Approximate Sentence Retrieval for Scalable and Efficient Example-based Machine Translation

Debasis Ganguly Johannes Leveling Sandipan Dandapat Gareth J.F. Jones
Centre for Next Generation Localisation (CNGL),
School of Computing, Dublin City University,
Dublin, Ireland
{dganguly, jleveling, sdandapat, gjones}@computing.dcu.ie

ABSTRACT

Approximate sentence matching (ASM) is an important technique for tasks in machine translation (MT) such as example-based MT (EBMT) which influences the translation time and the quality of translation output. We investigate different approaches to find similar sentences in an example base and evaluate their efficiency (runtime), effectiveness, and the resulting quality of translation output. A comparison of approaches demonstrates that i) a sequential computation of the edit distance between an input sentence and all sentences in the example base is not feasible, even when efficient algorithms to compute the edit distance are employed; ii) in-memory data structures such as *tries* and *ternary search trees* are more efficient in terms of runtime, but are not scalable for large example bases; iii) standard IR models which only cover material similarity (e.g. term overlap), do not perform well in finding the approximate matches, due to their lack of handling word order and word positions. We propose a new retrieval model derived from language modeling (LM), named LM-ASM, to include positional and ordinal similarities in the matching process, in addition to material similarity. Our IR based retrieval experiments involve reranking the top-ranked documents based on their true edit distance score. Experimental results show that i) IR based approaches result in about 100 times faster translation; ii) LM-ASM approximates edit distance better than standard LM by about 10%; and iii) surprisingly, LM-ASM even improves MT quality by 1.52% in comparison to sequential edit distance computation.

KEYWORDS: Example-Based Machine Translation, Information Retrieval, Approximate Sentence Matching, Edit Distance.

1 Introduction

The quality of data-driven machine translation (MT) mostly depends on the size of parallel data available for training. Statistical MT effectively discards all training data once the translation model and language model have been generated, which are typically based on word n-grams. This can lead to poor quality translations because translation context is limited by the value of n. In contrast, example-based MT (EBMT) usually stores the full sentence pairs in source and target data in an example base and uses translations of sentences similar to the input sentence as a template for translation. Thus, EBMT systems can often better capture long range dependencies and rich morphology. State-of-the-art MT systems comprise both statistical and example-based MT components (Dandapat et al., 2012). Typical EBMT systems (Somers, 2003; Nagao, 1984) comprise three processing stages to translate an input sentence Q with information from an example base of sentence pairs (S_i, T_i), where S_i and T_i are in source and target language, respectively: 1. Matching, where sentences similar to Q are identified, and the translation template T_i with the maximum similarity between Q and S_i is used as a skeleton for the translation. 2. Alignment, where the matching parts of Q and S_i are found to identify the remaining translation gaps and translation alternatives for the mismatches are obtained from TM. 3. Combination, where the translation fragments found in the previous steps are aggregated into the final translation.

The matching stage involves maximising a similarity between the input Q and all source language sentences S_i in the example base. The most widely used similarity measures in EBMT matching are based on the edit distance (ED), also known as Levenshtein distance (LD) (Levenshtein, 1966). Thus, the matching step of an EBMT system is a time-consuming process with runtime depending on the complexity of the similarity measure and the size of the example base. EBMT systems can usually only handle a moderate size example base in the matching stage. However, using a large example base is important to ensure high quality MT output. In order to make MT applicable for larger example bases while improving or maintaining its speed, we investigate different approaches to efficient approximate sentence matching: a) sequential algorithms for the computation of the similarity, where speed improvements are based on limiting the number of symbol comparisons (e.g. cut-off heuristics); b) data structures, where a traversal of the structure can be employed to compute the similarity; c) sentence indexing and retrieval, where aspects of similarity are traditionally modeled by factors based on frequencies such as *tf* and *idf*, but in the case of approximate sentence retrieval should also include word order and position.

We demonstrate that sequential (brute-force) approximate matching becomes too expensive for large example bases. Using in-memory data structures is efficient with respect to runtime, but requires much more memory. Information retrieval based on standard term weighting functions is not appropriate for finding best matches. Standard information retrieval (IR) is more efficient than sequential comparison, but not accurate enough in finding the most similar matches in top ranks.

We propose a hybrid, two-stage approach. First we retrieve sentences from the example base, scoring the results based on our proposed edit distance approximating retrieval model. In the second step, we rerank the results by their true LD score. Our approach thus restricts the edist distance computation to the set of top-ranked retrieved sentences instead of the full example base. The proposed retrieval model uses positional indexing and retrieval, reflecting three aspects of similarity: how similar term positions are, how similar the word order is, and how similar the sets of terms are. The approach is shown to be fast (i.e. efficient and more scalable), accurate in terms of mean reciprocal rank (MRR) (i.e. effective in retrieving approximate matches), and can yield even better translations than the sequential approaches (i.e. results in a higher BLEU score (Papineni et al., 2002)).

The rest of this paper is organized as follows: Section 2 introduces related work; Section 3 presents approaches to approximate sentence matching; Section 4 describes our proposed IR model; Section 5 explains our experimental setup, data and evaluation metrics; Section 6 presents and discusses the results. Section 7 concludes with an outlook on future work.

2 Related Work

Despite of a long history of research in IR and MT, there is still relatively little research on applying IR methods for MT. Two years before Levenshtein proposed the edit distance in 1966 (Levenshtein, 1966), Faulk (Faulk, 1964) argued that three aspects of similarity should be considered for approximate sentence matching for translation: 1. positional similarity of items (e.g. words occur in the same positions), 2. ordinal similarity (e.g. words have the same order), and 3. material similarity (e.g. the sets of words are similar). He investigates different similarity metrics in different languages and demonstrates that a high sentence similarity in one language correlates with a high similarity in the target languages. The edit distance has been widely used in diverse applications such as approximate name matching (Berghel and Roach, 1996), in the biomedical domain for the comparison of gene sequences (Yap et al., 1996), for spelling correction and dictionary search (Boytsov, 2011). and in music retrieval (Mongeau and Sankoff, 1990).

