Language-independent Compound Splitting with Morphological Operations

Klaus Macherey[1] **Andrew M. Dai**[2] **David Talbot**[1] **Ashok C. Popat**[1] **Franz Och**[1]

[1]Google Inc.
1600 Amphitheatre Pkwy.
Mountain View, CA 94043, USA
{kmach,talbot,popat,och}@google.com

[2]University of Edinburgh
10 Crichton Street
Edinburgh, UK EH8 9AB
a.dai@ed.ac.uk

Abstract

Translating compounds is an important problem in machine translation. Since many compounds have not been observed during training, they pose a challenge for translation systems. Previous decompounding methods have often been restricted to a small set of languages as they cannot deal with more complex compound forming processes. We present a novel and unsupervised method to learn the compound parts and morphological operations needed to split compounds into their compound parts. The method uses a bilingual corpus to learn the morphological operations required to split a compound into its parts. Furthermore, monolingual corpora are used to learn and filter the set of compound part candidates. We evaluate our method within a machine translation task and show significant improvements for various languages to show the versatility of the approach.

1 Introduction

A compound is a lexeme that consists of more than one stem. Informally, a compound is a combination of two or more words that function as a single unit of meaning. Some compounds are written as space-separated words, which are called open compounds (e.g. *hard drive*), while others are written as single words, which are called closed compounds (e.g. *wallpaper*). In this paper, we shall focus only on closed compounds because open compounds do not require further splitting.

The objective of compound splitting is to split a compound into its corresponding sequence of constituents. If we look at how compounds are created from lexemes in the first place, we find that for some languages, compounds are formed by concatenating existing words, while in other languages compounding additionally involves certain morphological operations. These morphological operations can become very complex as we illustrate in the following case studies.

1.1 Case Studies

Below, we look at splitting compounds from 3 different languages. The examples introduce in part the notation used for the decision rule outlined in Section 3.1.

1.1.1 English Compound Splitting

The word *flowerpot* can appear as a closed or open compound in English texts. To automatically split the closed form we have to try out every split point and choose the split with minimal costs according to a cost function. Let's assume that we already know that *flowerpot* must be split into two parts. Then we have to position two split points that mark the end of each part (one is always reserved for the last character position). The *number* of split points is denoted by K (i.e. $K = 2$), while the *position* of split points is denoted by n_1 and n_2. Since *flowerpot* consists of 9 characters, we have 8 possibilities to position split point n_1 within the characters $c_1, \ldots, c_8$. The final split point corresponds with the last character, that is, $n_2 = 9$. Trying out all possible single splits results in the following candidates:

$$\begin{aligned}
&\text{flowerpot} \rightarrow \text{f} + \text{lowerpot} \\
&\text{flowerpot} \rightarrow \text{fl} + \text{owerpot} \\
&\qquad\vdots \\
&\text{flowerpot} \rightarrow \textbf{flower} + \textbf{pot} \\
&\qquad\vdots \\
&\text{flowerpot} \rightarrow \text{flowerpo} + \text{t}
\end{aligned}$$

Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pages 1395–1404,
Portland, Oregon, June 19-24, 2011.

If we associate each compound part candidate with a cost that reflects how frequent this part occurs in a large collection of English texts, we expect that the correct split *flower* + *pot* will have the lowest cost.

1.1.2 German Compound Splitting

The previous example covered a case where the compound is constructed by directly concatenating the compound parts. While this works well for English, other languages require additional morphological operations. To demonstrate, we look at the German compound *Verkehrszeichen* (traffic sign) which consists of the two nouns *Verkehr* (traffic) and *Zeichen* (sign). Let's assume that we want to split this word into 3 parts, that is, $K = 3$. Then, we get the following candidates.

$$\begin{aligned}
&\text{Verkehrszeichen} \rightarrow \text{V} + \text{e} + \text{rkehrszeichen}\\
&\text{Verkehrszeichen} \rightarrow \text{V} + \text{er} + \text{kehrszeichen}\\
&\qquad\vdots\\
&\text{Verkehrszeichen} \rightarrow \textbf{Verkehr} + \textbf{s} + \textbf{zeichen}\\
&\qquad\vdots\\
&\text{Verkehrszeichen} \rightarrow \text{Verkehrszeich} + \text{e} + \text{n}
\end{aligned}$$

Using the same procedure as described before, we can lookup the compound parts in a dictionary or determine their frequency from large text collections. This yields the optimal split points $n_1 = 7, n_2 = 8, n_3 = 15$. The interesting part here is the additional s morpheme, which is called a linking morpheme, because it combines the two compound parts to form the compound *Verkehrszeichen*. If we have a list of all possible linking morphemes, we can hypothesize them between two ordinary compound parts.

