# A memory-based classification approach to marker-based EBMT

**Antal van den Bosch**
Dept. of Language and Information Sciences
Tilburg University, P.O. Box 90153
NL-5000 LE Tilburg, The Netherlands
Antal.vdnBosch@uvt.nl

**Nicolas Stroppa** and **Andy Way**
School of Computing
Dublin City University
Glasnevin, Dublin 9, Ireland
{nstroppa,away}@computing.dcu.ie

## Abstract

We describe a novel approach to example-based machine translation that makes use of marker-based chunks, in which the decoder is a memory-based classifier. The classifier is trained to map trigrams of source-language chunks onto trigrams of target-language chunks; then, in a second decoding step, the predicted trigrams are rearranged according to their overlap. We present the first results of this method on a Dutch-to-English translation system using Europarl data. Sparseness of the class space causes the results to lag behind a baseline phrase-based SMT system. In a further comparison, we also apply the method to a word-aligned version of the same data, and report a smaller difference with a word-based SMT system. We explore the scaling abilities of the memory-based approach, and observe linear scaling behavior in training and classification speed and memory costs, and log-linear BLEU improvements in the amount of training examples.

## 1 Introduction

The *decoding* step in machine translation is typically seen as an information-theoretic, Bayesian, stochastic process; the term refers directly to this. A natural alternative to stochastic decoding is discrete decoding. Knowledge-based methods could be characterized as such, but also inductive methods that learn to classify into discrete output spaces. Most of these methods, such as rule learners, decision tree learners, and $k$-nearest neighbor classifiers, do offer means to include non-discrete statistical weighting metrics reflecting the likelihood of events, but the essential difference with stochastic methods is that they produce discrete predictions, not probability distributions.

The search space induced by stochastic methods is usually too big to be completely explored; this space is typically reduced using approximative search strategies (such as beam search), while discrete methods in principle have no such requirement since they can directly produce the eventual discrete output, the translation. This would appear to be a reason to use discrete methods for decoding in machine translation. However, an immediate objection to that is the prohibitively large output space in which to distinguish the few correct outputs among the possibly hundreds of thousands to millions of possible outcomes. Even if the task is decomposed into the individual prediction of word-by-word translations, which obviously yields sub-standard results, the output space contains as many individual outcomes as there are words.

This vast output space rules out the application of machine learning algorithms such as support vector machines (Cortes and Vapnik, 1995) in their standard definition, in which they are restricted to binary classification problems. Solutions exist for more than two classes (e.g. Crammer and Singer (2002)), but these solutions tend to scale badly to very large amounts of classes. Rule learning algorithms such as Ripper (Cohen, 1995) and decision tree learners such as C4.5 (Quinlan, 1993) can deal with output spaces with many outcomes, but their scaling abilities with respect to numbers of classes are also limited. The one exception is the $k$-nearest neighbor classifier

| Dutch chunks | | | English chunks | | |
|---|---|---|---|---|---|
| left context | focus | right context | left context | focus | right context |
| | er is derhalve | in theorie genoeg tijd | there | has therefore been | enough time |
| ik nog even | op de zaak | zelf ingaan | to comment | on the issue itself | |
| dan logisch | en dat kan | ik dan ook | is perfectly logical | and I can | therefore |
| voor beperking | van staatssteun | waar dat nodig | restirctions | in state aid | where necessary |

Figure 1: Examples of aligned chunks in context.

(Cover and Hart, 1967; Dasarathy, 1991; Aha et al., 1991), which is insensitive to the number of classes in the output space. On the downside, the $k$-nearest neighbor classifier's scaling behavior with respect to the number of examples is notorious; compared to the aforementioned algorithms it is almost always slower in classification. Then again, fast approximations of the $k$-nearest neighbor classifier exist, such as IGTREE (Daelemans et al., 1997), a decision-tree approximation, which tends to trade some generalization performance against substantial speed gains in classification.

In this paper we explore the application of IGTREE to decoding in machine translation. Our explorations focus on Dutch-to-English translation, using EuroParl data (Koehn, 2005). We adopt a method which generates trigrams of output elements, and a post-processing method that searches the space of overlapping trigrams for the most likely translation. We train a word-based and a chunk-based system (the latter based on marker-based chunking), and compare these systems to a word-based and a phrase-based statistical machine translation system, respectively.