Improvements of the runtime complexity of the original algorithm include Ukkonen's cut-off heuristics (Ukkonen, 1985b,a) and Berghel and Roach's extension of this approach. Finally, Levenshtein automata (Schulz and Mihov, 2002) have been suggested as an efficient approach to spelling correction when an upper bound for the distance is known in advance. Due to the large number of different symbols (words) and because an upper bound for the distance is not known in advance, we did not investigate Levenshtein automata for the experiments described in this paper.

Alternative approaches to approximate matching include q-grams (Gravano et al., 2001) (i.e. word n-grams), variants of the longest common subsequence (Lin et al., 2011), and affine gap alignment (Needleman and Wunsch, 1970). Navarro (Navarro, 2001) presents an excellent survey on different approaches to approximate string matching.

The standard application of MT in IR is in Cross-language IR, where given a query in the source language, documents in a target language have to be retrieved (Di Nunzio et al., 2008). IR techniques have been applied to machine translation only recently. Hildebrand et al. (Hildebrand et al., 2005) apply IR to improve the quality of the training data for a statistical MT system. They adapt the language model for translation by selecting similar sentences from a large training corpus for training data and experiment with *tf-idf* and cosine similarity and Okapi's BM25 model (Robertson et al., 1998), finding no significant difference in their performance. They conclude that adaptation of the LM is helpful in improving translation quality. Similar to the experiments described in this paper, Koehn et al. apply IR and use a combination of q-gram matching and A* pruning (Koehn and Senellart, 2010). However, they do not report individual results for the retrieval effectiveness, only optimize the speed of approximate matching, and do not report the effect of applying their matching approach to TM in detail. They achieve a processing time per query of 247ms for sentence matching in the JRC-Acquis corpus and 4.3ms in a smaller corpus with product information. Dandapat et al. (Dandapat et al., 2012) investigate two methods to achieve scalable EBMT. First, they try bucketing sentences by length to limit the number of sentences in the example base that have to be compared against the input, assuming that similar sentences will have similar length. Thus, the best match is not guaranteed to be found when this heuristics is used. Their second method includes IR based on standard language modeling (LM), but individual results for the IR stage are not reported. For the retrieval experiments described in this paper, we use LM as a baseline. We investigate parameter settings and preprocessing options to obtain the best baseline for our proposed approach.

The application of edit distance for problems such as spell checking is different to its application in EBMT and TM in several aspects: 1. Typically, character sequences (i.e. strings) are considered for comparison. For our experiments, we aim at computing the similarity for sequences of symbols or tokens (i.e. words and all punctuation marks). 2. Given a large dictionary for approximate *string* matching, the distance of an input word to some word in the dictionary is usually quite low. This does not have to be the case for approximate *sentence* matching, where the size of the alphabet (the number of possible symbols) is much higher (i.e. different characters vs. different words) and the minimum edit distance can also be quite high. Thus, a general minimum edit distance threshold can not be specified in practice. 3. Providing an upper bound for the edit distance can speed up the computation considerably. For the application we consider here, no upper limit for the number of mismatches is known in advance. 4. We are interested in finding all matches with the highest similarity as opposed to finding a single close match or one best match. For the experiments described in this paper, we did not employ heuristics such as bucketing or assuming an upper bound for the edit distance to speed up processing. These methods can actually be utilized to reduce runtime for our proposed approach even more, but at the cost of the loss of accuracy.

An extension to language modeling, known as positional language modeling (PLM), includes a proximity heuristic rewarding a document where matched query terms occur close to each other (Lv and Zhai, 2009). The PLM favours documents where the query terms appear in the same order as that in the query. In PLM, relative term positions are modelled via their term context. In constract, our proposed retrieval model for approximate sentence matching takes into account the absolute term positions.

3 Approaches for Approximate Sentence Matching

Approximate sentence matching (ASM) is the problem of finding the sentences with the highest similarity to a given inpuit sentence in a collection of sentences. The matching stage of EBMT can be considered as an instance of ASM. It identifies sentence pairs (S_i, T_i) from the example base where S_i closely matches with the input sentence Q. The EBMT system considers the translation T_i where Q has maximum similarity to the source sentence S_i to build a skeleton for translation of the input. From the perspective of IR, we are trying to find sentences (documents) in the example base (document collection) which have a high similarity (document score) with the input sentence (query). The results are then used in the alignment and recombination stage of EBMT to produce a translation. The set of sentences with the highest Levenshtein score (LS), i.e. the highest fuzzy match score, to the input sentence is computed by a sequential approach and corresponds to the set of relevant documents. Relevance assessment for IR is typically based on manual assessment of pooled results, whereas in the experiments described in this paper, the relevant (correct) results are determined based by a sequential computation of fuzzy match scores. We introduce similarity metrics between sequences Q and D with length $|Q|$ and $|D|$, where i and j denote an index in a sequence and Q_i and D_j denote the ith and jth symbol in sequence Q and D.

3.1 Sequential Search

Levenshtein Distance Algorithms (LD$_{WF}$ and LD$_{BR}$). The Levenshtein distance or edit distance is typically employed to calculate a word-level “fuzzy-match” score to find the closest matching source language sentence in the example base. The edit distance for two sequences Q and D is defined as the minimum number of edit operations, i.e. symbol insertions, deletions and substitutions, needed to transform Q into D (Levenshtein, 1966). The sequential computation of the distance can be computationally expensive, because the Levenshtein algorithm has a runtime complexity of $O(|Q|$ x $|D|)$, which has to be calculated for every sentence in the example base. The straightforward solution to compute the edit distance is via dynamic programming, where a $|Q|$ x $|D|$ matrix is filled following a recursive schema. The value in the bottom-rightmost cell of the matrix is the edit distance $LD(Q, D)$ (Wagner and Fischer, 1974). A corresponding normalized Levenshtein similarity score (fuzzy match score) is computed as

$$LS(q, d) = 1 - LD(Q, D)/\max(|Q|, |D|) \qquad (1)$$

This score is used for all approaches described here which are based on computing the exact LD.