1.1.3 Greek Compound Splitting

The previous example required the insertion of a linking morpheme between two compound parts. We shall now look at a more complicated morphological operation. The Greek compound χαρτόκουτο (*cardboard box*) consists of the two parts χαρτί (*paper*) and κουτί (*box*). Here, the problem is that the parts χαρτό and κουτο are not valid words in Greek. To lookup the correct words, we must substitute the suffix of the compound part candidates with some other morphemes. If we allow the compound part candidates to be transformed by some morphological operation, we can lookup the transformed compound parts in a dictionary or determine their frequencies in some large collection of Greek texts. Let's assume that we need only one split point. Then this yields the following compound part candidates:

$$\begin{aligned}
&\text{χαρτόκουτο} \rightarrow \text{χ} + \text{αρτόκουτο}\\
&\text{χαρτόκουτο} \rightarrow \text{χ} + \text{αρτίκουτο} \quad g_2 : \text{ό} \,/\, \text{ί}\\
&\text{χαρτόκουτο} \rightarrow \text{χ} + \text{αρτόκουτί} \quad g_2 : \text{ο} \,/\, \text{ί}\\
&\qquad\vdots\\
&\text{χαρτόκουτο} \rightarrow \textbf{χαρτί} + \textbf{κουτί} \quad g_1 : \text{ό} \,/\, \text{ί}\,, \quad g_2 : \text{ο} \,/\, \text{ί}\\
&\qquad\vdots\\
&\text{χαρτόκουτο} \rightarrow \text{χαρτίκουτ} + \text{ο} \quad g_1 : \text{ό} \,/\, \text{ί}\\
&\text{χαρτόκουτο} \rightarrow \text{χαρτίκουτ} + \text{ί} \quad g_2 : \text{ο} \,/\, \text{ί}
\end{aligned}$$

Here, $g_k : s/t$ denotes the kth compound part which is obtained by replacing string s with string t in the original string, resulting in the transformed part g_k.

1.2 Problems and Objectives

Our goal is to design a language-independent compound splitter that is useful for machine translation. The previous examples addressed the importance of a cost function that favors valid compound parts versus invalid ones. In addition, the examples have shown that, depending on the language, the morphological operations can become very complex. For most Germanic languages like Danish, German, or Swedish, the list of possible linking morphemes is rather small and can be provided manually. However, in general, these lists can become very large, and language experts who could provide such lists might not be at our disposal. Because it seems infeasible to list the morphological operations explicitly, we want to find and extract those operations automatically in an unsupervised way and provide them as an additional knowledge source to the decompounding algorithm.

Another problem is how to evaluate the quality of the compound splitter. One way is to compile for every language a large collection of compounds together with their valid splits and to measure the proportion of correctly split compounds. Unfortunately, such lists do not exist for many languages.

While the training algorithm for our compound splitter shall be unsupervised, the evaluation data needs to be verified by human experts. Since we are interested in improving machine translation and to circumvent the problem of explicitly annotating compounds, we evaluate the compound splitter within a machine translation task. By decompounding training and test data of a machine translation system, we expect an increase in the number of matching phrase table entries, resulting in better translation quality measured in BLEU score (Papineni et al., 2002). If BLEU score is sensitive enough to measure the quality improvements obtained from decompounding, there is no need to generate a separate gold standard for compounds.

Finally, we do not want to split non-compounds and named entities because we expect them to be translated non-compositionally. For example, the German word *Deutschland* (Germany) could be split into two parts *Deutsch* (German) + *Land* (country). Although this is a valid split, named entities should be kept as single units. An example for a non-compound is the German participle *vereinbart* (agreed) which could be wrongly split into the parts *Verein* (club) + *Bart* (beard). To avoid overly eager splitting, we will compile a list of non-compounds in an unsupervised way that serves as an exception list for the compound splitter. To summarize, we aim to solve the following problems:

- Define a cost function that favors valid compound parts and rejects invalid ones.
- Learn morphological operations, which is important for languages that have complex compound forming processes.
- Apply compound splitting to machine translation to aid in translation of compounds that have not been seen in the bilingual training data.
- Avoid splitting non-compounds and named entities as this may result in wrong translations.