## 2 Example-based machine translation

Within the field of corpus-based machine translation, phrase-based statistical machine translation (SMT) is currently the dominant paradigm, yet much relevant research continues to be carried out in EBMT (Carl and Way, 2003). As with SMT, EBMT makes use of a corpus of source–target sententially-aligned examples to automatically extract translation resources. During translation, an EBMT system

1. searches the source side of the corpus for "close" matches and their equivalent target language translations;
2. identifies useful source–target fragments contained in those retrieved examples;
3. and recombines relevant target language fragments to derive the final translation of the input sentence.

The nature of the examples used in the first place may include using placeables as indicators of chunk boundaries (Brown, 1999), aligning phrase-structure (sub-)trees (Hearne and Way, 2006) or dependency trees (Watanabe et al., 2003).

In this work, we rely on a two-step approach. First, aligned sentences are segmented into chunks, which are then also aligned. The chunking and chunk alignment strategies we use are described in more detail in Section 3. This step provides us with a database of aligned chunks "in context". Some examples of such alignments are displayed in Figure 1.

Second, this database is given to a memory-based classifier, which stores the complete database in memory. When translating a new sentence, this sentence is itself segmented into chunks and matched against the database of aligned chunks. The matching and the recombination processes are described in Section 4.

## 3 Chunking and chunk alignment

In this Section, we describe the chunking and chunk alignments components of our system, which are used during the first step of our approach, i.e. when building the database of aligned chunks in context.

### 3.1 Marker-based chunking

One method for the extraction of chunks, used in the creation of the example database, is based on the Marker Hypothesis (Green, 1979), a psycholinguistic constraint which posits that all languages are marked for surface syntax by a specific closed set of lexemes or morphemes which signify context. Using a set of closed-class (or “marker”) words, such as determiners, conjunctions, prepositions, possessive and personal pronouns, aligned source-target sentences are segmented into chunks (Gough and Way, 2004) during a pre-processing step. A chunk is created at each new occurrence of a marker word, with the restriction that each chunk must contain at least one content (or non-marker) word. An example of such a chunking is given in Figure 2.

### 3.2 Chunk alignment

In order to align the chunks obtained by the chunking procedures described in Section 3.1, we used a simple greedy algorithm in which a source chunk is aligned to the “closest” target chunk.

In the following, $a$ denotes an alignment between a target sequence $e$ and a source sequence $f$, with $I = |e|$ and $J = |f|$. An alignment is a set of links between source positions and target positions:

$$a = \{(t_1, s_1), (t_2, s_2), \ldots, (t_n, s_n)\}, \qquad (1)$$

with $\forall k \in [1 \ldots n]$, $t_k \in [1 \ldots I]$ and $s_k \in [1 \ldots J$.

For each source chunk, we are looking for the “closest” target chunk, where the notion of proximity is based on conditional probabilities. In other terms, we have:

$$(t_k, s_k) \in a \Leftrightarrow \forall j \in J, P(e_{t_k}|f_{s_k}) \geq P(e_{t_k}|f_j). \qquad (2)$$

Since this model is asymmetric, due to the use of conditional probabilities, we perform the chunk alignment in both directions, and only keep the alignments that are found in both directions (i.e. the intersection), in order to obtain a set of high-quality chunk alignments. Note that it is also possible to use other alignment techniques, such as the “edit-distance style” dynamic programming alignment algorithm described in (Stroppa et al., 2006).

Instead of using an Expectation-Maximization algorithm to estimate the parameters, as commonly done when performing word alignment (Brown et al., 1993; Och, 2003), we directly compute these parameters by relying on the information contained within the chunks. The conditional probability $P(e_{t_k}|f_{s_k})$ can be computed in several ways. In our experiments, we have considered one main source of knowledge, based on word-to-word translation probabilities.

**Word-to-word probabilities** As a criterion to relate chunks, we use word-to-word probabilities, which are estimated using various statistical models and the EM algorithm (Och and Ney, 2003). Note that we are using the EM algorithm only to estimate word-to-word probabilities; the chunk-to-chunk probabilities described above do not require the use of this algorithm. Relationships between chunks are then computed through the following model, similar to IBM model 1 (Brown et al., 1993):

$$P(e_i, f_j) = \prod_k \sum_l P(e_{i_l}|f_{j_k}). \qquad (3)$$

This model is often used in SMT as a feature of a log-linear model; in this context, it is called a word-based lexicon model (Zens and Ney, 2004).

