Diplomarbeit

Comparison and Implementation of MT Evaluation Methods

Lena Morgenroth

2011-07-11

Betreuender Hochschullehrer:
Prof. Dr.-Ing. habil. Heiko Vogler

Institut für Theoretische Informatik
Fakultät Informatik
Technische Universität Dresden

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst habe.

Dresden, 11. Juli 2011 ______________________________

Contents

1. Introduction

Machine translation (MT) has been an area of research within the field of natural language processing since the 1950s. After some initial euphoria followed the realization that the goal of the fully automatic high quality translation (FAHQT) was not going to be reached so easily. Research continued with more realistic goals, such as developing MT systems that would aid humans in translation by providing at least informative, intelligible and correct output.

However, the FAHQT still remains a goal, and the process towards that goals wants to be measured. While any human being can easily and intuitively discern a very high quality translation from bad ones, the task of actually quantifying the goodness, measuring progress and comparing systems is anything but trivial.

Intuitively, a good translation is one that is understandable, has the same meaning as the original text, is grammatically fluent and stylistically adequate. However, such intuitive notions of translation quality can not systematically be used in the development and comparison of concrete nowadays systems.

Quantitative metrics have long been used to evaluate machine translation systems against each other, but involved purely human judgement in the beginning. A variety of scoring systems has been tried out that allowed quantitative evaluations to be performed by trained human evaluators. These measures allow for a somewhat fair comparison, but cannot be used in the development cycle, as they are too slow and costly.

Automatic metrics have been investigated at least as early as 1991, when Thompson [11] pushed for the exploration of automatic metrics for NLP tasks, especially for machine translation stating that

> ...fast, accurate, automatic evaluation methods are of vital importance in the development process for any large scale natural language processing application. Historically there has been little emphasis on evaluation in the Machine Translation community [...], [and] the methods proposed are not automatic, thus not fast, nor in most cases is there any obvious way to test their accuracy, that is to say the statistical significance of their results.

In 2005 Banerjee et.al. describe minimal requirements an automatic MT evaluation method has to fulfill [5]:

> In order to be both effective and useful, an automatic metric for MT evaluation has to satisfy several basic criteria. The primary and most

intuitive requirement is that the metric have very high correlation with quantified human notions of MT quality. Furthermore, a good metric should be as sensitive as possible to differences in MT quality between different systems, and between different versions of the same system. The metric should be consistent (same MT system on similar texts should produce similar scores), reliable (MT systems that score similarly can be trusted to perform similarly) and general (applicable to different MT tasks in a wide range of domains and scenarios).

The first fully automated metric for MT systems was BLEU [8], proposed by IBM in 2002, which we will describe in detail in section 2.2. It was soon widely adopted, leading to rapid improvements in machine translation quality due to a massive speed-up of the development cycle. However, as it also carries a number of drawbacks, alternative metrics have been proposed since, one of them TER [10] which we describe in detail in section 2.3. We also provide an implementation for BLEU.

2. Automatic Metrics for Machine Translation Evaluation

2.1. General Terms and Concepts

2.1.1. Segment-, Document-, and Corpus-Level Evaluation

The general idea behind most MT evaluation methods is comparing the likeness of the machine translation output and, ideally, several high quality reference translations (RT). TER and BLEU, which we will describe in more detail in this work, are examples of such similarity measures.

Independent of the actual method employed, a decision must be made about the granularity of the measurement. The lowest level commonly evaluated is the *segment level.* A segment usually corresponds to a sentence, but might also be a phrase that does not form a sentence (e.g. a news headline). A segment might also contain more than one sentence, e.g. if a single sentence in language A is translated into two sentences in language B. The segment in A would consist of one sentence, while the corresponding segment in B would consist of two. For MT evaluation, a text and its translation need to be divided up into corresponding segments.

Evaluation at the segment level is the most useful within the development cycle. Turian et.al. elaborate on this in section 3.2 of [12]:

> Consider how MT system developers would measure the effect of a system modification on a large development bitext. Typically, they would like to know not only whether the modification improved performance on some objective measure, but also why or why not. The fastest way to gain such insight is to compare the systems before and after output on some specific text sentences. The sentences that are most likely to highlight the qualitative effects of the modification to the MT system are those for which the objective evaluation measure changes the most.

However, segment level evaluation does not necessarily provide a very good insight into overall translation quality. Indeed, even human judgements on the segment level do not correlate very well with each other [12] (section 4), calling into question segment the value of segment level scores when it comes to reliably judging overall system quality.

Furthermore, the MT system might not work equally well on all kinds of segments, but depend on factors like the specific vocabulary used in that segment or its grammatical structure. Additionally, the MT evaluation method itself might not be equally accurate on all kinds of MT - RT segment pairings (see section 2.5.1 for some examples specific to the BLEU metric).

Evaluating MT quality on a large and diverse test corpus instead gives less weight to statistical outliers and allows for a more balanced judgement of overall translation quality of an MT system.

In evaluations such as NIST's OpenMT Evaluation, this level of granularity is usually referred to as the system level, implying an evaluation of the MT system itself, not the system's translation of a specific segment or a specific document. However, as has been argued e.g. by Gimenez in [3](p.20), "the behavior of automatic metrics depends on a number of variables such as the language pair, the specific domain of the translation task, and the typology of systems under evaluation." An MT system might, e.g. perform much better at translating instruction manuals than at translating newswire text, due to the different language phenomena that are encountered. We therefore refer to this level of evaluation as the *corpus level* instead, underlining the fact that the resulting score depends not only on the evaluated system and the MT evaluation measure, but also on the test corpus used for evaluation.

MT evaluation methods additionally often support evaluation at a medium granularity, the *document level*. This allows e.g. to identify genres or specific documents that an MT system has most trouble dealing with or performs best at, without having to deal with a variance as large as for segment-level scores.

2.1.2. Tokens, Tokenization, Phrases and N-Grams

Most of today's machine translation evaluation methods compare machine translated segments to reference translation segments by matching their respective words or phrases (i.e. sequences of words) with each other. However, the term "word" is somewhat inaccurate, as e.g. punctuation is often included in the regular matching process. For the purpose of this work, we refer to the smallest units for matching (i.e. words, punctuation, numbers, clitics, ...) as *tokens*.

Tokenization, i.e. the breaking up of a string of, say, unicode characters, representing a segment into a sequence of tokens (i.e. substrings, most possibly dropping white spaces in the process) itself brings up a number of questions. While separation at word boundaries marked by spaces is straightforward, real world data offers a number of (sometimes language dependend) cases to be considered.[1] We will not go into details of tokenization, but simply state that allowances should be made for language dependent adjustments, at least if a special case is frequent or a good correlation at the segment level is aimed for.

Most current methods for MT evaluation also include matching on longer sequences of tokens, i.e. phrases or n-grams. Phrase is a lingistic concept coming from the structuralist view of grammar. It refers to the constituents of a sentence

[1]e.g is repeated punctuation "?!" or "..." treated as one token or a sequence of tokens? Should all punctuation be treated as a separate token? What about in "3.14" or "-10" or "A. Einstein"?

that consist of more than one word belonging together grammatically , e.g. noun phrases like "a nice car" or verb phrases like "would have said". As the described MT evaluation methods do not have any built-in notion of grammar, we use the term a little more losely. For the purpose of this work, a *phrase* is a contiguous subsequence of one or more tokens in some segment. An *n-gram* is a contiguous subsequence of exactly n tokens in some segment. In the following, we use the terms 1-gram, unigram and token interchangeably.

We denote segments, phrases and n-grams as sequences of tokens and we use subscript to indicate a token's position within a sequence. E.g. take the segment $S = s_1s_2s_3s_4$, then $s_1, \ldots, s_4$ are all tokens or unigrams occurring in the segment. s_2s_3 is a bigram in S, $s_1s_2s_3$ is a trigram in S, and both are phrases in S. When writing about real natural language tokens rather than working on the symbolic level, we separate tokens within n-grams or phrases using a whitespace character as a delimiter. The bigrams my selfless and myself less are not identical, even though a direct concatenation without delimiter would yield myselfless in both cases.