2 Related work

Previous work concerning decompounding can be divided into two categories: monolingual and bilingual approaches.

Brown (2002) describes a corpus-driven approach for splitting compounds in a German-English translation task derived from a medical domain. A large proportion of the tokens in both texts are cognates with a Latin or Greek etymological origin. While the English text keeps the cognates as separate tokens, they are combined into compounds in the German text. To split these compounds, the author compares both the German and the English cognates on a character level to find reasonable split points. The algorithm described by the author consists of a sequence of *if-then-else* conditions that are applied on the two cognates to find the split points. Furthermore, since the method relies on finding similar character sequences between both the source and the target tokens, the approach is restricted to cognates and cannot be applied to split more complex compounds.

Koehn and Knight (2003) present a frequency-based approach to compound splitting for German. The compound parts and their frequencies are estimated from a monolingual corpus. As an extension to the frequency approach, the authors describe a bilingual approach where they use a dictionary extracted from parallel data to find better split options. The authors allow only two linking morphemes between compound parts and a few letters that can be dropped. In contrast to our approach, those operations are not learned automatically, but must be provided explicitly.

Garera and Yarowsky (2008) propose an approach to translate compounds without the need for bilingual training texts. The compound splitting procedure mainly follows the approach from (Brown, 2002) and (Koehn and Knight, 2003), so the emphasis is put on finding correct translations for compounds. To accomplish this, the authors use cross-language compound evidence obtained from bilingual dictionaries. In addition, the authors describe a simple way to learn glue characters by allowing the deletion of up to two middle and two end characters.[1] More complex morphological operations are not taken into account.

Alfonseca et al. (2008b) describe a state-of-the-art German compound splitter that is particularly robust with respect to noise and spelling errors. The compound splitter is trained on monolingual data. Besides applying frequency and probability-based methods, the authors also take the mutual information of compound parts into account. In addition, the

[1] However, the glue characters found by this procedure seem to be biased for at least German and Albanian. A very frequent glue morpheme like *-es-* is not listed, while glue morphemes like *-k-* and *-h-* rank very high, although they are invalid glue morphemes for German. Albanian shows similar problems.

authors look for compound parts that occur in different anchor texts pointing to the same document. All these signals are combined and the weights are trained using a support vector machine classifier. Alfonseca et al. (2008a) apply this compound splitter on various other Germanic languages.

Dyer (2009) applies a maximum entropy model of compound splitting to generate segmentation lattices that serve as input to a translation system. To train the model, reference segmentations are required. Here, we produce only single best segmentations, but otherwise do not rely on reference segmentations.

3 Compound Splitting Algorithm

In this section, we describe the underlying optimization problem and the algorithm used to split a token into its compound parts. Starting from Bayes' decision rule, we develop the Bellman equation and formulate a dynamic programming-based algorithm that takes a word as input and outputs the constituent compound parts. We discuss the procedure used to extract compound parts from monolingual texts and to learn the morphological operations using bilingual corpora.

3.1 Decision Rule for Compound Splitting

Given a token $w = c_1, \ldots, c_N = c_1^N$ consisting of a sequence of N characters c_i, the objective function is to find the optimal number $\hat{K}$ and sequence of split points $\hat{n}_0^{\hat{K}}$ such that the subwords are the constituents of the token, where[2] $n_0 := 0$ and $n_K := N$:

$$
\begin{aligned}
w = c_1^N \rightarrow (\hat{K}, \hat{n}_0^{\hat{K}}) &= \\
&= \arg\max_{K, n_0^K} \left\{ \Pr(c_1^N, K, n_0^K) \right\} \qquad (1) \\
&= \arg\max_{K, n_0^K} \left\{ \Pr(K) \cdot \Pr(c_1^N, n_0^K | K) \right\} \\
&\approx \arg\max_{K, n_0^K} \Big\{ p(K) \cdot \prod_{k=1}^{K} p(c_{n_{k-1}+1}^{n_k}, n_{k-1} | K) \cdot \\
&\qquad \cdot p(n_k | n_{k-1}, K) \Big\} \qquad (2)
\end{aligned}
$$

with $p(n_0) = p(n_K|\cdot) \equiv 1$. Equation 2 requires that token w can be fully decomposed into a sequence of lexemes, the compound parts. Thus, determining the optimal segmentation is sufficient for finding the constituents. While this may work for some languages, the subwords are not valid words in general as discussed in Section 1.1.3. Therefore, we allow the lexemes to be the result of a transformation process, where the transformed lexemes are denoted by g_1^K. This leads to the following refined decision rule:

$$
\begin{aligned}
w = c_1^N \rightarrow (\hat{K}, \hat{n}_0^{\hat{K}}, \hat{g}_1^{\hat{K}}) &= \\
&= \arg\max_{K, n_0^K, g_1^K} \left\{ \Pr(c_1^N, K, n_0^K, g_1^K) \right\} \qquad (3) \\
&= \arg\max_{K, n_0^K, g_1^K} \left\{ \Pr(K) \cdot \Pr(c_1^N, n_0^K, g_1^K | K) \right\} \qquad (4) \\
&\approx \arg\max_{K, n_0^K, g_1^K} \Big\{ p(K) \cdot \prod_{k=1}^{K} \underbrace{p(c_{n_{k-1}+1}^{n_k}, n_{k-1}, g_k | K)}_{\text{compound part probability}} \cdot \\
&\qquad \cdot p(n_k | n_{k-1}, K) \Big\} \qquad (5)
\end{aligned}
$$

The compound part probability is a zero-order model. If we penalize each split with a constant split penalty ξ, and make the probability independent of the number of splits K, we arrive at the following decision rule:

$$
\begin{aligned}
w = c_1^N \rightarrow (\hat{K}, \hat{n}_1^{\hat{K}}, \hat{g}_1^{\hat{K}}) &\\
= \arg\max_{K, n_0^K, g_1^K} \Big\{ \xi^K \cdot \prod_{k=1}^{K} p(c_{n_{k-1}+1}^{n_k}, n_{k-1}, g_k) \Big\} \qquad (6)
\end{aligned}
$$

3.2 Dynamic Programming

We use dynamic programming to find the optimal split sequence. Each split infers certain costs that are determined by a cost function. The total costs of a decomposed word can be computed from the individual costs of the component parts. For the dynamic programming approach, we define the following auxiliary function $\mathcal{Q}$ with $n_k = j$:

$$
\mathcal{Q}(c_1^j) = \max_{n_0^k, g_1^k} \Big\{ \xi^k \cdot \prod_{\kappa=1}^{k} p(c_{n_{\kappa-1}+1}^{n_\kappa}, n_{\kappa-1}, g_\kappa) \Big\}
$$

that is, $\mathcal{Q}(c_1^j)$ is equal to the minimal costs (maximum probability) that we assign to the prefix string c_1^j where we have used k split points at positions n_1^k. This yields the following recursive equation:

$$
\begin{aligned}
\mathcal{Q}(c_1^j) = \max_{n_k, g_k} \Big\{ & \xi \cdot \mathcal{Q}(c_1^{n_{k-1}}) \cdot \\
& \cdot p(c_{n_{k-1}+1}^{n_k}, n_{k-1}, g_k) \Big\} \qquad (7)
\end{aligned}
$$

[2] For algorithmic reasons, we use the start position 0 to represent a fictitious start symbol before the first character of the word.

Algorithm 1 Compound splitting

Input: input word $w = c_1^N$
Output: compound parts
  $\mathcal{Q}(0) = 0$
  $\mathcal{Q}(1) = \cdots = \mathcal{Q}(N) = \infty$
  for $i = 0, \ldots, N-1$ **do**
    for $j = i+1, \ldots, N$ **do**
      split-costs $= \mathcal{Q}(i) + \text{cost}(c_{i+1}^j, i, g_j) +$ split-penalty
      if split-costs $< \mathcal{Q}(j)$ **then**
        $\mathcal{Q}(j) =$ split-costs
        $\mathcal{B}(j) = (i, g_j)$
      end if
    end for
  end for

with backpointer

$$\mathcal{B}(j) = \arg\max_{n_k, g_k} \left\{ \xi \cdot \mathcal{Q}(c_1^{n_{k-1}}) \cdot \cdot p(c_{n_{k-1}+1}^{n_k}, n_{k-1}, g_k) \right\} \quad (8)$$

Using logarithms in Equations 7 and 8, we can interpret the quantities as additive costs rather than probabilities. This yields Algorithm 1, which is quadratic in the length of the input string. By enforcing that each compound part does not exceed a predefined constant length ℓ, we can change the second **for** loop as follows:

for $j = i+1, \ldots, \min(i+\ell, N)$ **do**

With this change, Algorithm 1 becomes linear in the length of the input word, $O(|w|)$.

