accuracy of WSC is inferior to that of the other methods.

In analysing the performance of WSC, we compared it to the closely related method of Sato (1992), and found that the slight modification made to Sato's method led to significant performance gains. Similarly with edit distance and similarity, we compared the classic 4-operation methods to simplified 3-operation methods, generated through the removal of the substitution operator. Our prediction that 3-operation methods were superior to 4-operation methods was verified in evaluation, and 3-operation edit similarity found to be the best of the four variants.

7.2. Conclusion

This research has been concerned principally with the relative impact of segment order and segmentation on translation retrieval performance for a TM system over Japanese data, as well as the value of n-gram-based local segment contiguity modelling. We simulated the effects of word order sensitivity vs. bag-of-words word order insensitivity by implementing a total of eight comparison methods: two bag-of-words approaches (VSM and token intersection) and six word-order-sensitive approaches (3-operation edit distance and similarity, 4-operation edit distance and similarity, and two versions of WSC). Each of these metrics was then tested under character-based and word-based indexing and in combination with a range of simple and mixed n-gram models, and the relative performance of each such system configuration evaluated. Evaluation of the output took the form of determining the set of optimal translation candidates through automatic means, and calculating the degree of correspondence between this set and the actual translation output. Character-based indexing was found to be superior to word-based indexing in all cases tested, particularly when supplemented with a character bigram model. Segmentation was thus discounted as being an unwarranted luxury. The bag-of-words methods were found to be about equal in retrieval potential to the better of the segment-order-sensitive methods, but the former tended to run faster.

In order to evaluate the impact of the segmentation system on final retrieval accuracy for word-based accuracy, we ran the various string comparison methods over the segment output of the ChaSen, JUMAN and ALTJAWS systems for the same dataset, as well as lexically normalised output from ALTJAWS. Our experiment showed a strong correlation between retrieval accuracy and reliability/consistency in the segmentation of compound nouns, and that lexical normalisation produces marginal gains in retrieval performance. We went on to test a range of static segment weighting schemata based on character type, and showed that modest accuracy gains were possible through such methods. We also tried converting the dataset into katakana and seeing what difference this would make to translation retrieval, and found that we were able to reproduce the same basic results as for fully lexically-differentiated data. From this, we hypothesised that the role of kanji in retrieval was not as crucial as we had originally thought. After having established a range of results over a limited dataset, we experimented with a second dataset of a more practical size, and investigated the relation between retrieval speeds and accuracy for representative methods over different sized portions of the overall corpus. This resulted in the confirmation of our finding for the original dataset that (bigram-based) character-based indexing is superior to (mixed unigram/bigram-based) word-based indexing, and also that the retrieval speed for VSM and 3-operation edit distance remains constant as the TM size increases, unlike 3-operation edit similarity which was slower by on order of one or two. The retrieval accuracy of VSM and 3-operation edit similarity was roughly equivalent, whereas 3-operation edit distance performed poorly. Finally in evaluation, we reversed the language direction of translation retrieval, and for English–Japanese retrieval, verified the ranking of the string comparison methods derived in Japanese–English retrieval.

Acknowledgements

Valuable input into this research was received from Francis Bond, Kentaro Inui, Masaaki Nagata, Emmanuel Planas and Yorick Wilks, as well as the anonymous reviewers of this paper.

Notes

[1] `http://chasen-legacy.sourceforge.jp`

[2] `http://www-lab25.kuee.kyoto-u.ac.jp/nl-resource/juman-e.html`

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

[4] Parallels can be drawn here with "one-sense-per-discourse" constraint of Gale et al. (1992), which states that a given word will only ever occur in a single sense, for a given document or domain.

[5] The term "bag-of-words" is misleading in the sense that it is orthogonal to the issue of character- vs. word-based indexing. While noting this anomaly, we retain the term due to it being well-established nomenclature within NLP circles.

[6] Note that the ordering of the strings is arbitrary, and that all the comparison methods described herein are commutative for the given implementations.

[7] So as to filter out any differences between character- and word-based indexing, the strings have been chosen such that all "words" are in fact single characters. Retrieval performance is thus identical for the two indexing paradigms.

[8] Observe that the case of $Max = 1$ corresponds to the complement of 3-operation edit distance.