2.1.3. Text Normalization

Additional *text normalization* steps after tokenization can be applied to machine and/or reference translation(s) before employing an MT evaluation method. Such measures range from the very simple (like e.g. lower casing), to elaborate methods that involve a lot of language specific ressources.

An example of a more far-reaching method is the use of a stemmer. Automatic stemming reduces different word forms to a canonical form, thus identifying e.g. "house" with "houses" and thereby generating additional matches. The use of stemmers has been shown to greatly increase correlation of automatic MT evaluation scores with human judgements for different automatic metrics as early as 2004 in [6]. On the downside, stemmers, like other language specific ressources, are usually only available for ressource-rich languages that have been the subject of much NLP research.

Going even further down the text normalization path, it is possible to include the semantic level as well, computing matches not on words but on concepts, e.g. by allowing the matching of synonyms. The further we go into text normalization, the more language-dependent we become. A useful approach is to develop robust general methods for evaluation that are a reasonable baseline for many or most languages and that can be extended to make use of more linguistic knowledge as it becomes available.

2.2. BLEU

In the following sections, we describe and define the BLEU[2] method as originally published by Papineni et.al. [8]. We will discuss the use of BLEU in practice in

[2]**Bi**Lingual **E**valuation **U**nderstudy

section 2.2.2. We document our implementation of the method in section 2.2.3.

2.2.1. The Original BLEU Metric

BLEU is a similarity measure comparing a machine translation against one or more reference translations. The score can be computed on the segment, document or corpus level. Comparison takes place segment-wise; matching is done on the word- and phrase-level.

In the following we will explain and define the computation of the precision score that underlies BLEU and its modification to allow for the use of multiple reference translations. As modified n-gram precision does not penalize incomplete translations, Papineni et.al. introduce a brevity penalty [8], which we describe, before explaining how BLEU is computed for the document and corpus level.

Unigram Precision

BLEU is based on the precision score, a well known measure when it comes to evaluating pattern recognition algorithms. *Precision* is the number of correctly recognized items in relation to the total number of recognized items that might include false positives.

In application to MT, precision refers to the ratio of correct n-gram occurrences in the translation to the total number of n-gram occurrences in the translation. An n-gram occurrence in the machine translation is correct in the sense of BLEU if an exactly matching occurrence can be found in any reference translation. An n-gram occurrence in a reference translation can only be matched against once. As we are only interested in the match count, it does not matter which of several possible matches is chosen.

Let us consider the following machine translated segment M and a corresponding reference translation segment R. Punctuation is treated as a separate token, and we assume that all text has been transformed to lower case. Correct unigrams in the machine translation and their matches in the reference are underlined. Either the first or the second occurrence of more in M can be matched with the single occurrence of more in R.

$\boldsymbol{M}$:
$\underline{\text{the}}$ $\underline{\text{situation}}$ even $\underline{\text{more}}$ complex , more $\underline{\text{dangerous}}$ $\underline{\text{than}}$ $\underline{\text{it}}$ $\underline{\text{was}}$ $\underline{\text{in}}$ past $\underline{\text{decades}}$

$\boldsymbol{R}$:
a $\underline{\text{situation}}$ $\underline{\text{more}}$ complicated and $\underline{\text{dangerous}}$ $\underline{\text{than}}$ $\underline{\text{it}}$ $\underline{\text{was}}$ $\underline{\text{in}}$ $\underline{\text{the}}$ previous $\underline{\text{decades}}$

A pure precision score on unigrams does not take word order into account at all. For the purpose of computing the unigram precision score, a segment can be regarded as a multiset of unigrams.[3]

[3] We cannot use sets, as we need to model the number of occurrences of each token in the segment. We could model unigrams as pairs of a token and its position within the segment instead, in

We use calligraphic letters for multisets and indicate the multiplicity of their elements as superscript, e.g. $\mathcal{M} = \{a^1, b^4\}$ is a multiset containing a once and b four times.

Definition 1 (N-Gram Multiset Representation) *Let $S = s_1 \ldots s_k$ be a segment consisting of k tokens. Let $n \geq 1$. Then $\mathcal{S}_n = \{s_1 \ldots s_n \ , \ \ldots \ , \ s_{k-n+1} \ldots s_k\}$ denotes its n-gram multiset representation.*

The 1-gram multiset representations of M and R are:

$$\begin{aligned}\mathcal{M}_1 = \{&\mathsf{complex}^1, \mathsf{dangerous}^1, \mathsf{decades}^1, \mathsf{even}^1, \mathsf{in}^1, \mathsf{it}^1, \mathsf{more}^2, \mathsf{past}^1, \mathsf{situation}^1, \mathsf{than}^1,\\ &\mathsf{the}^1, \mathsf{was}^1, ,^1\}\\ \mathcal{R}_1 = \{&\mathsf{a}^1, \mathsf{and}^1, \mathsf{complicated}^1, \mathsf{dangerous}^1, \mathsf{decades}^1, \mathsf{in}^1, \mathsf{it}^1, \mathsf{more}^1, \mathsf{previous}^1, \mathsf{situation}^1,\\ &\mathsf{than}^1, \mathsf{the}^1, \mathsf{was}^1\}\end{aligned}$$

Finding the multiset $\mathcal{C}_1$ of correctly translated 1-grams in a machine translated segment M with reference to some R is then equivalent to the intersection of their 1-gram multiset representations.

$$\mathcal{C}_1 = \mathcal{M}_1 \cap \mathcal{R}_1 = \{\mathsf{dangerous}^1, \mathsf{decades}^1, \mathsf{in}^1, \mathsf{it}^1, \mathsf{more}^1, \mathsf{situation}^1, \mathsf{than}^1, \mathsf{the}^1, \mathsf{was}^1\}.$$

Note that the unigrams are matched on string identity alone. M's the is the article preceding situation and most likely a false translation, as the reference translation has the indefinite article a instead. Nevertheless the is an element of $\mathcal{C}_1$ and thus counted as a correct unigram, as it is matched against the the preceding previous decades in R[4]. Calculation of the unigram precision score is then straightforward:

Definition 2 (Unigram Precision Score) *Let $M = m_1 \ldots m_k$ be a machine translation segment, $R = r_1 \ldots r_l$ a corresponding reference translation segment and $\mathcal{M}_1$ and $\mathcal{R}_1$ their respective 1-gram multiset representations. The 1-gram precision score of M in reference to R is*

$$p_1(M, R) = \frac{|\mathcal{C}_1|}{|\mathcal{M}_1|}$$

where $\mathcal{C}_1 = \mathcal{M}_1 \cap \mathcal{R}_1$ is the multiset of correct 1-grams in M.

order to use sets for representation. But position information is not explicitly needed for computation, hence multisets seem a better fit.

[4]This shortcoming of BLEU also allows for the incorrect matching of homographs, i.e. semantically different word forms with identical spelling, e.g. "can" in "he can dance" and "a can of worms".

In our example, the resulting unigram match count is $|\mathcal{C}_1| = 9$, the unigram precision score is $p_1 = \frac{|\mathcal{C}_1|}{|\mathcal{M}_1|} = \frac{9}{14} = 0,6428$.

Multiple Reference Translations and Modified Unigram Precision

Using n-gram precision with a single reference translation, BLEU is too strict on machine translations that are correct and fluent but happen to use a different vocabulary than the reference. In our example in past decades might be an adequate alternative translation to in the previous decades, but past is not recognized as a match for previous. Therefore, in order to better cover the space of possible good translations, which might differ in phrasing and word-choice, BLEU is designed to make use of multiple reference translations.

