Lattice Desegmentation for Statistical Machine Translation

Mohammad Salameh[†] **Colin Cherry**[‡] **Grzegorz Kondrak**[†]

[†]Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8, Canada
{msalameh,gkondrak}@ualberta.ca

[‡]National Research Council Canada
1200 Montreal Road
Ottawa, ON, K1A 0R6, Canada
Colin.Cherry@nrc-cnrc.gc.ca

Abstract

Morphological segmentation is an effective sparsity reduction strategy for statistical machine translation (SMT) involving morphologically complex languages. When translating into a segmented language, an extra step is required to desegment the output; previous studies have desegmented the 1-best output from the decoder. In this paper, we expand our translation options by desegmenting n-best lists or lattices. Our novel lattice desegmentation algorithm effectively combines both segmented and desegmented views of the target language for a large subspace of possible translation outputs, which allows for inclusion of features related to the desegmentation process, as well as an unsegmented language model (LM). We investigate this technique in the context of English-to-Arabic and English-to-Finnish translation, showing significant improvements in translation quality over desegmentation of 1-best decoder outputs.

1 Introduction

Morphological segmentation is considered to be indispensable when translating between English and morphologically complex languages such as Arabic. Morphological complexity leads to much higher type to token ratios than English, which can create sparsity problems during translation model estimation. Morphological segmentation addresses this issue by splitting surface forms into meaningful morphemes, while also performing orthographic transformations to further reduce sparsity. For example, the Arabic noun للدول *lldwl* "to the countries" is segmented as *l+* "to" *Aldwl* "the countries". When translating from Arabic, this segmentation process is performed as input preprocessing and is otherwise transparent to the translation system. However, when translating into Arabic, the decoder produces segmented output, which must be **desegmented** to produce readable text. For example, *l+ Aldwl* must be converted to *lldwl*.

Desegmentation is typically performed as a post-processing step that is independent from the decoding process. While this division of labor is useful, the pipeline approach may prevent the desegmenter from recovering from errors made by the decoder. Despite the efforts of the decoder's various component models, the system may produce mismatching segments, such as *s+ hzymp*, which pairs the future particle *s+* "will" with a noun *hzymp* "defeat", instead of a verb. In this scenario, there is no right desegmentation; the post-processor has been dealt a losing hand.

In this work, we show that it is possible to maintain the sparsity-reducing benefit of segmentation while translating directly into unsegmented text. We desegment a large set of possible decoder outputs by processing n-best lists or lattices, which allows us to consider both the segmented and desegmented output before locking in the decoder's decision. We demonstrate that significant improvements in translation quality can be achieved by training a linear model to re-rank this transformed translation space.

2 Related Work

Translating into morphologically complex languages is a challenging and interesting task that has received much recent attention. Most techniques approach the problem by transforming the target language in some manner before training the translation model. They differ in what transformations are performed and at what stage they are reversed. The transformation might take the form of a morphological analysis or a morphological segmentation.

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pages 100–110,
Baltimore, Maryland, USA, June 23-25 2014.

2.1 Morphological Analysis

Many languages have access to morphological analyzers, which annotate surface forms with their lemmas and morphological features. Bojar (2007) incorporates such analyses into a factored model, to either include a language model over target morphological tags, or model the generation of morphological features. Other approaches train an SMT system to predict lemmas instead of surface forms, and then inflect the SMT output as a post-processing step (Minkov et al., 2007; Clifton and Sarkar, 2011; Fraser et al., 2012; El Kholy and Habash, 2012b). Alternatively, one can reparameterize existing phrase tables as exponential models, so that translation probabilities account for source context and morphological features (Jeong et al., 2010; Subotin, 2011). Of these approaches, ours is most similar to the translate-then-inflect approach, except we translate and then desegment. In particular, Toutanova et al. (2008) inflect and re-rank n-best lists in a similar manner to how we desegment and re-rank n-best lists or lattices.

2.2 Morphological Segmentation

Instead of producing an abstract feature layer, morphological segmentation transforms the target sentence by segmenting relevant morphemes, which are then handled as regular tokens during alignment and translation. This is done to reduce sparsity and to improve correspondence with the source language (usually English). Such a segmentation can be produced as a byproduct of analysis (Oflazer and Durgar El-Kahlout, 2007; Badr et al., 2008; El Kholy and Habash, 2012a), or may be produced using an unsupervised morphological segmenter such as Morfessor (Luong et al., 2010; Clifton and Sarkar, 2011). Work on target language morphological segmentation for SMT can be divided into three subproblems: segmentation, desegmentation and integration. Our work is concerned primarily with the integration problem, but we will discuss each subproblem in turn.