[9] Although it is possible to use n-gram methods to encode segment order information, as per Nagao and Mori (1994).

[10] Each translation record is unique as an ordered SL–TL string pair, but there is scope for SL or TL coincidence between translation records, an effect which was observed to a limited degree in the data.

[11] With a similarity-based string comparison method, the set of optimal translation candidates is defined similarly, except that the distance function is replaced by a similarity function, and the various *dist* inequalities are reversed in direction.

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

[13] While we do not present the results here, both trigram and mixed bigram/trigram models were also tested over character- and word-based indexing. In the case of character-based indexing, trigrams were found to be inferior to bigrams but superior to unigrams, and mixed bigram/trigrams were found to be slightly superior to simple trigrams but inferior to both bigrams and mixed unigram/bigrams. With word-based indexing, both trigrams and mixed bigram/trigrams were found to be inferior to the basic unigram model. See Baldwin (2001b) for full details.

[14] As determined by the paired *t* test ($p < 0.05$) here and throughout the paper.

[15] The reason that we restrict discussion of lexical normalisation to ALTJAWS is that full normalisation is not possible with ChaSen and JUMAN. Note that it is possible to convert each segment to its "base" form with ChaSen and JUMAN, that is the form in which it would occur in a dictionary entry, but not to reduce it one step further to a canonical form. ChaSen and JUMAN thus would not pick up on the equivalence of and , lexical alternates of *kawaru* 'to change/alter', whereas ALTJAWS would. We recognise that there is some merit in performing partial lexical segmentation, but choose to confine our focus to fully-fledged lexical normalisation for the purposes of this paper.

[16] With foreign loan words such as シリンダ [*shirinda*] 'cylinder',the canonical form returned by ALTJAWS is the original English rather than a canonical Japanese form. We chose to ignore all such English canonical forms and retain the original Japanese due to the transliteration process being somewhat unreliable.

[17] Strictly speaking, there is some variation in results for character-based indexing between the different datasets, firstly due to variation in the partitioning of the data for each, and secondly due to random tie-breaking resulting in different final translation candidates being extracted from the same original translation candidate set.

[18] The error reduction rate for precision a over baseline precision b is calculated as $(a - b)/(1 - b)$.

[19] The corollary of this observation is that ChaSen and JUMAN do better than ALTJAWS over nonkatakana strings, suggesting the possibility for hybridisation between ALTJAWS and ChaSen/JUMAN. Due to differences in segment granularity, however, it would be far from simple to combine the different styles of output.

[20] Numerals and Roman characters are included in the definition of light segments for the given domain, as they are generally used to stipulate part numbers, pressures, tightening torques and the like. There is a high degree of variation in the values of all of these segment types, and little expectation of full lexical match.

[21] Recall that numbers in bold in Figure 14 indicate statistical significance relative to the corresponding method under the alternative indexing method (character vs. word).

[22] While the source corpus contains more translation records than this and is exhaustively aligned, some alignment links are rated as being of better quality than others. Rather than compromising the quality of the overall TM, we chose to take only those translation records of a certain quality, resulting in the indicated count.

[23] Token intersection was also tested, in fact, and found to perform almost identically to VSM. Detailed results are omitted from this paper.

[24] It would be possible to use character-based indexing for English to capture morphological variation. We leave this experiment for further research.

References

Aramaki, E., S. Kurohashi, H. Kashioka, and H. Tanaka: 2005, 'Probabilistic Model for Example-based Machine Translation'. In: *Proceedings of the Tenth Machine Translation Summit (MT Summit X)*. Phuket, Thailand, pp. 219–226.

Backhouse, A. E.: 1993, *The Japanese Language: An Introduction*. Oxford, UK: Oxford University Press.

Baldwin, T.: 2001a, 'Low-cost, High-performance Translation Retrieval: Dumber is Better'. In: *Proceedings of the 39th Annual Meeting of the ACL and 10th Conference of the EACL (ACL-EACL 2001)*. Toulouse, France, pp. 18–25.

Baldwin, T.: 2001b, 'Making Lexical Sense of Japanese–English Machine Translation: A Disambiguation Extravaganza'. Ph.D. thesis, Tokyo Institute of Technology.