Let us consider an alternative reference translation S and its unigram multiset representation:

***M*:**
the situation even more complex , more dangerous than it was in past decades

***S*:**
a situation more complex and dangerous than in past decades

$$\begin{aligned}\mathcal{M}_1 &= \{\text{complex}^1, \text{dangerous}^1, \text{decades}^1, \text{even}^1, \text{in}^1, \text{it}^1, \text{more}^2, \text{past}^1, \text{situation}^1, \text{than}^1, \\ &\quad \text{the}^1, \text{was}^1, \text{,}^1\} \\ \mathcal{S}_1 &= \{\text{a}^1, \text{and}^1, \text{complex}^1, \text{dangerous}^1, \text{decades}^1, \text{in}^1, \text{more}^1, \text{past}^1, \text{situation}^1, \text{than}^1\} \\ \mathcal{C}_1 &= \mathcal{M}_1 \cap \mathcal{S}_1 = \{\text{complex}^1, \text{dangerous}^1, \text{decades}^1, \text{in}^1, \text{more}^1, \text{past}^1, \text{situation}^1, \text{than}^1\}\end{aligned}$$

Matching against S individually, we obtain a unigram precision score of $p_1 = \frac{|\mathcal{C}_1|}{|\mathcal{M}_1|} = \frac{8}{14}$, which is similar to the score of matching against R, but obtained through different matches.

For a more realistic measure of correctly translated unigrams in M, we compute a *modified unigram precision* score by matching simultaneously against all available reference translations. A unigram is counted as correct if it has a matching occurrence in any of the references. In our example, this allows to recognize both the phrasing it was from R as well as complex and past from S as correctly translated unigrams in M.

***M*:**
the situation even more complex , more dangerous than it was in past decades

***R*:**
a situation more complicated and dangerous than it was in the previous decades

***S*:**
a situation more complex and dangerous than in past decades

Note that the second occurrence of more in M is not matched against more in S. BLEU imposes an upper limit on matches for the same n-gram. An n-gram in M

can only be matched as many times as the maximum number of its occurrences in any single reference translation. If this were not the case, BLEU would not penalize the overgeneration of words that are very common or only have one translation and thus occur in many reference translations.

Formally, we model matching with an upper limit as the intersection of M's multiset representation with the union of the multiset representations of all reference translations.

Definition 3 (Modified Unigram Precision Score) *Let M be a machine translated segment, $R^1, \ldots, R^m$ corresponding reference translation segments and $\mathcal{M}_1, \mathcal{R}_1^1, \ldots, \mathcal{R}_1^m$ their respective unigram multiset representations. Then M's modified unigram precision score with respect to $R^1, \ldots, R^m$ is*

$$p'_1(M, R^1, \ldots, R^m) = \frac{|\mathcal{C}'_1|}{|\mathcal{M}_1|} \text{ , } \textit{where } \mathcal{C}'_1 = \mathcal{M}_1 \cap \overline{\mathcal{R}}_1 \textit{ and } \overline{\mathcal{R}}_1 = \bigcap_{i=1}^{m} \mathcal{R}_1^i.$$

For our example, this yields:

$$\mathcal{M}_1 = \{\text{complex}^1, \text{dangerous}^1, \text{decades}^1, \text{even}^1, \text{in}^1, \text{it}^1, \text{more}^2, \text{past}^1, \text{situation}^1, \text{than}^1, \text{the}^1, \text{was}^1, ,^1\}$$

$$\mathcal{R}_1 = \{\text{a}^1, \text{and}^1, \text{complicated}^1, \text{dangerous}^1, \text{decades}^1, \text{in}^1, \text{it}^1, \text{more}^1, \text{previous}^1, \text{situation}^1, \text{than}^1, \text{the}^1, \text{was}^1\}$$

$$\mathcal{S}_1 = \{\text{a}^1, \text{and}^1, \text{complex}^1, \text{dangerous}^1, \text{decades}^1, \text{in}^1, \text{more}^1, \text{past}^1, \text{situation}^1, \text{than}^1\}$$

$$\mathcal{R}_1 \cup \mathcal{S}_1 = \{\text{a}^1, \text{and}^1, \text{complex}^1, \text{complicated}^1, \text{dangerous}^1, \text{decades}^1, \text{in}^1, \text{it}^1, \text{more}^1, \text{past}^1, \text{previous}^1, \text{situation}^1, \text{than}^1, \text{the}^1, \text{was}^1\}$$

$$\mathcal{C}'_1 = \mathcal{M}_1 \cap (\mathcal{R}_1 \cup \mathcal{S}_1) = \{\text{complex}^1, \text{dangerous}^1, \text{decades}^1, \text{in}^1, \text{it}^1, \text{more}^1, \text{past}^1, \text{situation}^1, \text{than}^1, \text{the}^1, \text{was}^1\}$$

The resulting modified unigram precision score is $p'_1(M, R, S) = \frac{|\mathcal{C}'_1|}{|\mathcal{M}_1|} = \frac{11}{14} = 0,7857$.

Higher Order N-Grams

Modified unigram precision certainly is a similarity score, albeit a very crude one for measuring translation quality. Not taking into account word order, it does not at all capture the well-formedness of the machine translated segment, providing a perfect score of 1.0 for

***M'*:**
a and complex dangerous decades in more past situation than

which is neither a fluent nor in any sense an adequate translation, nor intelligible beyond giving a very vague hint of the topic discussed.

So, while modified unigram precision will assign a high score to a translation that is very similar to a reference translation and thus very good, some very bad translations might also obtain very high scores, rendering the metric effectively useless.

To address this problem, BLEU incorporates higher order n-grams up to length $n = 4$, i.e. substrings consisting of 2, 3 and 4 consecutive tokens, into the metric. Computation of the modified precision score using higher order n-grams is identical to calculating unigram precision:

***M*:**
the situation even more complex , more dangerous than it was in past decades

***R*:**
a situation more complicated and dangerous than it was in the previous decades

***S*:**
a situation more complex and dangerous than in past decades

$\mathcal{C}'_2 = M_2 \cap (\mathcal{R}_2 \cup \mathcal{S}_2) = \{\text{dangerous than}^1, \text{in past}^1, \text{it was}^1, \text{more complex}^1, \text{past decades}^1,$
$\text{than it}^1, \text{was in}^1\}$

$p'_2(M, R, S) = \frac{|\mathcal{C}'_2|}{|\mathcal{M}_2|} = \frac{7}{13} = 0,5385$

For 3- and 4-grams we get:

$\mathcal{C}'_3 = \{\text{dangerous than it}^1, \text{in past decades}^1, \text{it was in}^1, \text{than it was}^1\}$

$p'_3(M, R, S) = \frac{|C'_3|}{|M_3|} = \frac{4}{12} = 0,3333$

$\mathcal{C}'_4 = \{\text{dangerous than it was}^1, \text{than it was in}^1\}$

$p'_4(M, R, S) = \frac{|C'_4|}{|M_4|} = \frac{2}{11} = 0,1818$

Modified n-gram precision scores are computed up to $n = 4$ and then combined via geometric averaging.

Definition 4 (Combined Modified Precision Score) *Let $p'_1, \ldots, p'_4$ be modified 1- to 4-gram precision scores of M with reference to $R^1, \ldots R^m$. Then the* combined modified precision score *of M with reference to $R^1, \ldots R^m$ is defined as:*

$$p'_{1234} = \sqrt[4]{p'_1 \cdot p'_2 \cdot p'_3 \cdot p'_4} = exp\left[\frac{1}{4}\sum_{n=1}^{4} \ln p'_n\right]$$

In our example, this yields a combined modified precision score of $p'_{1234}(M, R, S) = \sqrt[4]{\frac{11 \cdot 7 \cdot 4 \cdot 2}{14 \cdot 13 \cdot 12 \cdot 11}} = 0,4002$ for M.

Brevity Penalty

With the geometric mean of the modified n-gram precision scores, a notion of correctness of the n-grams making up the machine translation is captured. However, the pure precision scores do not evaluate any notion of completeness of the translation. Take as an example the following incomplete machine translation:

***M"*:**
than in past decades

When scoring M'' against S, we find every single n-gram ($1 \leq n \leq 4$) in M'' is correct in the sense that it also occurs in S, yielding a maximal combined score of $p'_{1234} = 1.0$, even though the translation doesn't offer any meaningful representation of the content of the original sentence (as concluded from the reference translation).