## 4 Memory-based decoding

The procedure of translating a new sentence is divided into a local phase in which memory-based classification takes place, and a global phase in which a translation of a sentence is assembled from the local predictions. The local phase operates at the chunk level, while the global phase operates at the sentence level. We describe both phases in more detail.

### 4.1 Local classification

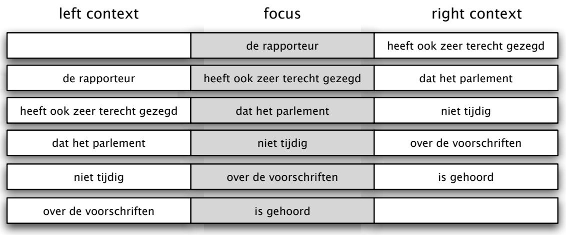
When a new sentence in the source language is presented as input, it is chunked, as described in Section 3. Subsequently, a sliding window is moved over the chunks to generate trigrams of chunks, where the first trigram of the sentence contains an empty left element, and the last trigram contains an empty right element. Figure 3 shows the trigrams generated for the chunked sentence *[de rapporteur] [heeft ook zeer terecht gezegd] [dat het parlement] [niet tijdig] [over de voorschriften] [is gehoord]*, to be translated as *the rapporteur has also quite rightly stated that parliament was not heard in time regarding the guidelines*.

To generate likely English translations, each Dutch chunk trigram is matched against the Dutch part of all aligned Dutch-English trigrams stored in the instance base of chunk trigrams derived

**Dutch:** *[de rapporteur] [heeft ook zeer terecht gezegd] [dat het parlement] [niet tijdig] [over de voorschriften] [is gehoord]*

**English:** *[the rapporteur] [has also quite rightly stated] [that parliament was not heard] [in time] [regarding the guidelines]*

Figure 2: A pair of Dutch and English sentences, with marker-based chunking. Marker words are underlined.



| left context | focus | right context |
|---|---|---|
| | de rapporteur | heeft ook zeer terecht gezegd |
| de rapporteur | heeft ook zeer terecht gezegd | dat het parlement |
| heeft ook zeer terecht gezegd | dat het parlement | niet tijdig |
| dat het parlement | niet tijdig | over de voorschriften |
| niet tijdig | over de voorschriften | is gehoord |
| over de voorschriften | is gehoord | |

Figure 3: An example sentence converted into six overlapping chunk trigrams.

from sentences in the training material (as displayed in Figure 1). Matching takes the form of a search in the decision-tree structure generated by IGTREE.

IGTREE[1] (Daelemans et al., 1997) is an algorithm for the top-down induction of decision trees. It compresses a database of labeled examples into a lossless-compression decision-tree structure that preserves the labeling information of all examples, and technically should be named a *trie* according to (Knuth, 1973). A labeled example is a feature-value vector, where features represent some input (here, a Dutch chunk trigram), associated with a symbolic class label representing some output (here, the aligned English chunk trigram). After the construction of the tree, it can be used to classify new examples not in the original database. A typical trie is composed of nodes that each represent a partition of the original example database, and the most frequent class of that partition. The root node of the trie thus represents the entire example database and carries the most frequent value as class label, while end nodes (leafs) represent a *homogeneous* partition of the database in which all examples have the same class label. A node is either a leaf, or is a non-ending node that branches out to nodes at a deeper level of the trie. Each branch represents a test on a feature value; branches fanning out of one node test on values of the same feature.

Classification in IGTREE is a straightforward traversal of the trie from the root node down, where a step is triggered by an exact match between a feature of the new example and an arc fanning out of the current node. When the next step ends in a leaf node, the homogeneous class at that node is returned; when no match is found with an arc fanning out of the current node, the most likely class stored at that node is returned.

To attain high compression levels, IGTREE adopts the same heuristic that most other decision-tree induction algorithms adopt, such as C4.5 (Quinlan, 1993), which is to create trees from a starting root node and branch out to test on the most informative, or most class-discriminative features first. Like C4.5, IGTREE uses information gain (IG) to estimate the discriminative power of features. The key difference between IGTREE and C4.5 is that IGTREE computes the IG of all features once on the full database of training examples, makes a feature ordering once on these computed IG values, and uses this ordering throughout the whole trie. Another difference with C4.5 is that IGTREE does not prune its produced trie, so that it performs a lossless compression of the labeling information of the original example database. As long as the database does not contain fully ambiguous examples (with the same features, but different class labels), the trie produced by IGTREE is able to reproduce the classifications of all examples in the original example database perfectly.