Baldwin, T. and H. Tanaka: 2000, 'The Effects of Word Order and Segmentation on Translation Retrieval Performance'. In: *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*. Saarbrücken, Germany, pp. 35–41.

Banerjee, S. and A. Lavie: 2005, 'METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments'. In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Ann Arbor, USA, pp. 65–72.

Brown, P. F., V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer: 1992, 'Class-Based *n*-gram Models of Natural Language'. *Computational Linguistics* **18**(4), 467–479.

Carl, M. and A. Way (eds.): 2003, *Recent Advances in Example-Based Machine Translation*. Dordrecht, Netherlands: Kluwer Academic.

Chen, S. F.: 1993, 'Aligning Sentences in Bilingual Corpora Using Lexical Information'. In: *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*. Columbus, USA, pp. 9–16.

Chen, S. F. and J. Goodman: 1996, 'An Empirical Study of Smoothing Techniques for Language Modeling'. In: *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. Santa Cruz, USA, pp. 310–318.

Culy, C. and S. Z. Riehemann: 2003, 'The Limits of N-Gram Translation Evaluation Metrics'. In: *Proceedings of the Ninth Machine Translation Summit (MT Summit IX)*. New Orleans, USA.

Doi, T., H. Yamamoto, and E. Sumita: 2005, 'Example-based machine translation using efficient sentence retrieval based on edit-distance'. *ACM Transactions on Asian Language Information Processing (TALIP)* **4**(4), 377–399.

Fujii, H. and W. B. Croft: 1993, 'A Comparison of Indexing Techniques for Japanese Text Retrieval'. In: *Proceedings of 16th International ACM-SIGIR Conference on Research and Development in Information Retrieval (SIGIR'93)*. Pittsburgh, USA, pp. 237–246.

Gale, W. and K. Church: 1993, 'A Program for Aligning Sentences in Bilingual Corpora'. *Computational Linguistics* **19**(1), 75–102.

Gale, W., K. Church, and D. Yarowsky: 1992, 'One Sense Per Discourse'. In: *Proceedings of the 4th DARPA Speech and Natural Language Workshop*. pp. 233–237.

Isahara, H.: 1998, 'JEIDA's English–Japanese Bilingual Corpus Project'. In: *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC'98)*. pp. 471–481.

Kitamura, E. and H. Yamamoto: 1996, 'Translation Retrieval System Using Alignment Data from Parallel Texts'. In: *Proceedings of the 53rd Annual Meeting of the IPSJ*, Vol. 2. pp. 385–386. (In Japanese).

Kukich, K.: 1992, 'Techniques for automatically correcting words in text'. *ACM Computing Surveys* **24**(4), 377–439.

Kurohashi, S. and M. Nagao: 1998, '*Nihongo keitai-kaiseki sisutemu JUMAN* [Japanese morphological analysis system JUMAN] version 3.5'. Technical report, Kyoto University. (in Japanese).

Langlais, P. and M. Simard: 2002, 'Merging Example-Based and Statistical Machine Translation'. In: *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas (AMTA)*. Tiburon, USA, pp. 104–113.

Manning, C. D. and H. Schütze: 1999, *Foundations of Statistical Natural Language Processing*. Cambridge, USA: MIT Press.

Marcu, D.: 2001, 'Towards a Unified Approach to Memory- and Statistical-Based Machine Translation'. In: *Proceedings of the 39th Annual Meeting of the ACL and 10th Conference of the EACL (ACL-EACL 2001)*. Toulouse, France, pp. 386–393.

Masek, W. and M. Paterson: 1980, 'A Faster Algorithm Computing String Edit Distances'. *Journal of Computer and System Sciences* **20**(1), 18–31. 1980.

Matsumoto, Y., A. Kitauchi, T. Yamashita, and Y. Hirano: 1999, '*Japanese Morphological Analysis System ChaSen Version 2.0 Manual*'. Technical Report NAIST-IS-TR99009, NAIST.

Moffat, A. and J. Zobel: 1996, 'Self-Indexing Inverted Files for Fast Text Retrieval'. *ACM Transactions on Information Systems* **14**(4), 349–379.