The usual method of counterbalancing precision with recall cannot be applied straightforwardly due to BLEU's reliance on multiple reference translations. In an MT context and for a single reference translation, n-gram recall would mean the number of correct n-grams in the MT in relation to the total number of correct n-grams, i.e. the number of n-grams in the single reference translation: $\frac{|\mathcal{C}_n|}{|\mathcal{R}_n|}$. However, the definition of recall on multiple reference translations is unclear.

While BLEU can be computed for a single reference translation, much better correlation with human judgment of translation quality is reported when a number of different reference translation (four is common) is considered [?]. Hence, Papineni et.al. chose a different route and define a heuristic *brevity penalty* that lowers the score of translations that are too short in comparison to the references.

More precisely, each MT segment is compared in length to the corresponding RT segment that matches it most closely in length. If two segments of different length are equally close in length, the shorter RT segment is chosen. The length of this RT segment is referred to as *effective reference length* for the MT segment. [5]

Definition 5 (Effective Reference Length) *Let M be an MT segment and $R^1, \ldots R^m$ corresponding RT segments. The* effective reference length *for M is defined as $|\mathcal{R}^i_1|$ for the shortest $R^i, 1 \leq i \leq m$ such the absolute value of $|\mathcal{M}_1| - |\mathcal{R}^i_1|$ is minimal.*

In our example, M'' consists of 4 tokens, R of 13 tokens and S of 10 tokens. The effective reference length is therefore 10 tokens.

If the MT segment is at least as long as its effective reference length, the multiplicative brevity penalty b factor is 1. As the machine translation gets shorter, b

[5]The BLEU implementation by NIST which was used for several years in the NIST Open MT Evaluations used the shortest reference translation instead, but later reverted to the original definitions. See comments in the header of ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13a.pl (retrieved July 10th, 2011).

approaches 0, reducing the overall BLEU score substantially for translations that are much too short. Formally, the brevity penalty is defined as:

Definition 6 (Brevity Penalty) *Let M be a machine translation segment, $\mathcal{M}_1$ its unigram multiset representation and r_{eff} its effective reference length. Then the multiplicative* brevity penalty *factor b for M is defined as:*

$$b(M, r_{eff}) = \begin{cases} 1, & \text{if } |\mathcal{M}_1| > r_{eff} \\ e^{(1-\frac{r_{eff}}{|\mathcal{M}_1|})}, & \text{if } |\mathcal{M}_1| \leq r_{eff} \end{cases}$$

We can now define the final segment-level BLEU score.

Definition 7 *Let M be an MT segment, $p'_{1234}(M, R^1, \ldots, R^m)$ its combined modified precision score with reference to $R^1, \ldots, R^m$, r_{eff} its effective reference length and $b(M, r_{eff})$ its brevity penalty factor. Then its* BLEU score *is defined as:*

$$bleu(M, R^1, \ldots, R^m) = b(M, r_{eff}) \cdot p'_{1234}(M, R^1, \ldots, R^m)$$

For our example M'' with $|\mathcal{M}''_1| = 4$ and its effective reference length of 10, this results in a BLEU score of:

$$bleu(M'', R, S) = e^{(1-\frac{10}{|\mathcal{M}''_1|})} \cdot p'_{1234}(M'', R, S) = e^{(1-\frac{|10|}{|4|})} \cdot 1.0 = 0,2231$$

distinguishing it clearly from M with $|\mathcal{M}_1| = 14$, an effective reference length of 13 and an overall BLEU score of

$$bleu(M, R, S) = 1 \cdot p'_{1234}(M, R, S) = 0,4002.$$

BLEU on Blocks of Text

To determine the BLEU score on the document or corpus level, instead of mere averaging of segment level scores, a different approach is taken that allows for a better levelling out of statistical outliers. While matching itself takes place at the segment level as described above, n-gram match counts are summed up for each n over the whole document or corpus and then divided by the summed up n-gram lengths of all MT segments for the entire document or test corpus.

For the brevity penalty, the effective reference length is computed by determining the effective reference length for each MT segment separately and then summing up those values for a total effective reference length for the entire document or corpus.

These aggregate values are then used to compute a document or corpus level score using the same definition of BLEU as in the previous section, but using the aggregate values instead.

2.2.2. Discussion

BLEU in 2002 and its variant NIST, which was proposed by Doddington [2] in the same year, were the first fully automatic MT evaluation methods that were widely adopted in the MT community. BLEU and NIST proved to decently correlate with human judgement when computed over large test sets for a variety of diverse language pairs [8](section 5), [7] (section 3), [2] (section 3). They were immediately incorporated into the NIST Open MT Evaluation alongside human judgements. Automatic evaluation using BLEU and NIST took the place of the sole official OpenMT Evaluation score in 2006, effectively replacing human judgements by relegating them to a bonus scoring for a few select systems[6]

Following their adoption by one of the most important evaluation events of the MT world, BLEU and NIST quickly became the most widely used metrics, despite numerous criticisms being voiced, and various alternative metrics that were proposed in subsequent years.

NIST itself set out to address two problems with the BLEU metric [2] (section 4):

- the use of geometric averaging for the combination of modified precision scores, which makes the metric vulnerable to variance due to low match counts for longer n-gram lengths. As an extreme case, consider the illustrative example where no 4-gram match can be found within an MT segment. This results in a modified 4-gram precision score of 0.0 which is propagated by the geometric average to the segment's final BLEU score. This effect has an especially dire impact on BLEU's segment level correlation with human judgements[7]. The way scoring on corpus level is defined prevents this phenomenon from having a similar effect on the corpus level score. Doddington proposes the use of arithmetic averaging instead[8].

- the equal weighting of all n-grams, regardless of how informative they are. Doddington proposes an n-gram weight that gives more importance to those n-grams that occur less frequently in the set of reference translations.

Additional criticisms towards BLEU have been voiced by Lavie et.al. [6] (section 2.1). They claim that

[6]see Evaluation Plan for NIST Open MT Evaluation 2006, available at http://www.ldc.upenn.edu/Catalog/docs/LDC2010T17/NISTOpenMT06EvalPlan_ v4.pdf (retrieved July 10th, 2011)

[7]Using a plain geometric average, a segment consisting of less than 4 tokens always has a BLEU score of 0.0.

[8]An alternative approach is to use the geometric average with smoothing, as implemented as an option in NIST's BLEU-implementation available at ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13a.pl.

- the brevity penalty is poor compensation for the lack of recall. Their experiments suggest that metrics that place much heigher weight on recall than on precision usually perform better when compare to human judgements than metrics that place heavy weight on precision.

- the fact that BLEU does not require explicit matches may increase .the likelihood of false matches, especially for common function words

Furthermore, according to Turian et.al. [12] a numerical BLEU score might indicate some relative improvement in a system, but it is hard to gain insight into the nature of the improvement. They call for more intuitive scoring techniques that deliver results that are more informative for the development cycle.

Gimenez in his 2008 doctoral thesis [3] (section 1.2.1) points to a number of cases, where translation quality improvements due to the incorporation of linguistic knowledge were not reflected by the BLEU score. This indicates that there are some important quality aspects that BLEU does not measure.

These known issues with BLEU have lead to the proposal of host of alternative metrics, many of which were evaluated in NIST's Metrics for Machine Translation Evaluation Challenge in 2008[9]. Select metrics from Metrics MATR 2008 are described in a 2009 *Special Issue on Automated Metrics for Machine Translation Evaluation* of *Machine Translation* [1]. This issue provides a good overview over state-of-the-art MT evaluation methods.

2.2.3. Implementation

This section describes the usage and the implementation of a Haskell program that provides BLEU scoring that is available on the CD that is part of this work.