4 Cost Function and Knowledge Sources

The performance of Algorithm 1 depends on the cost function cost$(\cdot)$, that is, the probability $p(c_{n_{k-1}+1}^{n_k}, n_{k-1}, g_k)$. This cost function incorporates knowledge about morpheme transformations, morpheme positions within a compound part, and the compound parts themselves.

4.1 Learning Morphological Operations using Phrase Tables

Let s and t be strings of the (source) language alphabet $\mathcal{A}$. A morphological operation s/t is a pair of strings $s, t \in \mathcal{A}^*$, where s is replaced by t. With the usual definition of the Kleene operator $*$, s and t can be empty, denoted by ε. An example for such a pair is ε/es, which models the linking morpheme *es* in the German compound *Bundesagentur* (federal agency):

$$\text{Bundesagentur} \rightarrow \text{Bund} + \text{es} + \text{Agentur} .$$

Note that by replacing either s or t with ε, we can model insertions or deletions of morphemes. The explicit dependence on position n_{k-1} in Equation 6 allows us to determine if we are at the beginning, in the middle, or at the end of a token. Thus, we can distinguish between start, middle, or end morphemes and hypothesize them during search.[3] Although not explicitly listed in Algorithm 1, we disallow sequences of linking morphemes. This can be achieved by setting the costs to infinity for those morpheme hypotheses, which directly succeed another morpheme hypothesis.

To learn the morphological operations involved in compounding, we determine the differences between a compound and its compound parts. This can be done by computing the Levenshtein distance between the compound and its compound parts, with the allowable edit operations being insertion, deletion, or substitution of one or more characters. If we store the current and previous characters, edit operation and the location (prefix, infix or suffix) at each position during calculation of the Levenshtein distance then we can obtain the morphological operations required for compounding. Applying the inverse operations, that is, replacing t with s yields the operation required for decompounding.

4.1.1 Finding Compounds and their Parts

To learn the morphological operations, we need compounds together with their compound parts. The basic idea of finding compound candidates and their compound parts in a bilingual setting are related to the ideas presented in (Garera and Yarowsky, 2008). Here, we use phrase tables rather than dictionaries. Although phrase tables might contain more noise, we believe that overall phrase tables cover more phenomena of translations than what can be found in dictionaries. The procedure is as follows. We are given a phrase table that provides translations for phrases from a source language l into English and from English into l. Under the assumption that English does not contain many closed compounds, we can search

[3] We jointly optimize over K and the split points n_k, so we know that $c_{n_{K-1}}^{n_K}$ is a suffix of w.

the phrase table for those single-token source words f in language l, which translate into multi-token English phrases $e_1, \ldots, e_n$ for $n > 1$. This results in a list of $(f; e_1, \ldots, e_n)$ pairs, which are potential compound candidates together with their English translations. If for each pair, we take each token e_i from the English (multi-token) phrase and lookup the corresponding translation for language l to get g_i, we should find entries that have at least some partial match with the original source word f, if f is a true compound. Because the translation phrase table was generated automatically during the training of a multi-language translation system, there is no guarantee that the original translations are correct. Thus, the bilingual extraction procedure is subject to introduce a certain amount of noise. To mitigate this, thresholds such as minimum edit distance between the potential compound and its parts, minimum co-occurrence frequencies for the selected bilingual phrase pairs and minimum source and target word lengths are used to reduce the noise at the expense of finding fewer compounds. Those entries that obey these constraints are output as triples of form:

$$(f; e_1, \ldots, e_n; g_1, \ldots, g_n) \tag{9}$$

where

- f is likely to be a compound,
- $e_1, \ldots, e_n$ is the English translation, and
- $g_1, \ldots, g_n$ are the compound parts of f.

The following example for German illustrates the process. Suppose that the most probable translation for *Überweisungsbetrag* is *transfer amount* using the phrase table. We then look up the translation back to German for each translated token: *transfer* translates to *Überweisung* and *amount* translates to *Betrag*. We then calculate the distance between all permutations of the parts and the original compound and choose the one with the lowest distance and highest translation probability: *Überweisung Betrag*.

4.2 Monolingual Extraction of Compound Parts

The most important knowledge source required for Algorithm 1 is a word-frequency list of compound parts that is used to compute the split costs. The procedure described in Section 4.1.1 is useful for learning morphological operations, but it is not sufficient to extract an exhaustive list of compound parts. Such lists can be extracted from monolingual data for which we use language model (LM) word frequency lists in combination with some filter steps. The extraction process is subdivided into 2 passes, one over a high-quality news LM to extract the parts and the other over a web LM to filter the parts.