Due to the fact that IGTREE computes the IG of all features once, it is functionally equivalent to IB1-IG (Daelemans and Van den Bosch, 1992; Daelemans et al., 1999), a $k$-nearest neighbor classifier for symbolic features, with $k = 1$ and using a particular feature weighting in the similarity function in which the weight of each feature is larger than the sum of all weights of features with a lower weight (e.g. as in the exponential sequence $1, 2, 4, 8, \ldots$ where $2 > 1$, $4 > (1 + 2)$, $8 > (1 + 2 + 4), \ldots$). Both algorithms base their

[1] http://ilk.uvt.nl/timbl

| Rank | Chunk | Position | IG |
|---|---|---|---|
| 1 | middle | final | 11.44 |
| 2 | left | final | 10.00 |
| 3 | middle | penultimate | 8.69 |
| 4 | right | final | 7.70 |
| 5 | left | penultimate | 7.17 |
| 6 | middle | first | 5.90 |
| 7 | left | first | 5.83 |
| 8 | right | penultimate | 5.50 |
| 9 | right | first | 5.50 |

Table 1: The nine positional word features with the highest IG in the chunk trigram prediction task.

classification on the example that matches on most features, ordered by their IG, and guess a majority class of the set of examples represented at the level of mismatching. IGTree, therefore, can be seen as an approximation of IB1-IG with $k = 1$, having favorable asymptotic complexities as compared to IB1-IG. IGTree's computational bottleneck is the trie construction process, which has an asymptotic complexity of $O(n\ lg(v)\ f)$ of CPU, where $n$ is the number of training examples, $v$ is the average branching factor of IGTree (how many branches fan out of a node, on average), and $f$ is the number of features. Storing the trie, on the other hand, costs $O(n)$ in memory (for which we will provide empirical back-up evidence later), which is less than the $O(n\ f)$ of IB1-IG. Classification in IGTree takes an efficient $O(f\ lg(v))$ of CPU, versus the cumbersome $O(n\ f)$ of IB1-IG, given that in the typical case $n$ is much higher than $f$ or $v$ (Van den Bosch, 1997).

There is a considerable experimental freedom in choosing the actual feature encoding of the chunk trigrams. A natural first choice would be to have three features, namely the three chunks making up the trigram, but these features have many values that will occur only rarely, and thus will carry not too much discriminatory power. We therefore also considered individual positions as features, such as the final word of the middle chunk, the first word of the right chunk, the last word of the left chunk, etc.

After computing the IG of all possible positional features, we limited the number of features to the nine features with the highest IG values, listed in Table 1. The limit of nine was imposed on the basis of exploratory experiments on the development data, which showed no improvements beyond this selection of features. As the table shows, the most discriminative positional features are the final (rightmost) words of the middle chunk and the left chunk.

### 4.2 Global search

Translating one Dutch sentence involves the classification of all of its chunks in trigram context, to English chunk trigrams. To convert the set of overlapping trigrams into a single translation, the overlap between the predicted trigrams is used in such a way that the order of aligned chunks can in principle be altered in the English sentence. Figure 4 illustrates a perfect case of a resolution of the overlap. The first predicted chunk trigram signals by its empty left chunk that the middle chunk (labeled 'A') is indeed the first chunk of the English sentence. Also, the first chunk overlaps in two chunks with the fourth predicted chunk trigram. From this, the method concludes that the middle chunk of the fourth predicted (labeled 'B') is the second chunk to be concatenated; subsequently, the middle chunks of the third and the second prediction are concatenated.

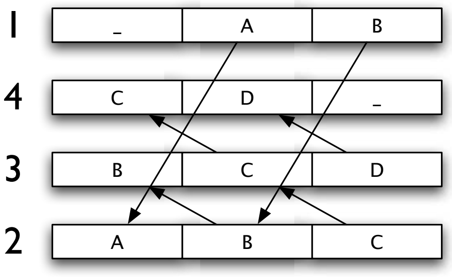


Figure 4: Producing a global solution by resolving the overlap between four trigrams. The left number indicates the resulting order in which the middle chunks are concatenated.

In the ideal case, the overlap between two chunk trigrams is a double exact match of the two overlapping chunks. In our current system, overlap is established if it involves an exact match of either of the two. If no exact match is found, the system backs off to simply retaining the order of the predicted chunks, and taking the middle chunk of each predicted chunk trigram.