Usage

The program is called from command line, specifying at least a file containing the machine translation and a directory containing the corresponding reference translations. This directory must not contain any other files. Files are simple text files containing one segment per line. Care has to be taken that corresponding segments in machine translation and reference translations are at the same line number in the respective documents, as the implementation uses only the line number to identify corresponding segments. The syntax for calling the programm is:

```
Usage: BLEU -m MTFILE -r REFDIR [OPTION]...
```

The following options are provided:

[9]Results are publicly available at http://www.itl.nist.gov/iad/mig//tests/metricsmatr/2008/results/index.html

Short	Long	Parameter	Explanation
-h	–help		print usage information
-m		FILE	file containing the MT
-r		DIR	directory containing the RTs
	–shortest		use shortest reference for brevity penalty (default: reference closest in length)
	–arithmetic		use arithmetic averaging (default: geometric averaging)
	–nosegments		suppress segment level scoring
-n	–ngrams	NVALUES	NVALUES is a space-separated list of n-gram lengths used for scoring (default "1 2 3 4")

The default values reflect the original BLEU definitions from [8] as they are described in section 2.5.1 of this work.

No option for evalutating at document level is provided. As computation is identical as for the corpus level, this goal can be realized by passing MT and RT files that contain segments of only a single document instead of segments from an entire test corpus of several documents.

Documentation of Implementation

The implementation consists of wo modules, `Main` (dealing with command line input) and `BLEU` (handling preprocessing and computation). We will not explain the Main module in this paper, as it mainly deals with I/O that is unspecific to BLEU or MT evaluation in general. We will explain and quote select code that is relevant for computing the BLEU score. Complete documentation of all functions and data types from both modules is available can be generated from the source code using *haddock*.

Data Types All computation options provided on the command line (or the default values, if none are provided) are stored in an `Options` record:

```
data Options = Options {
        optM      :: Maybe FilePath,
        optRs     :: Maybe FilePath,
        optBP     :: BP,
        optComb   :: Combine,
        optSegs   :: Bool,
        optNVal   :: [Int]
        }
```

where `BP` and `Combine` encode options for brevity penalty and combination of the various n-gram modified precision scores:

```
data BP      = Shortest | Closest
data Combine = GeomAvg  | ArithAvg
```

BLEU as originally defined uses the RT segment closest in length to the corresponding MT segment as effective reference length for brevity penalty computation, but in practice, the shortest reference translation has been used as an alternative (see section 2.5.1).

BLEU uses geometric averaging, whereas the NIST-variant of BLEU employs arithmetic averaging. Other options might be defined to extend the program, e.g. geometric averaging with smoothing as in the NIST-implementation of BLEU[10].

Segments are stored in multiset representation. Corresponding segments from all reference translations are stored together in one list, as they are always processed together. We use the following type synonyms for greater clarity of code:

```
type MSeg   =  M.MultiSet String
type RSegs  = [M.MultiSet String]
```

We usually represent the entirety of our machine translation as a `[MSeg]` and all of our reference translations in their entirety as a `[RSegs]`.

After preprocessing (see the next paragraph) all information necessary for computation is stored in a `BLEUInput` record:

```
data BLEUInput = BLEUInput { mt            :: [[MSeg]]
                           , refs          :: [[RSegs]]
                           , mlengths      :: [Int]
                           , mNgramLengths :: [[Int]]
                           , rlengths      :: [[Int]]
                           , myOpts        :: Options
                           }
```

`mt` is a list containing several copies of the machine translation, each in n-gram multiset representation for a different value of n. `rt` represents the reference translations in the same way. `mlengths` and `rlengths` store the lengths of all machine translation and reference translation segments, respectively, in tokens. `mNgramLengths` additionally stores for each n-gram length specified in `optNval myOpts` the length of each machine translation segment in n-grams. `myOpts` contains the computation options.

After the computation, results are stored in a `BLEUOutput` record:

```
data BLEUOutput = BLEUOutput { corpusBLEU         :: Float
                             , segBLEUs           :: [Float]
                             , corpusTokenLength  :: Int
                             , segTokenLengths    :: [Int]
                             , corpusERL          :: Int
                             , segERLs            :: [Int]
                             }
```

Additionally to the actual scores at corpus and, if evaluated, at segment level, the length of the machine translation in tokens is provided for the levels evaluated and the effective reference length for both segment and corpus level.

Preprocessing Preprocessing of input is handled by `bleuPacker` (see below), which in turn makes use of three helper functions `tokenize`, `makeNgrams`, and `nGramLengths`.

[10]ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v13a.pl (retrieved July 10th, 2011)

`tokenize` and `makeNgrams` work on the segment level and are mapped by bleuPacker over the lists representing the complete translations as needed.

Tokenization is provided by `tokenize`, which takes as input a string representing a segment and returns a list of strings each representing a token.

```haskell
tokenize :: String -> [String]
```

Tokenization is performed using unicode categories. Whitespace is dropped. Symbols are treated as a separate token each. Punctuation is treated as a separate token as well, unless followed or preceeded by a number.[11]

`makeNgrams` takes an integer `n` and a tokenized segment and returns the segment's `n`-gram multiset representation.

```haskell
makeNgrams :: Int -> [String] -> M.MultiSet String
makeNgrams n tokenizedString = makeNgrams' M.empty n tokenizedString
   where makeNgrams' intermediate n tokenizedString
      | (length tokenizedString) < n = intermediate
      | otherwise                    = makeNgrams' newInter
                                                   n
                                                   (tail tokenizedString)
      where newInter = M.insert (unwords (take n tokenizedString))
                                intermediate
```

An empty `MultiSet` is recursively filled with n-grams starting from the beginning of the tokenized segment.

`nGramLengths` takes a list of values for n for which the lengths in n-grams of each segment are to be computed, and the list containing the length of these segments in tokens. It returns a list containing for each n in `nList` (line 2) the length of all segments in n-grams.

```haskell
1  nGramLengths :: [Int] -> [Int] -> [[Int]]
2  nGramLengths nList lengthList = zipWith ($)
3                                          mapSubtractFuncs
4                                          (repeat lengthList)
5     where mapSubtractFuncs     = map (map) subtractFuncs
6           subtractFuncs        = map (\n x -> x-n+1) nList
```

As we know that a segment of `x` tokens contains `x-n+1` n-grams, we can use the already available segment lengths in tokens to compute lengths in n-grams more efficiently than by calling `MultiSet.size` on each multiset representation.

`bleuPacker` ties together the previously defined functions to produce a `BLEUInput` record containing all data necessary for the computation of scores.

```haskell
bleuPacker :: Options -> String -> [String] -> BLEUInput
bleuPacker opts mRaw rRaw =  BLEUInput { mt            = m,
                                         mlengths      = ml,
                                         mNgramLengths = mNGLs,
                                         refs          = rs,
                                         rlengths      = rl,
                                         myOpts        = opts
                                       }
   where m         = zipWith ($) nGramizeM (repeat tokensM)
      rs           = zipWith ($) nGramizeR (repeat tokensR)
      nGramizeM    = map (map . makeNgrams) (optNVal opts)
      nGramizeR    = map (map) nGramizeM
```

[11]This is equivalent to the 'international-tokenization' option of the BLEU-implementation in NIST's mteval-v13a.pl script.

```
13        mNGLs        = nGramLengths (optNVal opts) mls
14        mls          = map length tokensM
15        rls          = map (map length) tokensR
16        tokensM      = map tokenize m'
17        tokensR      = map (map tokenize) rs'
18        m'           = lines mRaw
19        rs'          = transpose $ map lines rRaw
```

It takes an `Options` record as argument followed by a string, representing the complete machine translation, and a list of strings, each representing a complete reference translation. MT and RTs are split up into lines at lines 18 and 19, respectively. (Remember that each line represents a segment.) We additionally need to transpose the list of lists representing the references so that all corresponding reference segments are grouped together.

Each segment of both MT and RTs is then tokenized in lines 16 and 17.

Lines 14 and 15 compute the length in tokens of every segment.

The length of the machine translation is further processed in line 13 to additionally yield for each n-gram length specified in `optNVal opts` the lengths of each segment of the MT in n-grams.

Line 11 defines `nGramizeM` as a list of functions, one for each specified n-gram length. Each function in `nGramizeM` is of type `[[String]] -> [M.MultiSet String]`, i.e. it transforms a list of tokenized segments into a list of segments in n-gram multiset representation. In line 9, each of these functions is applied to a copy of the tokenized machine translation. Thus, `m` contains several full multiset representations of the MT, one for each specified n-gram length.