4.2.1 Phase 1: Bootstrapping pass

In the first pass, we generate word frequency lists derived from news articles for multiple languages. The motivation for using news articles rather than arbitrary web texts is that news articles are in general less noisy and contain fewer spelling mistakes. The language-dependent word frequency lists are filtered according to a sequence of filter steps. These filter steps include discarding all words that contain digits or punctuations other than hyphen, minimum occurrence frequency, and a minimum length which we set to 4. The output is a table that contains preliminary compound parts together with their respective counts for each language.

4.2.2 Phase 2: Filtering pass

In the second pass, the compound part vocabulary is further reduced and filtered. We generate a LM vocabulary based on arbitrary web texts for each language and build a compound splitter based on the vocabulary list that was generated in phase 1. We now try to split every word of the web LM vocabulary based on the compound splitter model from phase 1. For the compound parts that occur in the compound splitter output, we determine how often each compound part was used and output only those compound parts whose frequency exceed a predefined threshold n.

4.3 Example

Suppose we have the following word frequencies output from pass 1:

floor	10k	poll	4k
flow	9k	pot	5k
flower	15k	potter	20k

In pass 2, we observe the word *flowerpot*. With the above list, the only compound parts used are *flower* and *pot*. If we did not split any other words and threshold at $n = 1$, our final list would consist of *flower* and *pot*. This filtering pass has the advantage of outputting only those compound part candidates

which were actually used to split words from web texts. The thresholding also further reduces the risk of introducing noise. Another advantage is that since the set of parts output in the first pass may contain a high number of compounds, the filter is able to remove a large number of these compounds by examining relative frequencies. In our experiments, we have assumed that compound part frequencies are higher than the compound frequency and so remove words from the part list that can themselves be split and have a relatively high frequency. Finally, after removing the low frequency compound parts, we obtain the final compound splitter vocabulary.

4.4 Generating Exception Lists

To avoid eager splitting of non-compounds and named entities, we use a variant of the procedure described in Section 4.1.1. By emitting all those source words that translate with high probability into single-token English words, we obtain a list of words that should not be split.[4]

4.5 Final Cost Function

The final cost function is defined by the following components which are combined log-linearly.

- The split penalty ξ penalizes each compound part to avoid eager splitting.
- The cost for each compound part g_k is computed as $-\log C(g_k)$, where $C(g_k)$ is the unigram count for g_k obtained from the news LM word frequency list. Since we use a zero-order model, we can ignore the normalization and work with unigram counts rather than unigram probabilities.
- Because Algorithm 1 iterates over the characters of the input token w, we can infer from the boundaries (i, j) if we are at the start, in the middle, or at the end of the token. Applying a morphological operation adds costs 1 to the overall costs.

Although the cost function is language dependent, we use the same split penalty weight $\xi = 20$ for all languages except for German, where the split penalty weight is set to 13.5.

5 Results

To show the language independence of the approach within a machine translation task, we translate from languages belonging to different language families into English. The publicly available Europarl corpus is not suitable for demonstrating the utility of compound splitting because there are few unseen compounds in the test section of the Europarl corpus. The WMT shared translation task has a broader domain compared to Europarl but covers only a few languages. Hence, we present results for German-English using the WMT-07 data and cover other languages using non-public corpora which contain news as well as open-domain web texts. Table 1 lists the various corpus statistics. The source languages are grouped according to their language family.

For learning the morphological operations, we allowed the substitution of at most 2 consecutive characters. Furthermore, we only allowed at most one morphological substitution to avoid introducing too much noise. The found morphological operations were sorted according to their frequencies. Those which occurred less than 100 times were discarded. Examples of extracted morphological operations are given in Table 2. Because the extraction procedure described in Section 4.1 is not purely restricted to the case of decompounding, we found that many morphological operations emitted by this procedure reflect morphological variations that are not directly linked to compounding, but caused by inflections.

To generate the language-dependent lists of compound parts, we used language model vocabulary lists[5] generated from news texts for different languages as seeds for the first pass. These lists were filtered by discarding all entries that either contained digits, punctuations other than hyphens, or sequences of the same characters. In addition, the infrequent entries were discarded as well to further reduce noise. For the second pass, we used the lists generated in the first pass together with the learned morphological operations to construct a preliminary compound splitter. We then generated vocabulary lists for monolingual web texts and applied the preliminary compound splitter onto this list. The used

[4] Because we will translate only into English, this is not an issue for the introductory example *flowerpot*.