The usefulness of a target segmentation depends on its correspondence to the source language. If a morphological feature does not manifest itself as a separate token in the source, then it may be best to leave its corresponding segment attached to the stem. A number of studies have looked into what granularity of segmentation is best suited for a particular language pair (Oflazer and Durgar El-Kahlout, 2007; Badr et al., 2008; Clifton and Sarkar, 2011; El Kholy and Habash, 2012a). Since our focus here is on integrating segmentation into the decoding process, we simply adopt the segmentation strategies recommended by previous work: the Penn Arabic Treebank scheme for English-Arabic (El Kholy and Habash, 2012a), and an unsupervised scheme for English-Finnish (Clifton and Sarkar, 2011).

Desegmentation is the process of converting segmented words into their original surface form. For many segmentations, especially unsupervised ones, this amounts to simple concatenation. However, more complex segmentations, such as the Arabic tokenization provided by MADA (Habash et al., 2009), require further orthographic adjustments to reverse normalizations performed during segmentation. Badr et al. (2008) present two Arabic desegmentation schemes: table-based and rule-based. El Kholy and Habash (2012a) provide an extensive study on the influence of segmentation and desegmentation on English-to-Arabic SMT. They introduce an additional desegmentation technique that augments the table-based approach with an unsegmented language model. Salameh et al. (2013) replace rule-based desegmentation with a discriminatively-trained character transducer. In this work, we adopt the Table+Rules approach of El Kholy and Habash (2012a) for English-Arabic, while concatenation is sufficient for English-Finnish.

Work on integration attempts to improve SMT performance for morphologically complex target languages by going beyond simple pre- and post-processing. Oflazer and Durgar El-Kahlout (2007) desegment 1000-best lists for English-to-Turkish translation to enable scoring with an unsegmented language model. Unlike our work, they *replace* the segmented language model with the unsegmented one, allowing them to tune the linear model parameters by hand. We use both segmented and unsegmented language models, and tune automatically to optimize BLEU.

Like us, Luong et al. (2010) tune on unsegmented references,[1] and translate with both segmented and unsegmented language models for English-to-Finnish translation. However, they adopt a scheme of word-boundary-aware

[1]Tuning on unsegmented references does not require substantial modifications to the standard SMT pipeline. For example, Badr et al. (2008) also tune on unsegmented references by simply desegmenting SMT output before MERT collects sufficient statistics for BLEU.

morpheme-level phrase extraction, meaning that target phrases include only complete words, though those words are segmented into morphemes. This enables full decoder integration, where we do n-best and lattice re-ranking. But it also comes at a substantial cost: when target phrases include only complete words, the system can only generate word forms that were seen during training. In this setting, the sparsity reduction from segmentation helps word alignment and target language modeling, but it does not result in a more expressive translation model. Furthermore, it becomes substantially more difficult to have non-adjacent source tokens contribute morphemes to a single target word. For example, when translating "with his blue car" into the Arabic بسيارته الزرقاء *bsyArth AlzrqA'*, the target word *bsyArth* is composed of three tokens: *b+* "with", *syArp* "car" and *+h* "his". With word-boundary-aware phrase extraction, a phrase pair containing all of "with his blue car" must have been seen in the parallel data to translate the phrase correctly at test time. With lattice desegmentation, we need only to have seen *AlzrqA'* "blue" and the three morphological pieces of *bsyArth* for the decoder and desegmenter to assemble the phrase.

3 Methods

Our goal in this work is to benefit from the sparsity-reducing properties of morphological segmentation while simultaneously allowing the system to reason about the final surface forms of the target language. We approach this problem by augmenting an SMT system built over target segments with features that reflect the desegmented target words. In this section, we describe our various strategies for desegmenting the SMT system's output space, along with the features that we add to take advantage of this desegmented view.

3.1 Baselines

The two obvious baseline approaches each decode using one view of the target language. The **unsegmented** approach translates without segmenting the target. This trivially allows for an unsegmented language model and never makes desegmentation errors. However, it suffers from data sparsity and poor token-to-token correspondence with the source language.

The **one-best desegmentation** approach segments the target language at training time and then desegments the one-best output in post-processing. This resolves the sparsity issue, but does not allow the decoder to take into account features of the desegmented target. To the best of our knowledge, we are the first group to go beyond one-best desegmentation for English-to-Arabic translation. In English-to-Finnish, although alternative integration strategies have seen some success (Luong et al., 2010), the current state-of-the-art performs one-best-desegmentation (Clifton and Sarkar, 2011).