Lines 12 and 10 perform the same for the reference translations.

Scoring The computation of scores is coordinated by `bleuHandler`.

```
bleuHandler :: BLEUInput -> BLEUOutput
```

It calls the functions that perform the various computation steps and aggregates some values for further processing. Its definition is straightforward and we do not explain it here. However, we document all functions that are necessary for computing scores. The central function for scoring the input is `bleu`.

```
bleu :: Combine -> ([Int], [Int], Int, Int) -> Float
bleu comb (matchCounts
         , lengths_in_ngrams
         , mLength
         , efRefLength) = brevityPenalty * combinedPrecs
   where combinedPrecs   = case comb of
                             GeomAvg  -> geomAvg precisions
                             ArithAvg -> arithAvg precisions
         precisions      = zipWith (/)
                                   (map fromIntegral matchCounts)
                                   (map fromIntegral lengths_in_ngrams)
         brevityPenalty  = exp (min 0 (1 - (fromIntegral efRefLength) /
                                           (fromIntegral mLength)
                                      )
                               )
```

It computes a single score. In the following, we explain its use for computing segment level scores. `bleu`, as well as all other functions defined in the following, is used in exactly the same way to process aggregate values in order to produce the corpus level score. However, for greater clarity, we refrain from pointing this out for every following function.

`bleu` takes as input a combination option and a quadruple containing:

- the segment's number of n-gram matches for each n-gram length specified in the options
- the segment's lengths in n-grams for each n-gram length
- the length of the segment in tokens
- the segment's effective reference length

The modified n-gram precision scores are computed in lines 9-11 and combined in lines 6-8 according to the specified combination option, where

```
geomAvg  :: [Float] -> Float
arithAvg :: [Float] -> Float
```

compute the geometric and arithmetic average of a list of `Float` values.

The brevity penalty is computed in lines 12-15 of `bleu`. The `BP` option need not be evaluated here, because it is taken care of in the `effectiveRefLength` function:

```
 1  effectiveRefLength :: BP -> Int -> [Int] -> Int
 2  effectiveRefLength Shortest
 3                     _
 4                     rlengths  = minimum rlengths
 5  effectiveRefLength Closest
 6                     mlength
 7                     rlengths  = minimum (head lengthsByDifference)
 8     where lengthsByDifference = groupBy equal' (sortBy compare' rlengths)
 9           compare'            = compare `on` (abs . (\x -> x - mlength))
10           equal' x y          = (compare' x y) == EQ
```

It takes as input the brevity penalty option, the length of an MT segment and the list containing the lengths of all corresponding RT segments.

In case we use the length of the shortest available RT segment, the length of the MT segment is of no interest and we choose the shortest RT segment's length as effective reference length.

When using the RT segment that is closest in length to the MT segment, in line 9 we define a comparison function that compares the RTs' lengths using a value's numerical distance from the given MT segment length. The RTs lengths are grouped according to this function in line 8. If several RT lengths are at the same distance, the lower value is chosen as effective reference length in line 7.

Match counts are also determined on the segment level by `matchCounts`:

```
matchCount :: MSeg -> RSegs -> Int
matchCount m refs = M.size matches
   where matches           = M.intersection m potentialMatches
         potentialMatches = foldr M.maxUnion M.empty refs
```

The computation follows the definitions in section 2.5.1 exactly, intersecting the MT's multiset n-gram representation with the multiset union over the multiset n-gram representations of all corresponding RTs.

2.3. TER

TER[12] was proposed in 2006 by Snover et.al. [10] as an alternative evaluation method to the commonly used BLEU and NIST metrics. It measures the amount of editing necessary to transform a machine translation into the reference translation segment that it is most similar to.

While BLEU measures similarity directly, TER measures difference, with a maximal score of 1.0 indicating a very bad translation and a minimal score of 0.0 indicating a very good translation. TER, like BLEU, compares the machine translation and the reference translation segment by segment. TER scores can be computed for the segment, document and corpus level.

We describe TER as originally defined in [10] (section 3). In some places we provide more detail and definitions that are more concise. As the description in [10] allow for various different solutions to technicalities, our definitions are not always identical to the ones underlying the freely available implementation by Snover et.al. [13], which also includes some further optimisations. We will point out known differences and similarities.

In the following sections, we will first consider the notion of edit distance without shifts. Computation of minimum edit distance yields an optimal sequence of insertion, deletion and substitution operations transforming - in our case - a machine translation segment into a corresponding reference translation segment. The operation sequence implies an alignment between the two segments, which is described and used as the basis for further computations. We will then introduce and define phrasal shifts as an additional edit operation. As the exact computation of minimum edit distance with shifts has been shown to be NP-complete [9], Snover et. al. employ various heuristics for efficient computation, which we explain in subsection 2.3. Finally we will explain how TER makes use of multiple reference translations, and how computation of document and corpus level scores is realized.

Minimum Edit Distance and Alignments

Minimum edit distance measures the number of insertions, deletions and substitutions necessary to transform one sequence of symbols into another. In our case, we transform a machine translated segment, i.e. a sequence of tokens, into a sequence of tokens that is equal to a corresponding reference translation segment[14]. Minimum edit distance can be computed efficiently using dynamic programming in $\mathcal{O}(k \cdot l)$, where k and l are the length in tokens of the MT segment and the RT segment. A good description of this well-known algorithm can be found in [4] (p. 107 ff.)

[12]**T**ranslation **E**dit **R**ate, sometimes also rendered as **T**ranslation **E**rror **R**ate

[13]Available at http://www.umiacs.umd.edu/ snover/terp/ as of July 10th, 2011

[14]Snover et.al. in their implementation actually compute the transformations necessary to transform a reference into the machine translation segment. This results in the same value for minimum edit distance. The resulting operation list with operations transforming R into M can be converted to an operation list as used here by simply replacing each occurrence of `I` with `D` and vice versa.

The minimum edit distance algorithm also yields an optimal operation sequence describing the transformations. Consider the following MT segment M and a corresponding reference segment R:

***M*:**
`more complex than in the previous decades a complex situation`
***R*:**
`a more complex situation than in the past decades`

The minimum edit distance algorithm yields an operation sequence `O = I__I___S_DDD`, where `I` stands for insertion, `D` for deletion, `S` for substitution and `_` for a match. When the operations are applied to M from left to right, we obtain R:

```
M:                     more complex than in the previous decades a complex situation
ins(a):                a more complex than in the previous decades a complex situation
match(more):                                --- " ---
match(complex):                             --- " ---
ins(situation):        a more complex situation than in the previous decades a complex situation
match{than}:                                --- " ---
match{in}:                                  --- " ---
match{the}:                                 --- " ---
sub{previous -> past}: a more complex situation than in the past decades a complex situation
match{decades}:                             --- " ---
del{a}:                a more complex situation than in the past decades complex situation
del{complex}:          a more complex situation than in the past decades situation
del{situation}:        a more complex situation than in the past decades
= R:                   a more complex situation than in the past decades
```

Different weights can be given to the different operations, and weights can be made dependend upon the operation's argument, i.e. the token that is to be inserted, deleted, substituted or matched. The operation sequence output by the algorithm depends on how the weights are set. A weight function thus needs to be decided upon before computing the minimum edit distance. The original TER uses uniform weights of 1 for insertions, deletions and substitutions and a weight of 0 for matches[15].

Once we have obtained the operation sequence, we can determine the minimum edit distance adding up the weights for each operation. For our example, this yields a minimum edit distance of 6.