[5] The vocabulary lists also contain the word frequencies. We use the term vocabulary list synonymously for a word frequency list.

Family	Src Language	#Tokens Train src/trg		#Tokens Dev src/trg		#Tokens Tst src/trg	
Germanic	Danish	196M	201M	43,475	44,479	72,275	74,504
	German	43M	45M	23,151	22,646	45,077	43,777
	Norwegian	251M	255M	42,096	43,824	70,257	73,556
	Swedish	201M	213M	42,365	44,559	70,666	74,547
Hellenic	Greek	153M	148M	47,576	44,658	79,501	74,776
Uralic	Estonian	199M	244M	34,987	44,658	57,916	74,765
	Finnish	205M	246M	32,119	44,658	53,365	74,771

Table 1: Corpus statistics for various language pairs. The target language is always English. The source languages are grouped according to their language family.

Language	morpholog. operations
Danish	-/ε, s/ε
German	-/ε, s/ε, es/ε, n/ε, e/ε, en/ε
Norwegian	-/ε, s/ε, e/ε
Swedish	-/ε, s/ε
Greek	ε/α, ε/ς, ε/η, o/ί, o/ί, o/ν
Estonian	-/ε, e/ε, se/ε
Finnish	ε/n, n/ε, ε/en

Table 2: Examples of morphological operations that were extracted from bilingual corpora.

compound parts were collected and sorted according to their frequencies. Those which were used at least 2 times were kept in the final compound parts lists. Table 3 reports the number of compound parts kept after each pass. For example, the Finnish news vocabulary list initially contained 1.7M entries. After removing non-alpha and infrequent words in the first filter step, we obtained 190K entries. Using the preliminary compound splitter in the second filter step resulted in 73K compound part entries.

The finally obtained compound splitter was integrated into the preprocessing pipeline of a state-of-the-art statistical phrase-based machine translation system that works similar to the Moses decoder (Koehn et al., 2007). By applying the compound splitter during both training and decoding we ensured that source language tokens were split in the same way. Table 4 presents results for various language-pairs with and without decompounding. Both the Germanic and the Uralic languages show significant BLEU score improvements of 1.3 BLEU points on average. The confidence intervals were computed using the bootstrap resampling normal approximation method described in (Noreen, 1989). While the compounding process for Germanic languages is rather simple and requires only a few linking morphemes, compounds used in Uralic languages have a richer morphology. In contrast to the Germanic and Uralic languages, we did not observe improvements for Greek. To investigate this lack of performance, we turned off transliteration and kept unknown source words in their original script. We analyzed the number of remaining source characters in the baseline system and the system using compound splitting by counting the number of Greek characters in the translation output. The number of remaining Greek characters in the translation output was reduced from $6,715$ in the baseline system to $3,624$ in the system which used decompounding. In addition, a few other metrics like the number of source words that consisted of more than 15 characters decreased as well. Because we do not know how many compounds are actually contained in the Greek source sentences[6] and because the frequency of using compounds might vary across languages, we cannot expect the same performance gains across languages belonging to different language families. An interesting observation is, however, that if one language from a language family shows performance gains, then there are performance gains for all the languages in that family. We also investigated the effect of not using any morphological operations. Disallowing all morphological operations accounts for a loss of 0.1 - 0.2 BLEU points across translation systems and increases the compound parts vocabulary lists by up to 20%, which means that most of the gains can be achieved with simple concatenation.

The exception lists were generated according to the procedure described in Section 4.4. Since we aimed for precision rather than recall when constructing these lists, we inserted only those source

[6]Quite a few of the remaining Greek characters belong to rare named entities.

Language	initial vocab size	#parts after 1st pass	#parts after 2nd pass
Danish	918,708	132,247	49,592
German	7,908,927	247,606	45,059
Norwegian	1,417,129	237,099	62,107
Swedish	1,907,632	284,660	82,120
Greek	877,313	136,436	33,130
Estonian	742,185	81,132	36,629
Finnish	1,704,415	190,507	73,568

Table 3: Number of remaining compound parts for various languages after the first and second filter step.