Nagao, M. and S. Mori: 1994, 'A New Method of N-gram Statistics for Large Number of N and Automatic Extraction of Words and Phrases from Large Text Data of Japanese'. In: *Proceedings of the 15th International Conference on Computational Linguistics (COLING '94)*. Kyoto, Japan, pp. 611–615.

Nakamura, N.: 1989, 'Translation Support by Retrieving Bilingual Texts'. In: *Proceedings of the 38th Annual Meeting of the IPSJ*, Vol. 1. pp. 357–358. (In Japanese).

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

Och, F. J. and H. Ney: 2003, 'A Systematic Comparison of Various Statistical Alignment Models'. *Computational Linguistics* **29**(1), 19–51.

Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu: 2002, 'BLEU: a Method for Automatic Evaluation of Machine Translation'. In: *Proceedings of the 40th Annual Meeting of the ACL and 3rd Annual Meeting of the NAACL (ACL-02)*. Philadelphia, USA, pp. 311–318.

Planas, E. and O. Furuse: 1999, 'Formalizing Translation Memories'. In: *Proceedings of the Seventh Machine Translation Summit (MT Summit VII)*. Singapore, pp. 331–339.

Planas, E. and O. Furuse: 2000, 'Multi-level Similar Segment Matching Algorithm for Translation Memories and Example-based Machine Translation'. In: *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*. Saarbrücken, Germany, pp. 621–627.

Planas, E. and O. Furuse: 2003, 'Formalizing Translation Memory'. in (Carl and Way, 2003), pp. 157–188.

Salton, G.: 1971, *The SMART Retrieval System: Experiments in Automatic Document Processing*. Upper Saddle River, USA: Prentice-Hall.

Sato, S.: 1992, 'CTM: An Example-Based Translation Aid System'. In: *Proceedings of the 14th International Conference on Computational Linguistics (COLING '92)*. Nantes, Frances, pp. 1259–1263.

Sato, S.: 1994, 'Example-Based Translation and its MIMD Implementation'. In: H. Kitano and J. Hendler (eds.): *Massively Parallel Artificial Intelligence*. AAAI/ MIT Press, pp. 171–201.

Sato, S. and T. Kawase: 1994, '*A High-Speed Best Match Retrieval Method for Japanese Text*'. Technical Report IS-RR-94-9I, JAIST.

Sato, S. and M. Nagao: 1990, 'Toward Memory-based Translation'. In: *Proceedings of the 13th International Conference on Computational Linguistics (COLING '90)*. Helsinki, Finland, pp. 247–252.

Somers, H.: 2003a, 'Recent advances in Example-Based Machine Translation'. in (Carl and Way, 2003), pp. 3–57.

Somers, H.: 2003b, 'Translation Memory Systems'. In: H. Somers (ed.): *Computers and Translation: A Translator's Guide*. Amsterdam/Philadelphia: John Benjamins, pp. 31–47.

Sumita, E. and Y. Tsutsumi: 1991, 'A Practical Method of Retrieving Similar Examples for Translation Aid'. *Transactions of the IEICE* **J74-D-II**(10), 1437–1447. (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).

Trujillo, A.: 1999, *Translation Engines: Techniques for Machine Translation*. London, UK: Springer Verlag.

Veale, T. and A. Way: 1997, 'Gaijin: A Bootstrapping, Template-Driven Approach to Example-Based MT'. In: *Proceedings of RANLP 1997 (Recent Advances in Natural Language Processing)*. Tzigov Chark, Bulgaria, pp. 239–244.

Véronis, J. (ed.): 2000, *Parallel Text Processing: Alignment and Use of Translation Corpora*. Dordrecht, Netherlands: Kluwer Academic.

Wagner, R. A. and M. J. Fischer: 1974, 'The String-to-String Correction Problem'. *Journal of the ACM* **21**(1), 168–173.

Wu, D.: 1994, 'Aligning Parallel English–Chinese Text Statistically with Lexical Criteria'. In: *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*. Las Cruces, USA, pp. 80–87.

Zhai, C. and J. Lafferty: 2004, 'A study of smoothing methods for language models applied to information retrieval'. *ACM Transactions on Information Systems (TOIS)* **22**(2), 179–214.