### 4.3 Word-level memory-based decoding

As a natural comparison to the marker-based chunking approach, we also implemented a word-level memory-based decoder which operates on words rather than chunks (the word alignments can be obtained using the GIZA++ software[2], Och and Ney (2003).) Word trigrams are predicted by IGTREE, and overlap among predicted word trigrams determines the final ordering of the words in the generated English translation. We adopted a sliding window of three left neighboring words and three right neighboring words, on the basis of exploratory experiments on the development data, which showed no improvements beyond this context width.

## 5 Data preparation

The experiments were carried out using the English and Dutch sections of the Europarl datasets (Koehn, 2005). This corpus is extracted from the Proceedings of the European Parliament, from 1996 to 2003. Europarl contains approximately 28 million words for each language, and is aligned at the sentence level. In these experiments, we focused on the Dutch to English translation task.

From this corpus of 1,012,671 aligned sentences, we randomly extracted 1,500 aligned sentences for development and 1,500 for testing, the rest (1,009,671 aligned sentences) being used for training. In order to get a more confident set of aligned sentences for training, we performed some filtering based on the lengths and the relative lengths of the sentences, ending up with a training set of 728,339 aligned sentences. Note that the system was trained using only these datasets; we did not include any kind of external data.

As a pre-processing step, the English sentences were tokenized using the Maximum-Entropy based tokenizer of the OpenNLP toolkit[3]. This tokenizer was also used for Dutch, as we found that it was properly dealing with almost all punctuation marks. We additionally used a set of regular expressions specific to Dutch, for example to deal with the construction of plural forms of nouns ending with vowels (*'s*), or compound expressions with the infix *-en-*. Finally, case information was removed.

As there are about four chunks in the average Dutch sentence, the memory-based classifier is confronted with a training set of 2,894,153 examples. In this training set 2,768,551 different English chunk trigrams occur as classes, indicating, not surprisingly, that there are only very few chunk trigrams occurring more than once.

## 6 Results

The system output is evaluated with respect to the BLEU (Papineni et al., 2002), Word-Error-Rate (WER), and Precision-independent-Error-Rate (PER) evaluation metrics. Since we extracted the testing data from the aligned corpus, these metrics are computed using only one reference. We did not take case information into account during the evaluation process.

### 6.1 Baselines

We compared our approach to two other systems: a word-based and a phrase-based SMT system. Both systems make use of word alignments extracted from the training data, using the GIZA++ software (Och and Ney, 2003). This software implements the IBM alignment models (Brown et al., 1993) as well as the HMM alignment model (Vogel et al., 1996); the parameters of the models are estimated using the EM algorithm. Word alignment is performed in both directions and we followed the "refined" method of (Koehn et al., 2003) to extract a set of high-quality word alignments from the original uni-directional alignment sets. For the phrase-based system, we apply the technique described in (Koehn et al., 2003) to build a database of aligned phrases consistent with the word alignments. In both cases, decoding is performed using the phrase-based decoder PHARAOH (Koehn, 2004), with standard parameters. During decoding, a (target) language model is also needed: we use a simple 3-gram language model trained on the English portion of the training data, using the SRI Language Modeling Toolkit[4] (Stolcke, 2002), with modified Kneser-Ney smoothing (Chen and Goodman, 1996). We developed a phrase-based SMT system that can back-off to the word unigram level (as is the standard), but for comparison purposes with our chunk-level decoder which does not back off to the unigram level, we have also developed a phrase-based SMT system that only makes use of phrases. The results obtained on the Dutch to English translation task using these systems are reported in Table 2.

As expected, the phrase-based SMT system that can back off to the word level outperforms the word-based system, according to all of the metrics.

[2] http://www.fjoch.com/GIZA++.html

[3] http://opennlp.sourceforge.net/

[4] http://www.speech.sri.com/projects/srilm/

| | BLEU | WER | PER |
|---|---|---|---|
| Word-based SMT | 0.2041 | 75.38 | 52.07 |
| PB SMT with words | 0.2572 | 68.48 | 48.08 |
| PB SMT without words | 0.2171 | 71.60 | 52.81 |

Table 2: Baseline scores: BLEU, WER and PER for the word-level SMT and two variants of phrase-based SMT, with and without words.