Several improvements have been proposed to improve runtime complexity. Ukkonen's Enhancend Dynamic Programming Approximation algorithm (Ukkonen, 1985b) for computing edit distance has the worst case complexity $O(|Q|$ x $B)$, where B is an upper bound of the edit distance. The improvement results from the fact that $distance(i, j)$ values are non-decreasing along any given diagonal. Only those $distance(i, j)$ values p for which i is the highest numbered row on which p occurs on diagonal k for a given threshold k have to be computed. The modified Berghel-Roach algorithm (Berghel and Roach, 1996) is an extension of Ukkonen's cut-off heuristic. It achieves 42% speed-up compared to Ukkonen's approach for name matching and is 79% faster than the Wagner-Fischer algorithm. We employ the Berghel-Roach algorithm (LD$_{BR}$) to investigate speed-up by an algorithm with lower runtime complexity. As a baseline for runtime, we compute the edit distance based on the Wagner-Fischer algorithm (LD$_{WF}$).

Longest Common Subsequence (LCS). Our experiments to improve MT with IR methods are based on the assumption that the sentence with the minimum edit distance in the source language is a good template for its translation. This assumption is widely accepted and has been validated for translation memory (TM) (Sikes, 2007). We compare our experimental results for the edit distance with corresponding results for the longest common subsequence (LCS) (Gusfield, 1997). The LCS is the longest shared subsequence of – not necessarily consecutive – symbols in two sequences. The corresponding score is computed by replacing the edit distance LD in Equation 1 with the LCS and calculating the scores by iterating over all examples in the example base.

3.2 Data Structures

Efficient data structures have been proposed for fast approximate lookup operations in dictionaries (e.g. for spelling correction), but to the best of our knowledge, the following approaches have not been investigated for approximate sentence matching, i.e. for computing the edit distance for sequences of words. In fact, the existing implementations consider only characters as symbols, while our implementation abstracts from that view and allows any type of symbol as the basic element of a sequence.

Tries (TR). A trie is an ordered tree data structure that can be used to store key-value pairs where the keys are sequences of symbols (Gusfield, 1997). Each node in a trie stores a single symbol of the corresponding sequence and a node represents the prefix of the key on the path of the root up to that node. All the children of a node have a common prefix of the sequence associated with that node, and the root represents the empty sequence. Values are associated with nodes corresponding to the end of a sequence, i.e. leaf nodes and some inner nodes. The main idea to efficiently compute the edit distance with tries is to compute only the part of the distance matrix up to the length of the current prefix, so that redundant computations are avoided for sequences sharing the same prefix. For our experiments, we regard sentences comprised of tokens as sequences of symbols and store the corresponding document ID (i.e. sentence ID) as a value.

Ternary search trees (TST). Ternary search trees (Bentley and Sedgewick, 1997) are tree structures where each node has three children. Similar to hash tables, ternary search trees can be employed as an associative structure to store key-value pairs (here: pairs of sentence and document ID). Each node of a ternary search tree stores a single symbol for comparison with a symbol of a search key, and pointers to three children which determine which subtree to search next, based on the result of a comparison, i.e. lower, equal or higher lexicographical order. As for tries, only part of the edit distance matrix – a single row – is computed at a given node.

BK-trees (BKT). Burkhard and Keller (Burkhard and Keller, 1973) proposed BK-trees for efficient file searching. A BK-tree is a metric tree adapted to discrete metric spaces, defined by a distance metric $D(x, y)$. A BK-tree can be formed as follows. An arbitrary element a is selected as the root node. The root node may have zero or more subtrees. The kth subtree is recursively built of all elements y such that $D(x, y) = k$. As BK-trees support distance metrics, and not similarities, we directly employ the edit distance as a metric. To improve runtime, we use a distance threshold of 2, i.e. subtrees exceeding the threshold distance are not visited. This setting was obtained empirically and chosen to yield the best results in terms of effectiveness and efficiency. Lower values result in finding fewer correct results, higher values more than double the runtime because more nodes have to be visited. Note that this cutoff heuristic results in lower runtime but also lower effectiveness.

3.3 Information Retrieval Approaches

The problem with brute force, sequential computation is that it involves a linear search for the closest match sentence by computing edit distance between the given query and each sentence in the example base. The time complexity is thus $O(N)$, N being the number of sentences in the collection, which clearly makes the brute-force approach infeasible for large collections. In-memory data structures could be much faster in LD computation, but have high space complexity, i.e. require a huge memory and are thus not scalable for very large collections.

Standard IR models. Information retrieval (IR) is concerned with finding relevant documents from a document collection, given a query (Manning et al., 2008). IR involves computing document scores for a given query by aggregating term scores from the local importance of a term in a document (e.g. the term frequency, *tf*) and a global importance factor of the term in the document collection (e.g. the inverse document frequency, *idf*).

Generally speaking, a retrieval model aims to compute the similarity between a document and a query. As a simplification, queries and documents can be defined as vectors over the vocabulary term space, so that the similarity can be computed as a dot product between query and document term vector (see Equation 2).

$$sim(d, q) = \sum_{i=1}^{|V|} d_i q_i \tag{2}$$

In standard IR, similarity features such as the word position in the query or a document are typically ignored. Features such as word ordering are rarely modeled.

IR makes use of *inverted list* structures which are a combination of in-memory data structures and file structures. The list of documents where a term occurs, namely the *postings list* for a term is loaded into memory from files hashed by the term identifier. This combination results in very fast retrieval and also makes retrieval scalable to very large collections.

The inverted list data structure is suitable for computing the query-document similarity, because document vector term weights i.e. the d_i values are read from the postings list of the ith query term, and accumulated over all query terms to calculate the final similarity. The computational complexity of the similarity is thus $O(\sum_{i=1}^{n} s_i)$ where n is the number of query terms and s_i is the size of the postings list for the ith query term. In practise, n is a small constant and $s_i << N$, where N is the number of document in the collection. Hence the method is very fast.