The operation sequence also implies an alignment that tells us, which positions in M and R correspond (i.e. arise through a substitution or a match during the transformation from M to R). Such alignments can be represented graphically; the minimum edit distance alignment of our example looks like this:

```
O: I   _     _        I       _   _  _   S          _    D    D        D
M:   more complex           than in the previous decades a complex situation
       |     |                |   |  |   |          |
R: a more complex situation than in the past     decades
```

More formally, we define alignments as follows:

[15]the implementation by Snover et.al. additionally supports the definition of customized weight functions

Definition 8 (Alignment) *Let $k, l \geq 1$. Let $M = m_1 \ldots m_k$ be an MT segment and $R = r_1 \ldots r_l$ a corresponding RT segment. Let $pos(M) = \{1, \ldots, k\}$ and $pos(R) = \{1, \ldots, l\}$ be the set of position in M and R, respectively. An* alignment *is a bijection $align : apos(M) \to apos(R)$, where $apos(M) \subseteq pos(M), apos(R) \subseteq pos(R)$ are the subsets of aligned positions in M and R. We also use $align_{M \to R}$ to refer to an alignment and $align_{R \to M}$ to its inverse.*

Note in the example above that aligned words do not have to be equal. While it seems to make sense semantically that `previous` and `past` are aligned to each other, this is merely a chance effect. We would have obtained the same operation sequence and thus the same alignment if M had `kitten` in the 6th position instead of `previous`.

The alignment in the example in fact isn't very good. `a` and `situation` remain unaligned, even though they occur in both M and R. Yet, it is the best that the minimum edit distance computation can provide. The alignment would be more reasonable semantically, and the edit distance reduced by 2, if we would allow crossing alignments. Crossing alignments can be emulated by allowing shifts as an additional operation.

Phrasal Shifts

Edit distance without shifts does not account very well for the fact that in natural language, phrases can oftentimes be shifted around within a sentence without compromising fluency or intelligibility (too much). TER seeks to remedy this by allowing the shifting of phrases as an additional operation.

In the example above, by shifting the phrase `situation` towards the left to position 2 within M, we obtain:

***M'*:**
```
more complex situation complex than in the previous decades a
```

Running the minimum edit distance algorithm on M' and R we obtain the following operation list and alignment.

```
O: I   _     _        _        D      _  _    _     S        _    D
M’:  more complex situation complex than in  the previous decades a
       |     |        |               |  |    |     |        |
R: a more complex situation         than in  the   past   decades
```

This alignment is somewhat more reasonable semantically, and it takes only 4 insertions, deletions and substitutions to transform M' into R. TER uses a uniform weight of 1 for all shift operations regardless of the length of the shifted phrase or how far it is shifted within M. This leads to an overall edit distance of 5, if we first perform the shift and then all insertions, deletions and substitutions.

Formally, the shift operation can be defined as follows:

Definition 9 (Shift Operation) *Let $M = m_1 \dots m_k, k \geq 1$ be a sequence. Let $S = m_i \dots m_{i'}, 1 \leq i \leq i' \leq k$ be the subsequence of M to be shifted and $|S| = i' - i + 1$ its length. We define X as the sequence that remains after deleting S from M, i.e. $X = x_1 \dots x_{k-|S|} = m_1 \dots m_{i-1} m_{i'+1} \dots m_k$. Let $1 \leq p \leq k - |S| + 1$ be a position in M. Then the sequence obtained from M by shifting S to position p is defined as*

$$shift(M, S, p) = x_1 \dots x_{p-1} S x_p \dots x_{k-|S|}$$

For our example we get:

```
position:            1    2       3          4       5    6        7        8        9       10
M                  = more complex than       in      the  previous decades  a        complex situation
S                  =                                                                 COMPLEX SITUATION
X                  = more complex than       in      the  previous decades  a
shift(M,S,2) = M’  = more COMPLEX SITUATION complex than in       the      previous decades a
```

To obtain semantically reasonable alignments it would suffice to allow for the shifting of single tokens, as any phrasal shift can be expressed as a series of single token shifts. However, this would not reduce the edit distance by much; at least it would make it dependend on the number of words, favoring shorter shifts over longer ones. This is not in correspondence with our notion of grammaticality that places greater value on the shifted phrases being grammatical than on their length[16].

Heuristics

Optimal computation of minimum edit distance with shifts has been shown to be NP-complete by [9]. To allow for efficient computation, Snover et.al. use various heuristics.

The computation of edit distance is divided up into two phases. First a sequence of beneficial shifts is determined using a greedy search. After performing the shifts, minimum edit distance is computed[17].

In the shift phase, rather than choosing from all possible shifts, electable shifts must fulfill three conditions. In order to define them, we need the additional concept of *misalignments*.

[16]E.g. ”as I a had too much coffee today, I can’t go to sleep now” and ”I can’t go to sleep now, as I had too much coffee today” are almost equally correct, despite the considerable length of the shift necessary to transform one sentence into the other.

[17]Snover et.al. make use of a beam search with the dynamic programming algorithm for dealing efficiently with very long segments.

Definition 10 (Misalignments) *Let $M = m_1 \dots m_k$ be an MT segment, $R = r_1 \dots r_l$ a corresponding RT segment and align : $pos(M) \rightarrow pos(R)$ an alignment for M and R.*

We say there is a misalignment at position $i, 1 \leq i \leq k$ in M with respect to align if one of the following conditions holds:

1. *there is no j such that $align_{M \rightarrow R}(i) = j$ (deletion)*
2. *$align_{M \rightarrow R}(i) = j$ and $m_i \neq r_j$ (substitution)*

We say there is a misalignment in R at position $j, 1 \leq j \leq l$ with respect to align if one of the following conditions holds:

1. *there is no i with $align_{R \rightarrow M}(j) = i$ (insertion)*
2. *$align_{R \rightarrow M}(j) = i$ and $r_j \neq m_i$ (substitution)*

We can now define the electable shifts:

Definition 11 (Electable Shifts) *Let $M = m_1 \dots m_k$ be an MT segment, $R = r_1 \dots r_l$ a corresponding RT segment, align : $pos(M) \rightarrow pos(R)$ an alignment for R and M. Let $S = m_i \dots m_{i'}$ be a subsequence of M. Let $1 \leq p \leq k - (i' - i)$ and $p = align_{R \rightarrow M}(j)$. $shift(M, S, p)$ is an electable shift if all of the following conditions are met:*

- *there is a subsequence $R_S = r_j \dots r_{j'}$ of R such that $S = r_j \dots r_{j'}$*
- *there is a misalignment in M in at least one position $q_M, i \leq q_M \leq i'$*
- *there is a misalignment in R in at least one position $q_R, j \leq q_R \leq j'$*

TER computes the sequence of beneficial shifts by trying out all electable shift and computing what the minimum edit distance in insertions, deletions and substitutions would be after the shift. It chooses the most advantageous shift that improves the minimum edit distance by at least one. If several shifts yield equal improvements, then one of the longest shifts is chosen. This process is repeated until no more beneficial shifts are found.

The sequence of beneficial shifts is then applied to M and the minimum edit distance in insertions, deletions and substitutions computed for the result. The

TER edit distance between M and R is then defined as the total number of shift, insertion, deletion and substitution operations performed to transform M into R.

If several corresponding RT segments are available for M, the edit distance is computed individually for each of them and the lowest one is chosen.

The TER score is then defined as follows:

Definition 12 (Segment-level TER Score) . *Let M be an MT segment and $R^1, \ldots, R^m$ corresponding RT segments. Let $ed(M, R^1, \ldots, R^m)$ be the TER edit distance of M with respect to $R^1, \ldots, R^m$ and l the average length of $R^1, \ldots, R^m$. Then the segment level TER-score is defined as:*

$$ter(M, R^1, \ldots, R^m) = \frac{ed(M, R^1, \ldots, R^m)}{l}$$

For the corpus- and document-level TER scores an MT segment length weighted average of all individual segment TER scores is computed.

A. Appendix: Documentation of Haskell Functions and Data Types

The following short documentation was excerpted and adapted from the documentation at haskell.org. Within each section they are listed in alphabetical order.

A.1. From Data.Char [1]

```
isNumber :: Char -> Bool
```

Returns `True` for any Unicode space character, and the control characters `\t, \n, \r, \f, \v`.

```
isPunctuation :: Char -> Bool
```

Selects Unicode punctuation characters, including various kinds of connectors, brackets and quotes.

```
isSpace :: Char -> Bool
```

Selects white-space characters in the Latin-1 range. (In Unicode terms, this includes spaces and some control characters.)

```
isSymbol :: Char -> Bool
```

Selects Unicode symbol characters, including mathematical and currency symbols.