System	BLEU[%] w/o splitting	BLEU[%] w splitting	Δ	CI 95%
Danish	42.55	44.39	1.84	($\pm$ 0.65)
German WMT-07	25.76	26.60	0.84	($\pm$ 0.70)
Norwegian	42.77	44.58	1.81	($\pm$ 0.64)
Swedish	36.28	38.04	1.76	($\pm$ 0.62)
Greek	31.85	31.91	0.06	($\pm$ 0.61)
Estonian	20.52	21.20	0.68	($\pm$ 0.50)
Finnish	25.24	26.64	1.40	($\pm$ 0.57)

Table 4: BLEU score results for various languages translated into English with and without compound splitting.

Language	Split	source	translation
German	no	Die EU ist nicht einfach ein Freundschaftsclub.	The EU is not just a Freundschaftsclub.
	yes	Die EU ist nicht einfach ein Freundschaft Club	The EU is not simply a friendship club.
Greek	no	Τι είναι παλμοκωδική διαμόρφωση;	What παλμοκωδική configuration?
	yes	Τι είναι παλμο κωδικη διαμόρφωση;	What is pulse code modulation?
Finnish	no	Lisävuodevaatteet ja pyyheliinat ovat kaapissa.	Lisävuodevaatteet and towels are in the closet.
	yes	Lisä Vuode Vaatteet ja pyyheliinat ovat kaapissa.	Extra bed linen and towels are in the closet.

Table 5: Examples of translations into English with and without compound splitting.

words whose co-occurrence count with a unigram translation was at least $1,000$ and whose translation probability was larger than 0.1. Furthermore, we required that at least 70% of all target phrase entries for a given source word had to be unigrams. All decompounding results reported in Table 4 were generated using these exception lists, which prevented wrong splits caused by otherwise overly eager splitting.

6 Conclusion and Outlook

We have presented a language-independent method for decompounding that improves translations for compounds that otherwise rarely occur in the bilingual training data. We learned a set of morphological operations from a translation phrase table and determined suitable compound part candidates from monolingual data in a two pass process. This allowed us to learn morphemes and operations for languages where these lists are not available. In addition, we have used the bilingual information stored in the phrase table to avoid splitting non-compounds as well as frequent named entities. All knowledge sources were combined in a cost function that was applied in a compound splitter based on dynamic programming. Finally, we have shown this improves translation performance on languages from different language families.

The weights were not optimized in a systematic way but set manually to their respective values. In the future, the weights of the cost function should be learned automatically by optimizing an appropriate error function. Instead of using gold data, the development data for optimizing the error function could be collected without supervision using the methods proposed in this paper.

References

Enrique Alfonseca, Slaven Bilac, and Stefan Paries. 2008a. Decompounding query keywords from compounding languages. In *Proc. of the 46th Annual Meeting of the Association for Computational Linguistics (ACL): Human Language Technologies (HLT)*, pages 253--256, Columbus, Ohio, USA, June.

Enrique Alfonseca, Slaven Bilac, and Stefan Paries. 2008b. German decompounding in a difficult corpus. In A. Gelbukh, editor, *Lecture Notes in Computer Science (LNCS): Proc. of the 9th Int. Conf. on Intelligent Text Processing and Computational Linguistics (CICLING)*, volume 4919, pages 128--139. Springer Verlag, February.

Ralf D. Brown. 2002. Corpus-Driven Splitting of Compound Words. In *Proc. of the 9th Int. Conf. on Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 12--21, Keihanna, Japan, March.

Chris Dyer. 2009. Using a maximum entropy model to build segmentation lattices for mt. In *Proc. of the Human Language Technologies (HLT): The Annual Conf. of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 406--414, Boulder, Colorado, June.

Nikesh Garera and David Yarowsky. 2008. Translating Compounds by Learning Component Gloss Translation Models via Multiple Languages. In *Proc. of the 3rd Internation Conference on Natural Language Processing (IJCNLP)*, pages 403--410, Hyderabad, India, January.

Philipp Koehn and Kevin Knight. 2003. Empirical methods for compound splitting. In *Proc. of the 10th Conf. of the European Chapter of the Association for Computational Linguistics (EACL)*, volume 1, pages 187--193, Budapest, Hungary, April.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proc. of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)*, volume 1, pages 177--180, Prague, Czech Republic, June.

Eric W. Noreen. 1989. *Computer-Intensive Methods for Testing Hypotheses*. John Wiley & Sons, Canada.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311--318, Philadelphia, Pennsylvania, July.