Although the inverted list structure of IR looks promising, it can not be directly applied to the ASM problem because of basic differences: In standard IR, documents are much longer than queries, compared to ASM, where documents and queries have similar length. Preprocessing techniques such as removing stopwords and eliminating punctuation, and applying a stemmer are common for IR and reduce the index size and the retrieval time, because highly frequent terms such as stopwords are excluded from the inverted lists.[1] In ASM, keeping all terms (i.e. words and punctuation) is important for exact matching of symbols and to retain the syntactic sentence structure for translation. Relative and absolute term positions are usually ignored in standard IR (except for phrase search, which partially models relative term position). In ASM, it might be important to implicitly retain word order as a similarity factor by matching n-grams.

The LD computation for ASM cannot directly be performed, i.e. it is not possible to compute the exact edit distance between a document and a given query within an IR application. One thus has to strive for designing a similarity function which produces rankings as close as possible to the ranking as computed by the LD computation. However, this proves difficult because standard IR similarity scores work on the principle of material similarity, i.e. a document is potentially ranked higher if it contains a higher number of query terms. For each ith query term match we add the contributing factor of $d_i q_i > 0$ to the document score. However, material similarity alone may not be a good approximation for ranking based on LD.

Towards better sentence similarity metric for LD approximation. Three measures of similarity are crucial for ASM, namely the positional, ordinal and material similarity (Faulk, 1964). A direct application of standard IR methods for ASM can only estimate the last one, i.e. the material similarity. In addition to material similarity, it is required to consider the following, particularly for finding closest edit distance sentences for EBMT (Somers, 2003). Firstly, a sentence from the example base with a partial exact match of words (identical surface forms of words) in the query sentence is helpful because the longer the matched portion, the higher the likelihood is of generating a target translation of good quality. Secondly, identical term positions between Q and D indicate structural similarity, which is helpful in inferring the inherent reordering between D and the translation. Thirdly, similarity in length between Q and D requires substitution operations between mismatched portions during the *recombination* stage of EBMT. This is advantageous over insertion, since the positions in the translation of D to substitute are exactly known during recombination. To the best of our knowledge, a retrieval model incorporating aspects of material, ordinal, and positional similarity has not yet been proposed for ASM.

[1]Note that while we aim at reducing runtime, we still include stopwords and punctuation for indexing in our main experiments, which is contrary to a standard IR setup.

4 ASM Retrieval Model

The objective of approximate sentence matching is to estimate the edit distance ranking as accurately as possible without computation of the actual distances. Let Q be a query sentence and D a sentence for which we need to estimate its edit distance from Q.

Language modeling (LM) is a state-of-the-art retrieval model (Ponte and Croft, 1998), where a document D is scored by the posterior probability of generating the given query Q, i.e. $P(D|Q)$ (Hiemstra, 2000). This in turn is estimated at indexing time from the prior probability $P(Q|D)$ using the assumption that the query terms can be generated independently from a document D by a linear combination of two events of either generating a query term q from D (i.e. the *tf* component) with probability λ or generating it from the collection (i.e. the *idf* component) with probability $(1-\lambda)$, as shown in Equation 3 ($cf(q)$ and cs denote the collection frequency of term q and the collection size respectively). Equation 3 does not take into account the relative (or even absolute) positions of documents and query terms.

$$P(D|Q) \propto P(D)P(Q|D) = P(D)\prod_{q\in Q} P(q|D) = P(D)\prod_{q\in Q} \lambda\frac{tf(q,D)}{|D|} + (1-\lambda)\frac{cf(q)}{cs} \tag{3}$$

We propose an extension of the LM retrieval model for ASM, which we call LM-ASM. In contrast to standard LM, the probability of generating the query Q from a document D in case of LM-ASM has two components: i) $P_{len}(Q|D)$ which denotes how close is the length of Q to the length of D; and ii) $P_{pos}(Q|D)$ representative of how likely it is for a query term to belong to the same absolute position where it occurs in D.

The first likelihood component minimizes the likely number of insertions and deletions while transforming D to Q and is given by Equation 4. Note that this likelihood function decreases with an increase in the absolute difference of the lengths of D and Q, attaining a maximum for $|D| = |Q|$.

$$P_{len}(Q|D) = \min(|Q|,|D|)/\max(|Q|,|D|) \tag{4}$$

To model the generation of a query term in its absolute position from the corresponding absolute position of that term in a document D, we use the following notation. Let $pos(q,D)$ be the set of absolute positions of a term q in a document D. In contrast to standard LM, for ASM, we also need to consider the current position of a query term. Let q_i be the query term at position i. Thus, even if the same query term q occurs in multiple positions in a query, we compute $P_{pos}(q_i|D)$ for all positions i where q occurs.

$$P_{pos}(q_i|D) = 1 \,/ \min_{j\in pos(q_i,D)} (|j-i|+1) \tag{5}$$

In Equation 5, $|j-i|$ denotes the absolute value of the minimum difference of term positions.

The final probability of $P(Q|D)$ is thus given by multiplying the two components $P_{len}(.)$ and $P_{pos}(.)$ of Equations 4 and 5 into Equation 6.

$$P(Q|D) = P_{len}(Q|D)P_{pos}(Q|D) = \frac{\min(|Q|,|D|)}{\max(|Q|,|D|)}\prod_{i=1}^{|Q|}\Big(\frac{1}{\min_{j\in pos(q_i,D)} |j-i|+1}\Big) \tag{6}$$

Analogous to LM, in order to avoid underflow for multiplications of small numbers, we implement the positional score with the log transform (omitted for brevity).

A closer look at Equation 6 reveals that for every matching query term in a document D, we include the reciprocal of the absolute differences of the term positions in the score. The higher this difference, the lower is the similarity component being aggregated. In case a query term does not occur in D, nothing is added to the score. We propose to use this approach for approximate sentence retrieval and call it LM-ASM.