However, when the phrase-based system is limited to using phrases only, its performance is roughly comparable to the word-based SMT system, illustrating that it is the combination that yields the boosted performance. In general, these results show the non-triviality of the task, since even the better of the two phrase-based systems has a WER of $68.48$.

**Chunk-level decoder** The performance of the memory-based trigram-chunk decoder, listed under “Trigram chunk MB” in Table 3, is considerably lower than the phrase-based SMT baseline system (without the back-off option to the word level). We performed two additional experiments in which we simplified the memory-based trigram-chunk decoder. First, we trained a memory-based classifier to generate bigrams of chunks, consisting of the focus chunk along with its right-hand neighboring chunk. Second, we trained a memory-based classifier to map a trigram of Dutch chunks to only the middle English chunks. In the latter case chunks cannot be reordered anymore, but the decoder could still produce valid chunk translations.

| | BLEU | WER | PER |
|---|---|---|---|
| Phrase-based SMT | 0.2171 | 71.60 | 52.81 |
| Trigram chunk MB | 0.0729 | 85.96 | 73.02 |
| Bigram chunk MB | 0.0921 | 81.40 | 67.45 |
| Unigram chunk MB | 0.1103 | 77.15 | 64.98 |

Table 3: Performance of variants of the chunk-level memory-based decoder, compared against the phrase-based SMT baseline (without words).

As Table 3 shows, the trigram and bigram chunk decoders are in fact outperformed by the unigram decoder, which in turn is still outperformed by the phrase-based SMT system. It appears that the massive amount of classes (about 2.7 million classes in the trigram case) is making the class space so sparse that the memory-based decoders are severely affected by it.

**Word-level decoder** The word-level memory-based decoders were trained and tested on the same material, yielding the scores listed in Table 4. As the table shows, the trigram word decoder lags behind in BLEU score and PER, but does have a lower WER as compared to the word-based SMT baseline system, interestingly. The trigram word decoder performs better than the unigram word-level memory-based decoder in terms of BLEU score and WER, but not in PER.

| | BLEU | WER | PER |
|---|---|---|---|
| Word-based SMT | 0.2041 | 75.38 | 52.07 |
| Trigram word MB | 0.1785 | 71.29 | 56.08 |
| Unigram word MB | 0.1351 | 72.81 | 54.78 |

Table 4: Performance of variants of the word-level memory-based decoder, compared against the word-based SMT baseline.

**Overall comparison** It would seem logical to explain the relatively lower performance of the chunk-level decoders on the basis of their very sparse class space; in 2.8 million instances, about 2.7 million chunk trigrams occur, and 1.3 million single chunks. The majority of these classes occur only once, and it is obvious that in new material many of the chunks that should be predicted do not occur in the training data at all – hence, the memory-based decoder can never predict them. This may partly explain their low performance, but in the word-level trigram data, no less than 3.6 unique word trigrams occur. Still, these are 3.6 million classes in 12 million instances, the actual number of words in our training corpus. Thus, the choice to encode examples chunk by chunk (yielding 2.8 instances) rather than word by word (yielding 12 million instances) appears to be an important reason for the lower performance of the chunk-level decoders.

Sparseness is always relative to the amount of training material, and this amount is essentially an experimental variable. We explored the scaling properties of our four memory-based decoders systems by training them on increasing amounts of learning data up to the maximal available amount. Figure 5 displays the four learning curves against a logarithmic x axis. All four curves display an upward trend; with more data, all four systems would perform better still. Yet, the word unigram curve is the one that flattens most; this simple decoder offers the best score until about 500,000

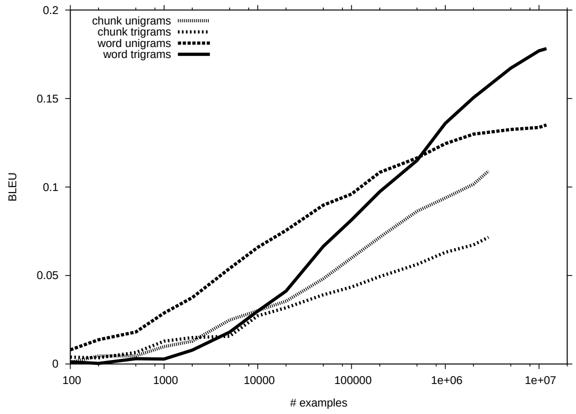


Figure 5: Increasing BLEU scores by the memory-based chunk-level and word-level decoders, with increasing amounts of learning examples. The x-axis has a logarithmic scale.

training instances, but after this point it is overtaken by the word trigram system that continues with a steeper curve. With its average log-linear increase of about 2.5% per ten-fold increase of the amount of learning material, about ten times more data would be needed to let the trigram word-level decoder equal the current score of the word-based SMT system. While the chunk-based systems show rather convincing log-linear growth in their learning curves, they would need another one or two orders of magnitude more training data to reach the same level of performance.