A.2. From Data.List [2]

```
filter :: (a -> Bool) -> [a] -> [a]
```

`filter`, applied to a predicate and a list, returns the list of those elements that satisfy the predicate.

```
foldl :: (a -> b -> a) -> a -> [b] -> a
```

`foldl`, applied to a binary operator, a starting value (typically the left-identity of the operator), and a list, reduces the list using the binary operator, from left to right. The list must be finite.

[1] From http://haskell.org/ghc/docs/latest/html/libraries/base/Data-Char.html (retrieved on July 10th, 2011)

[2] From http://haskell.org/ghc/docs/latest/html/libraries/base/Data-List.html (retrieved on July 10th, 2011)

```
foldr :: (a -> b -> b) -> b -> [a] -> b
```

`foldr`, applied to a binary operator, a starting value (typically the right-identity of the operator), and a list, reduces the list using the binary operator, from right to left. The list must be finite.

```
groupBy :: (a -> a -> Bool) -> [a] -> [[a]]
```

The `groupBy` function takes an equality test and a list and returns a list of lists such that the concatenation of the result is equal to the list argument. Moreover, each sublist in the result contains only elements that are equal by the supplied equality test.

```
head :: [a] -> a
```

Extracts the first element of a list, which must be non-empty.

```
length :: [a] -> Int
```

Returns the length of a finite list.

```
lines :: String -> [String]
```

Breaks a string up into a list of strings at newline characters. The resulting strings do not contain newlines.

```
map :: (a -> b) -> [a] -> [b]
```

`map f xs` is the list obtained by applying `f` to each element of `xs`.

```
minimum :: Ord a => [a] -> a
```

Returns the minimum value from a list, which must be non-empty, finite, and of an ordered type.

```
repeat :: a -> [a]
```

`repeat x` is an infinite list, with `x` the value of every element.

```
reverse :: [a] -> [a]
```

`reverse xs` returns the elements of `xs` in reverse order. `xs` must be finite.

```
sortBy :: (a -> a -> Ordering) -> [a] -> [a]
```

Sorts a list using the provided comparison function.

```
sum :: Num a => [a] -> a
```

Computes the sum of a finite list of numbers.

```
tail :: [a] -> [a]
```

Extracts the elements after the head of a list, which must be non-empty.

```
take :: Int -> [a] -> [a]
```

`take n`, applied to a list `xs`, returns the prefix of `xs` of length `n`, or `xs` itself if `n > length xs`.

```
transpose :: [[a]] -> [[a]]
```

Transposes the rows and columns of its argument.
For example, `transpose [[1,2,3],[4,5,6]] == [[1,4],[2,5],[3,6]]`

```
unwords :: [String] -> String
```

`unwords` is an inverse operation to `words`. It joins words with separating spaces.

```
words :: String -> [String]
```

Breaks a string up into a list of words, which were delimited by white space.

```
zip4 :: [a] -> [b] -> [c] -> [d] -> [(a, b, c, d)]
```

Takes four lists and returns a list of corresponding quadruples. If one input list is shorter, excess elements of the longer lists are discarded.

```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
```

Generalises `zip` by zipping with the function given as the first argument, instead of a tupling function. (`zip` takes two lists and returns a list of corresponding pairs. If one input list is short, excess elements of the longer list are discarded.) For example, `zipWith (+)` is applied to two lists to produce the list of corresponding sums.

A.3. From Data.Maybe [3]

```
data Maybe a
```

Encapsulates an optional value. A value of type `Maybe a` either contains a value of type `a` (represented as `Just a`), or it is empty (represented as `Nothing`).

```
fromJust :: Maybe a -> a
```

Extracts the element out of a `Just` and throws an error if its argument is `Nothing`.

A.4. From Data.MultiSet [4]

```
empty :: MultiSet a
```

The empty multiset.

```
insert :: Ord a => a -> MultiSet a -> MultiSet a
```

Insert an element in a multiset.

```
intersection :: Ord a => MultiSet a -> MultiSet a -> MultiSet a
```

The intersection of two multisets. The number of occurrences of each element in the intersection is the minimum of the number of occurrences in the arguments.

```
maxUnion :: Ord a => MultiSet a -> MultiSet a -> MultiSet a
```

The union of two multisets. The number of occurences of each element in the union is the maximum of the number of occurrences in the arguments.

```
size :: MultiSet a -> Occur
```

[3]From http://haskell.org/ghc/docs/latest/html/libraries/base/Data-Maybe.html (retrieved July 10th, 2011

[4]From http://hackage.haskell.org/packages/archive/multiset/0.2.1/doc/html/Data-MultiSet.html (retrieved July 10th, 2011)

The number of elements in the multiset.

A.5. From System.Console.GetOpt [5]

```
data ArgDescr a
```

Describes whether an option takes an argument or not, and if so how the argument is injected into a value of type a. Constructors:

- `NoArg a`: no argument expected
- `ReqArg (String -> a) String`: option requires argument
- `(Maybe String -> a) String`: optional argument

```
data ArgOrder a
```

What to do with options following non-options. Constructors:

- `RequireOrder`: no option processing after first non-option
- `Permute`: freely intersperse options and non-options
- `ReturnInOrder (String -> a)`: wrap non-options into options

```
getOpt :: ArgOrder a -> [OptDescr a] -> [String] -> ([a], [String], [String])
```

Processes the command-line, and returns a triple consisting of the option arguments, a list of non-options, and a list of error messages. The arguments are:

- The order requirements (see `ArgOrder`)
- The option descriptions (see `OptDescr`)
- The actual command line arguments

```
data OptDescr a
```

Each `OptDescr` describes a single option. Constructor: `Constructor: Option [Char] [String] (ArgDescr a) String`, where the arguments to `Option` are:

- list of short option characters
- list of long option strings (without ”–”)
- argument descriptor
- explanation of option for user

```
usageInfo :: String -> [OptDescr a] -> String
```

Returns a string describing the usage of a command, derived from the header (first argument) and the options described by the second argument.

[5]From http://www.haskell.org/ghc/docs/7.0.3/html/libraries/base-4.3.1.0/System-Console-GetOpt.html (retrieved July 10th, 2011)

Bibliography

[1] Machine translation, 2009.

[2] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, HLT '02, pages 138–145, San Francisco, CA, USA, 2002.

[3] Jesus Gimenez. *Empirical Machine Translation and its Evaluation.* PhD thesis, Universitat Politecnica de Catalunya, 2008.

[4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition).* Prentice Hall, 2008.

[5] Alon Lavie and Abhaya Agarwal. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL 2005 Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*, pages 65–72, 2005.

[6] Alon Lavie, Kenji Sagae, and Shyamsundar Jayaraman. The significance of recall in automatic metrics for mt evaluation. In *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas (AMTA 2004)*, pages 134–143, 2004.

[7] Kishore Papineni, Salim Roukos, Todd Ward, John Henderson, and Florence Reeder. Corpus-based comprehensive and diagnostic mt evaluation: initial arabic, chinese, french, and spanish results. In *Proceedings of the second international conference on Human Language Technology Research*, HLT '02, pages 132–137, San Francisco, CA, USA, 2002.

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Association for Computational Linguistics*, pages 311–318, 2002.

[9] Dana Shapira and James A. Storer. Edit distance with move operations. In *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching*, pages 85–98, 2002.

[10] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and Ralph Weischedel. A study of translation error rate with targeted human annotation. In *Proceedings of the Association for Machine Translation in the Americas (AMTA 2006)*, 2006.

[11] Henry S. Thompson. Automatic evaluation of translation quality: Outline of methodology and report on pilot experiment. In *(ISSCO) Proceedings of the Evaluators' Forum*, pages 215–223, 1991.

[12] Joseph Turian, Luke Shen, and I. Dan Melamed. Evaluation of machine translation and its evaluation. In *Proceedings of MT Summit IX*, pages 386–393, 2003.