5 Experimental Setup

Data. We conduct experiments to report the accuracy of our EBMT approach for English-Turkish (EN-TR) and English-French (EN-FR) translation tasks. In order to compare the translation performance of our approaches, we use the EBMT system described in (Dandapat et al., 2012), which follows the framework in (Nagao, 1984) as a baseline. We do not combine our EBMT approach with an SMT system, as we focus on effectiveness and efficiency evaluation of IR for ASM to improve the EBMT component.

The two data sets used for all our experiments represent two language pairs with parallel data of different size and type. Statistics for the data sets, IWSLT 09[2] and European Medicines Agency[3] (EMEA), are shown in Table 1. The original EMEA corpus comprises approximately 1M sentences, including many duplicates. We discard duplicates and consider only sentences with unique translation equivalents.

Name	Domain	Language pair	Example base	Avg. sentence length	Test data
IWSLT 09	Travel	EN-TR	19,972 sentences	9.5 words	414 sentences
EMEA	Medicine	EN-FR	250,806 sentences	18.8 words	10,000 sentences

Table 1: Statistics on the two evaluation data sets.

Evaluation Objectives. Our experiments focus on different research questions: *Which approach is the most efficient and scales up to large example bases for EBMT?* To investigate this question, we conduct experiments using different algorithms and data structures for EBMT matching as described in Section 3 and 4 and compare them with our proposed two stage approach based on IR methods and reranking IR results.

Which approach has the highest accuracy when a trade-off between efficiency and effectiveness becomes necessary? Naturally, sequential computation will yield the highest accuracy (e.g. MRR) of results. However, when using IR methods, we expect a drop in effectiveness because the edit distance similarity is only approximated by the scoring method. Related to this question is the aspect of preprocessing (word matching), e.g. *Should stopwords or punctuation be removed or should a stemmer be applied?* We expect that stopwords and punctuation are actually important for approximate sentence matching and that stemming would decrease retrieval effectiveness. We perform experiments using different preprocessing methods and indexing word-level n-grams.

Which approach leads to the highest translation quality? With the sequential computation, we will find all and only exact approximate matches (i.e. all sentences with the exact maximum edit distance score LS). Using IR methods, potentially only a subset of all "relevant" sentences [4] will be identified. A related question is *How many documents should be retrieved in the IR stage?* One objective in ASM is to restrict the number of retrieved documents to a small number, typically 10 to

[2]http://mastarpj.nict.go.jp/IWSLT2009/2009/12/downloads.html
[3]http://opus.lingfil.uu.se/EMEA.php
[4]Relevant in this context means the best translation template in the target language.

20, since it is more efficient to retrieve a small number of sentences; and it is more efficient to rerank a small set of sentences by the true LS (e.g. by computing $LS(Q, D)$ for the top-ranked results). For example, retrieving 100 or more documents in the retrieval step could result in higher runtime, because the set of retrieved documents has to be reranked again (unless only a single document is retrieved). We try to determine the number of documents needing to be retrieved empirically to achieve this trade-off, i.e. achieve a satisfactory MT performance without sacrificing computation speed.

Evaluation Metrics. The experiments described in this paper aim at maintaining or improving three aspects of performance for large example bases: *efficiency*, *effectiveness*, and *translation quality*, described as follows. 1. *Efficiency:* A lower runtime implies that larger example bases can be used and the MT system becomes more scalable. We report the time for reading and indexing the documents in seconds (IT), computation time (CT), and the average time per sentence (AT), exlcuding indexing time. 2. *Effectiveness:* As some similarity scores and the proposed retrieval approach approximate the edit distance, instead of actually computing it, the accuracy of finding correct matches with minimum LD (more specifically: with maximum LS) is measured. We employ IR metrics such as mean average precision (MAP) and the mean reciprocal rank (MRR) (Voorhees, 1999). The significance of MRR is that the closer the MRR is to 1, the fewer documents need to be reranked using their true edit distances. We also report the number of queries in the test set with a reciprocal rank (RR) of 1 (i.e. the top ranked document is relevant) and with $|RR > 0|$, i.e. the result set contains at least one relevant result. A reciprocal rank of 1 means that a correct result is already at rank 1, and $RR > 0$ implies that there is at least one correct result contained in the set of retrieved results. In addition, we report the number of retrieved documents (#ret) and the number of relevant documents retrieved, i.e. *recall* (#rel_ret). Achieving high recall is important to enable an additional level of processing to choose a particular sentence for the EBMT matching phase among a set of candidate sentences with equal LS values with respect to the input sentence. Since developing an efficient tie-breaking heuristic is outside the scope of this paper and we are not aware of any such existing work, we simply resolve ties by choosing the sentence with minimum identifier for all our experiments. However, we include recall and MAP as evaluation metrics to justify the relative usefulness of a system under the presence of a tie-breaking mechanism. 3. *Translation quality:* Finally, we compute how good the final translation output is, based on the standard MT evaluation score BLEU (Papineni et al., 2002).

Relevance Judgements. Since the objective is to approximate edit distance scores, the target documents (relevant documents) are those with the maximum LS scores with respect to the query. The "relevance judgements" are thus obtained from computing the sentence pairs in the example base with maximum LS score (see Equation 1) for a query, presuming that all sentences with the minimum distance (or highest sentence similarity) are correct or relevant. This can lead to a high number of relevant results for queries which have many exact or near-matches in the data. For the EN-TR data, there are 4.74 relevant results per query on average; queries in the EN-FR data have 16.38 relevant results per query on average. To resolve these ties for EBMT, the highest scoring document with the lowest document ID is selected for subsequent translation stages.