### 6.2 Training and decoding speed

Due to its favorable scaling abilities, IGTREE only tends to grow linearly in memory usage with the amount of training examples. In our experiments[5] we observed the following:

- The amount of nodes in the trie is almost the same as the amount of training examples; with the 2.8 million examples in the trigram chunk-level system, the trie contains 2.9 million nodes;
- The time it takes to build the trie is roughly but consistantly linear in the amount of training examples – it takes 311 seconds to build the above-mentioned 2.9 million-node trie, and it takes about 1 second per 11 thousand training examples;
- With the current amounts of training material, classification time is roughly linear in the amount of training examples – the maximally-sized trie classifies all 8,948 test examples within 3 seconds, and for every million of training examples added, testing becomes about 1 second slower.

Since SMT systems rely on complex parameter estimations, expensive computations are required. We measured the processing times needed by the word-based and phrase-based SMT systems, for both training and decoding. They are reported on Table 5. The results in the table indicate that memory-based decoding with IGTREE is two orders of magnitude faster than decoding in phrase-based SMT. Clearly, the memory-based classifiers represent a different trade-off in the quality–speed continuum.

| | Training | Decoding |
|---|---|---|
| Phrase-based SMT | ± 23 h | ± 30 m |
| Unigram chunk MB | align + ± 4 m | ± 1 s |
| Trigram chunk MB | align + ± 4 m | ± 3 s |
| Unigram word MB | align + ± 21 m | ± 2 s |
| Trigram word MB | align + ± 29 m | ± 7 s |

Table 5: Training and decoding times for three systems. The memory-based systems make use of alignments (“align”) generated separately.

## 7 Discussion

We developed a straightforward approach to decoding in machine translation, based on memory-based classification. The approach fits into the EBMT framework; it allows the mapping of sequences of chunks (here, marker-based chunks) in the source language to sequences of translated chunks in the output language. Also, the method allows the development of straightforward word-level translation.

We found that at present the method is not competitive when compared to default word-based and phrase-based SMT systems in terms of BLEU scores. Yet, the best memory-based decoder, the trigram word-level system, attains a performance that approaches the performance of the word-based SMT system (it yields worse BLEU and PER scores, but performs better in terms of WER). The assumed positive effect of predicting overlapping trigrams of chunks was not observed. We

[5] Experiments with the memory-based classifier were run on AMD Opteron 3.4 Ghz CPUs; experiments with SMT models were run on an Intel Xeon 3.2 Ghz CPU.

reasoned that the task of predicting these trigrams is harmed by the extreme sparseness of the class space; only two or more orders of magnitude of additional learning material could get these methods on a par with the SMT baseline.

On the other hand, we showed that the memory-based classifier, IGTREE, a $k$-nearest neighbor approximation, scales well to increased amounts of learning material. Even better than theoretically predicted, within the current experiment we observed that training and classification time and also memory storage costs were linear in the number of training examples. Extrapolating these results (with the expectation that training times and classification speeds will in fact grow slightly superlinearly), we conclude for now that memory-based decoders are interesting in cases where there is a need for speedy training and/or classification, e.g. in on-line translation engines.

In future work we plan to combine the word-level and chunk-level memory-based decoders, e.g. to back off to word-level predictions when chunk-level predictions are made below a certain confidence level, to offer a better comparison with standard phrase-based SMT systems. More generally, we have not explored at all the use of confidence in the predictions of IGTREE. Despite the fact that IGTREE does not produce probability distributions, it would be relevant to use the information as to whether a prediction was made at a leaf or a non-ending node, and at what level of the tree. We also plan to test various other chunking and chunk alignment method, to investigate the influence of this important preprocessing step. We aim to look at English-to-Dutch translation, and other language pairs. Finally, we aim to generalize the trigram method towards the constraint satisfaction inference framework introduced in (Canisius et al., 2006), which offers a generic and powerful search algorithm for exploiting the overlap in predicted $n$-grams.