Test System. The test system is a standard PC with a 3.16 GHz Core 2 Duo CPU and 8 GB RAM. The retrieval engine is SMART[5], which was modified to support positional indexing and language modeling. In the postings list of every term, we store the document ID for each document the term

[5]ftp://ftp.cs.cornell.edu/pub/smart/

Run	Parameter			IR effectiveness			
Name	SR	ST	n	#ret	#rel_ret	MAP	MRR
tf	N	N	1	20700	615	0.289	0.289
tf-idf	N	N	1	20700	710	0.338	0.346
LM	N	N	1	20700	**1270**	**0.593**	**0.617**
LM	N	Y	1	18944	1148	0.513	0.547
LM	Y	Y	1	13349	296	0.180	0.250
LM-ASM	N	N	1	20700	**1295**	**0.638***	**0.699**
LM-ASM	N	Y	1	18944	1174	0.579*	0.641*
LM-ASM	Y	Y	1	13349	265	0.183	0.617*
LM	N	N	2	20700	1414	**0.658**	0.688
LM-ASM	N	N	2	20700	**1482**	0.650	**0.733***

Table 2: IR results on EN-TR data for retrieval of 50 documents per query. Parameter settings include stopword removal (SR), stemming (ST), and the use of n-grams.

appears in and the absolute position of that term in that particular document. The EBMT system is an implementation of the translation by analogy approach described in (Nagao, 1984).

6 Experimental Results

Effect of Preprocessing. ASM is different from standard IR. Therefore, we explore how to optimize the IR settings for the ASM problem. We have argued that standard IR pre processing such as stopword removal and stemming may not be suitable for ASM. Table 2 shows the results for different IR approaches. In this section, we are only interested in the retrieval effectiveness which is measured in terms of how close the retrieved set is with respect to the reference set of minimum edit distance sentences from the training set for each test sentence. Results are obtained by retrieving 50 sentences. The results in Table 2 can be interpreted as follows. Simple retrieval methods such as raw term frequency (*tf*) or *tf-idf* do not perform well for ASM. In contrast to standard IR, stopword removal and stemming decrease performance. The number of retrieved documents (#ret) is also lower compared to the runs where neither stopword removal nor stemming is performed, because some query sentences comprise of only stopwords e.g. "I am sorry". Stemming also degrades retrieval performance. So the the question *'Should stopwords or punctuation be removed or should a stemmer be applied?'* can be answered with no. In our experiments, all forms of preprocessing degrade performance.

We employ LM as the baseline for our retrieval experiments. In order to obtain the best LM baseline, we conducted experiments with varying values for the λ parameter in the interval $[0, 1]$. Selecting $\lambda = 0.99$ yields the highest MAP (0.593). This explains that *idf* is not important for ASM. Normalized *tf* (i.e. $\frac{tf(t,D)}{|D|}$) suffices to approximate LS (see Equation 3).

Our proposed method LM-ASM produces significantly better results both in terms of MAP and MRR as compared to LM[6]. LM-ASM, unlike LM, takes into consideration the term position differences between document words and the given query words, thus better estimating the number of edit operations and hence the edit distance. Note that LM-ASM does not rely on collection statistics such as *idf*.

[6]A '*' denotes significantly better result of LM or LM-ASM over its counterpart with the same experimental settings.

Name	Efficiency (runtime)			IR effectiveness						MT
	IT [s]	CT [s]	AT [s]	#ret	#rel_ret	MAP	MRR	|RR=1|	|RR>0|	BLEU
LD$_{WF}$	0.959	227.709	0.550	1962	1962	1.000	1.000	414	414	21.71
LD$_{BR}$	0.530	53.380	0.129	1962	1962	1.000	1.000	414	414	21.71
LCS	0.403	52.328	0.126	2143	1679	0.814	0.858	340	378	21.48
TR	0.958	30.816	0.074	1962	1962	1.000	1.000	414	414	21.71
BKT	2.130	13.147	0.032	1301	795	0.536	0.633	262	262	18.25
TST	0.845	12.672	0.031	1962	1962	1.000	1.000	414	414	21.71
tf-idf$_1$	0.696	0.340	0.001	414	87	0.182	0.210	88	88	9.94
LM$_1$	0.671	0.338	0.001	414	190	0.316	0.459	191	191	17.42
LM-ASM$_1$	0.617	1.743	0.004	414	239	0.347	0.577	240	240	18.78
LM-2$_1$	0.914	0.467	0.001	414	232	0.377	0.560	233	233	21.10
LM-ASM-2$_1$	0.840	2.460	0.006	414	262	0.367	0.642	266	266	20.29
tf-idf$_{10}$	0.696	0.364	0.001	4140	380	0.312	0.216	43	218	20.29
LM$_{10}$	0.671	0.596	0.001	4140	788	0.553	0.611	191	368	21.69
LM-ASM$_{10}$	0.617	1.963	0.005	4140	902	0.603	0.670	240	376	21.09
LM-2$_{10}$	0.914	1.271	0.003	4140	827	0.619	0.684	233	377	21.15
LM-ASM-2$_{10}$	0.840	3.264	0.008	4140	893	0.609	0.730	266	377	21.76
tf-idf$_{50}$	0.663	0.458	0.001	20700	1272	0.514	0.549	164	391	21.40
LM$_{50}$	0.671	2.195	0.005	20700	1417	0.593	0.617	191	405	21.51
LM-ASM$_{50}$	0.617	3.562	0.009	20700	1390	0.638	0.701	240	400	21.58
LM-2$_{50}$	0.914	2.311	0.006	20700	1414	0.658	0.688	233	409	21.39
LM-ASM-2$_{50}$	0.840	4.304	0.010	20700	1480	0.638	0.733	266	402	22.08

Table 3: Experimental results on EN-TR data set. Evaluation metrics include indexing time (IT), computation time (CT), and the average time per sentence (AT).

We also experimented with bi-grams ($n = 2$) on the best IR settings for the uni-grams i.e. without stopword removal and stemming. The results are shown in the last two rows of Table 2. It can be seen that use of 2-grams yields in significant improvement is obtained in the MRR (0.733 vs. 0.699) at the cost of an insignificant decrease in MAP. The reason for the improvement can be attributed to the better estimation of the word order and term overlap achieved by bi-grams.