## Acknowledgements

Thanks to Science Foundation Ireland (http://www.sfi.ie) Principal Investigator Award 05/IN/1732 for part-funding this research. Work by the first author was funded by the Netherlands Foundation for Scientific Research.

## References

D. W. Aha, D. Kibler, and M. Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 6:37–66.

P. Brown, S. Della Pietra, V. Della Pietra, and R.L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

R. Brown. 1999. Adding linguistic knowledge to a lexical example-based translation system. In *Proceedings of TMI 1999*, pages 22–32, Chester, England.

S. Canisius, A. Van den Bosch, and W. Daelemans. 2006. Constraint satisfaction inference: Non-probabilistic global inference for sequence labelling. In *Proceedings of the EACL 2006 Workshop on Learning Structured Information in Natural Language Applications*, pages 9–16, Trento, Italy.

M. Carl and A. Way. 2003. *Recent Advances in Example-Based Machine Translation*, volume 21 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, Dordrecht, The Netherlands.

S.F. Chen and J. Goodman. 1996. An empirical study of smoothing techniques for language modelling. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 310–318. ACL.

W. Cohen. 1995. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann.

C. Cortes and V. Vapnik. 1995. Support vector networks. *Machine Learning*, 20:273–297.

T. M. Cover and P. E. Hart. 1967. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21–27.

K. Crammer and Y. Singer. 2002. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2):201–233.

W. Daelemans and A. Van den Bosch. 1992. Generalisation performance of backpropagation learning on a syllabification task. In M. F. J. Drossaers and A. Nijholt, editors, *Proceedings of TWLT3: Connectionism and Natural Language Processing*, pages 27–37, Enschede. Twente University.

W. Daelemans, A. Van den Bosch, and A. Weijters. 1997. IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423.

W. Daelemans, A. Van den Bosch, and J. Zavrel. 1999. Forgetting exceptions is harmful in language learning. *Machine Learning, Special issue on Natural Language Learning*, 34:11–41.

B. V. Dasarathy. 1991. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA.

N. Gough and A. Way. 2004. Robust large-scale EBMT with marker-based segmentation. In *Proceedings of TMI 2004*, pages 95–104, Baltimore, Maryland.

T. Green. 1979. The necessity of syntax markers. two experiments with artificial languages. *Journal of Verbal Learning and Behavior*, 18:481–496.

M. Hearne and A. Way. 2006. Disambiguation strategies for data-oriented translation. In *Proceedings of the 11th Conference of the European Association for Machine Translation*, Oslo, Norway.

D. E. Knuth. 1973. *The art of computer programming*, volume 3: Sorting and searching. Addison-Wesley, Reading, MA.

P. Koehn, F.-J. Och, and D. Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT-NAACL 2003*, pages 48–54, Edmonton, Canada.

P. Koehn. 2004. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Proceedings of AMTA 2004*, pages 115–124, Washington, District of Columbia.

P. Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Machine Translation Summit X*, pages 79–86, Phuket, Thailand.

F.-J. Och and H. Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

F.-J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL 2003*, pages 160–167, Sapporo, Japan.

K. Papineni, S. Roukos, T. Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of ACL 2002*, pages 311–318, Philadelphia, PA.

J.R. Quinlan. 1993. C4.5*: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

A. Stolcke. 2002. SRILM – An extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904, Denver, Colorado.

N. Stroppa, D. Groves, A. Way, and K. Sarasola. 2006. Example-based machine translation of the Basque language. In *Proceedings of AMTA 2006*, pages 232–241, Cambridge, MA.

A. Van den Bosch. 1997. *Learning to pronounce written words: A study in inductive language learning*. Ph.D. thesis, Universiteit Maastricht.

S. Vogel, H. Ney, and C. Tillmann. 1996. HMM-based word alignment in statistical translation. In *Proceedings of COLING 1996*, pages 836–841, Copenhagen, Denmark.

H. Watanabe, S. Kurohashi, and E. Aramaki. 2003. Finding translation patterns from paired source and target dependency structures. In M. Carl and A. Way, editors, *Recent Advances in Example-Based Machine Translation*, pages 397–420. Kluwer Academic Publishers, Dordrecht, The Netherlands.

R. Zens and H. Ney. 2004. Improvements in phrase-based statistical machine translation. In *Proceedings of HLT-NAACL 2004*, pages 257–264, Boston, MA.