Comparison of Approaches. Results for our experiments are shown in Table 3 and 4. Numeric indices in the run name indicate the number of documents retrieved per query. The improvement of the LM-ASM approach with bi-gram indexing over the baseline MT as obtained by brute-force sequential compuatation, is statistically significant with a reliability of 97%, for the EN-TR dataset, as seen by comparing the BLEU score of $\mathrm{LD_{WF}}$ (21.71) with that of LM-ASM-2_{50} (22.08). [7]

Results on the English-Turkish data set are shown in Table 3. We conduct selected experiments on the English-French data set (see Table 4) to investigate scalability and efficiency of the approaches on a larger example base. We briefly revisit the unanswered questions raised in Section 5.

Which approach is the most efficient and scales up to large example bases for EBMT? A sequential approach to compute the edit distance between an input sentence and all source language sentences in the example base is not feasible, because it requires too much run-time (columns CT and AT in

[7] This improvement is statistically significant as measured by the paired-bootstrap resampling (Koehn, 2004).

Name	Efficiency (runtime)			IR effectiveness						MT
	IT [s]	CT [s]	AT [s]	#ret	#rel_ret	MAP	MRR	|RR=1|	|RR>0|	BLEU
LD_{WF}	4.653	105813.952	10.581	163751	163751	1.000	1.000	10000	10000	48.42
LD_{BR}	7.355	48695.741	4.870	163751	163751	1.000	1.000	10000	10000	48.42
TR	30.611	27248.317	2.725	163751	163751	1.000	1.000	10000	10000	48.42
TST	8.004	13502.705	1.350	163751	163751	1.000	1.000	10000	10000	48.42
LM_1	8.293	10.546	0.001	100000	3457	0.322	0.410	3654	3654	35.18
LM-ASM$_1$	8.448	65.604	0.006	100000	3536	0.325	0.381	3804	3804	37.68
LM_{10}	8.293	55.570	0.006	100000	7314	0.436	0.410	3654	5927	41.35
LM-ASM$_{10}$	8.448	110.808	0.011	100000	7521	0.444	0.418	3804	6168	42.53
LM_{50}	8.293	93.831	0.009	100000	19949	0.431	0.445	3654	7053	44.36
LM-ASM$_{50}$	8.448	148.889	0.015	100000	26159	0.435	0.465	3804	7247	44.92

Table 4: Experimental results on EN-FR data set.

Table 3). The sequential brute force approach has the lowest efficiency, but can be improved by either using a metric that can be computed faster, such as LCS, or more efficient data structures such as TR and TST. Interestingly, LCS is a good approximation of the edit distance (as can be seen from the MT scores in the table) and requires a similar run time compared to LD_{BR}. Sequential approaches do not scale up well when using large example bases, even when efficient implementations of the Levenshtein algorithm are used (see Table 4). The use of data structures such as ternary search trees and tries is more efficient, but requires that the structures are kept in memory. Thus, these approaches have higher memory requirements. IR approaches are the most efficient, even when a reranking phase based on sequential computation of LS is included.

From the results in Table 4, we observe that when the sequential computation or the efficient data structures are used, the translation quality is high, but at the cost of a runtime of 1.4 seconds in the best case (TST) and 10.6 seconds in the worst case (LD_{WF}). This makes a close-to real-time translation nearly impossible. The use of IR yields a much better runtime, but performance depends on how many documents are retrieved in the initial stage. Note for all of these top-ranked documents, the true edit distance has to be computed to facilitate result reranking.

Which approach has the highest accuracy when a trade-off between efficiency and effectiveness becomes necessary? The best exact approach to compute the edit distance is the ternary search tree (see columns MRR), which however suffers from higher memory requirements because the data structure has to be kept in memory. As expected, standard IR approaches such as *tf-idf* do not perform well in finding approximate matches, due to missing constraints for the search (e.g. word ordering and word position). Additional reranking of results by the normalized similarity score (based on LD_{WF}) ensures that correct matches are at top ranks (see columns $|\mathrm{RR} = 1|$ and $|\mathrm{RR} > 0|$). The combination of IR followed by a reranking stage is efficient and effective and leads to a high translation quality (see columns AT, MRR, and BLEU in Tables 3 and 4).

Standard LM performs better than *tf-idf*, but does not take into account word positions. Positional ranking in LM-ASM manages to rank the sentences candidates with a high precision and takes into account word positions and word order. It consistently outperforms standard LM in terms of MAP, MRR, and recall (see Table 3), at a moderate cost of additional runtime per query.

Which approach leads to the highest translation quality? Sequential computation of the edit distance based scores yields the highest effectiveness and MT scores, together with all other approaches aiming at finding the exact results with minimum edit distance, i.e. TR and TST.

Surprisingly, we found that the MT scores for our proposed approach with bi-gram indexing are actually higher than the scores for the sequential computation. To explain why our proposed approach to ASM achieves a higher BLEU score than experiments based on sequential LD computation, we looked at some sentences in detail. It can be argued that choosing the minimum edit distance sentence globally from the example base may not necessarily lead to the best translation. For the English input sentence *"where can i buy accessories"* (TR: *nereden aksesuar alabilirim*), the most similar sentence retrieved by sequential LD computation from the full example-base is *"where can i buy plates"* (TR: *tabak almak istiyorumi*). Since, ties in LD are resolved by taking the sentence with minimum document identifier, the IR method LM-ASM-2_{50} retrieves a different sentence – with the same LS – namely *"where can i buy stockings"* (TR: *nereden çorap satın alabilirim*). The target language sentence parallel to the source sentence candidate found by LD_{WF} compared to *"tabak almak istiyorumi"* does not have a single word common to the reference translation (TR: *nereden aksesuar alabilirim*), whereas the sentence retrieved by LM-ASM-2_{50} has two words in common, as shown with bold-face. The translation template is thus better for the latter which is also reflected in a higher overlap of the final translation output (TR: *nereden aksesuarı satın alabilirim*) with the reference output thus resulting in a higher BLEU score for LM-ASM-2.

How many documents should be retrieved in the IR stage? The results in Table 3 and 4 show that there is a trade-off between speed and quality of translation when using IR: A very fast translation can be achieved by simply retrieving a single sentence, thus completely eliminating the need for a reranking stage, but this depends on a high MRR of the initial retrieval. Retrieving more documents results in a higher translation score because for more sentences the best approximate match is found. However this also results in a higher processing time per sentence, because the edit distance has to be computed for more sentences. In practice, this implies that the user of an EBMT system can control whether he is more interested in a high-quality or in a fast translation.

Thus, it proves important to choose source-side sentences with care. Edit distance can be a used as a first level filter to choose a set of candidate sentences. A second level choice is required to carefully break the ties to ensure selection of the best source side sentence with an equivalent target language sentence *close* to the reference translation.

7 Conclusions

IR can benefit in solving problems beyond the search for information. In this paper, we have described how the adaptation of an information retrieval model to fit specific requirements of approximate retrieval can help to make EBMT more scalable, efficient, and even produce better translations. To solve the problem of approximate sentence matching, we proposed and evaluated LM-ASM, a novel IR model which incorporates three aspects of similarity, namely positional, ordinal, and material similarity. Our approach demonstrates the usefulness of IR for other tasks. The evaluation covers three aspects: IR metrics, metrics of the problem domain (e.g. BLEU score), and non-functional requirements (e.g. the runtime). We found that our proposed approach for ASM, LM-ASM, outperforms sequential computation of scores and in-memory data structures in terms of runtime, while losing almost none of the translation quality.

As part of future work, we plan to investigate alternatives for generating the relevance assessments based on the BLEU score obtained when using a potentially relevant sentence as translation template. This could lead to similarity metrics and IR models that better approximate the MT quality.

References

Bentley, J. L. and Sedgewick, R. (1997). Fast algorithms for sorting and searching strings. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on discrete algorithms*, pages 360–369, Philadelphia, PA, USA. SIAM.

Berghel, H. and Roach, D. (1996). An extension of Ukkonen's enhanced dynamic programming ASM algorithm. *ACM Trans. Inf. Syst.*, 14(1):94–106.

Boytsov, L. (2011). Indexing methods for approximate dictionary searching: Comparative analysis. *J. Exp. Algorithmics*, 16:1–91.

Burkhard, W. A. and Keller, R. M. (1973). Some approaches to best-match file searching. *Communications of the ACM (CACM)*, 16(4):230–236.

Dandapat, S., Morrissey, S., Way, A., and van Genabith, J. (2012). Combining EBMT, SMT, TM and IR technologies for quality and scale. In *Proceedings of the Joint Workshop on Exploiting Synergies between Information Retrieval and Machine Translation (ESIRMT) and Hybrid Approaches to Machine Translation (HyTra)*, pages 48–58, Avignon, France. ACL.

Di Nunzio, G. M., Ferro, N., Mandl, T., and Peters, C. (2008). Clef 2007: Ad hoc track overview. In *Advances in Multilingual and Multimodal Information Retrieval, 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2007*, volume 5152 of *LNCS*, pages 13–32. Springer.

Faulk, R. D. (1964). An inductive approach to language translation. *Communications of the ACM (CACM)*, 7(11):647–653.

Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., Pietarinen, L., and Srivastava, D. (2001). Using q-grams in a DBMS for approximate string processing. *IEEE Data Eng. Bull.*, 24(4):28–34.

Gusfield, D. (1997). *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Hiemstra, D. (2000). *Using Language Models for Information Retrieval*. PhD thesis, Center of Telematics and Information Technology, AE Enschede.

Hildebrand, A., Eck, M., Vogel, S., and Waibel, A. (2005). Adaptation of the translation model for statistical machine translation based on information retrieval. In *Proceedings of EAMT*, pages 133–142.

Koehn, P. (2004). Statistical Significance Tests for Machine Translation Evaluation. In *EMNLP 2004*, pages 388—395.

Koehn, P. and Senellart, J. (2010). Fast approximate string matching with suffix arrays and A* parsing. In *Meeting of the Association for Machine Translation of the Americas (AMTA)*.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Lin, H.-J., Wu, H.-H., and Wang, C.-W. (2011). Music matching based on rough longest common subsequence. *Journal of information science and engineering*, 27:95–110.

Lv, Y. and Zhai, C. (2009). Positional language models for information retrieval. In *Proceedings of SIGIR '09*, pages 299–306.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Mongeau, M. and Sankoff, D. (1990). Comparison of musical sequences. *Computers and the Humanities*, 24:161–175.

Nagao, M. (1984). A framework of a mechanical translation between Japanese and English by analogy principle. In *Artificial and human intelligence*, pages 173–180, New York, NY, USA. Elsevier North-Holland, Inc.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *SIGIR'98*, pages 275–281. ACM.

Robertson, S. E., Walker, S., and Beaulieu, M. (1998). Okapi at TREC-7: Automatic ad hoc, filtering, VLC and interactive track. In Harman, D. K., editor, *The Seventh Text REtrieval Conference (TREC-7)*, NIST Special Publication 500-242, pages 253–264, Gaithersburg, MD, USA. National Institute of Standards and Technology (NIST).

Schulz, K. and Mihov, S. (2002). Fast string correction with Levenshtein-automata. *International journal of document analysis and recognition*, 5:67–85.

Sikes, R. (2007). Fuzzy matching in theory and practice. *Multilingual*, 18(8):39–43.

Somers, H. (2003). *An overview of EBMT*. Kluwer. In Michael Carl and Andy Way (eds) Recent advances in Example-Based Machine Translation, Dordrecht.

Ukkonen, E. (1985a). Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118.

Ukkonen, E. (1985b). Finding approximate patterns in strings. *J. Algorithms*, 6(1):132–137.

Voorhees, E. M. (1999). The TREC-8 question answering track report. In *In Proceedings of TREC-8*, pages 77–82.

Wagner, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *J. ACM*, 21(1):168–173.

Yap, T. K., Frieder, O., and Martino, R. L. (1996). *High performance computational methods for biological sequence analysis*. Kluwer.