3.2 n-best Desegmentation

The one-best approach can be extended easily by desegmenting n-best lists of segmented decoder output. Doing so enables the inclusion of an unsegmented target language model, and with a small amount of bookkeeping, it also allows the inclusion of features related to the operations performed during desegmentation (see Section 3.4). With new features reflecting the desegmented output, we can re-tune our enhanced linear model on a development set. Following previous work, we will desegment 1000-best lists (Oflazer and Durgar El-Kahlout, 2007).

Once n-best lists have been desegmented, we can tune on unsegmented references as a side-benefit. This could improve translation quality, as it brings our training scenario closer to our test scenario (test BLEU is always measured on unsegmented references). In particular, it could address issues with translation length mismatch. Previous work that has tuned on unsegmented references has reported mixed results (Badr et al., 2008; Luong et al., 2010).

3.3 Lattice Desegmentation

An n-best list reflects a tiny portion of a decoder's search space, typically fixed at 1000 hypotheses. Lattices[2] can represent an exponential number of hypotheses in a compact structure. In this section, we discuss how a lattice from a multi-stack phrase-based decoder such as Moses (Koehn et al., 2007) can be desegmented to enable word-level features.

Finite State Analogy

A phrase-based decoder produces its output from left to right, with each operation appending the translation of a source phrase to a growing target hypothesis. Translation continues un-

[2]Or forests for hierarchical and syntactic decoders.

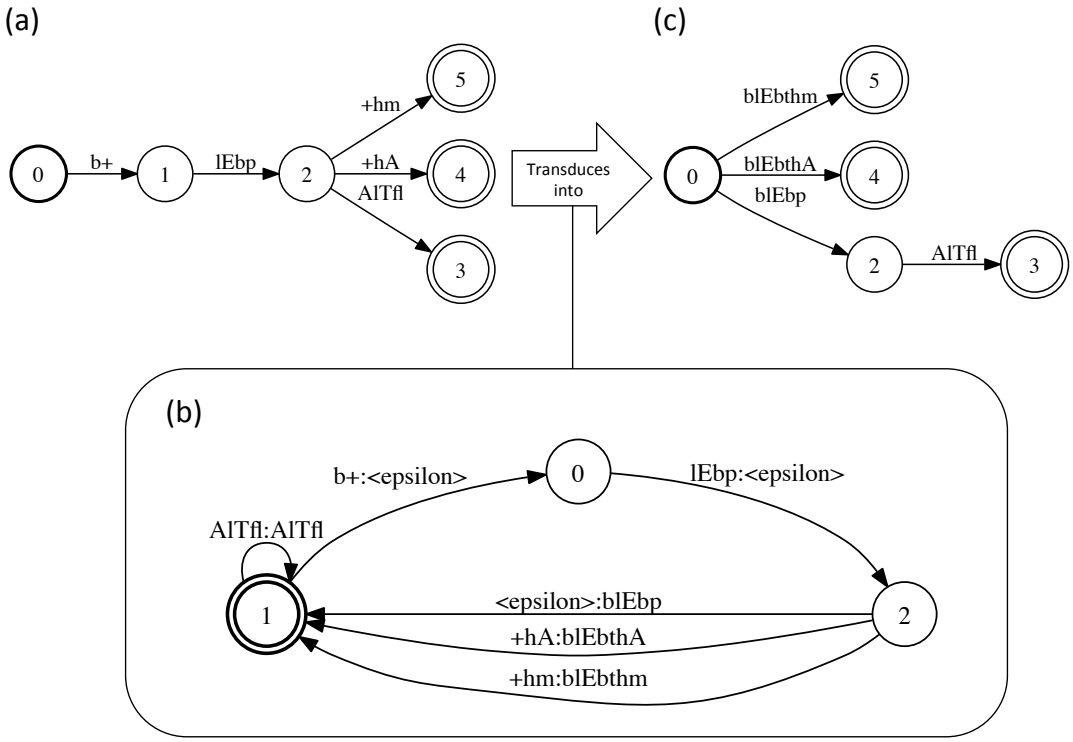

Figure 1: The finite state pipeline for a lattice translating the English fragment "with the child's game". The input morpheme lattice (a) is desegmented by composing it with the desegmenting transducer (b) to produce the word lattice (c). The tokens in (a) are: *b+* "with", *lEbp* "game", *+hm* "their", *+hA* "her", and *AlTfl* "the child".

til each source word has been covered exactly once (Koehn et al., 2003).

The search graph of a phrase-based decoder can be interpreted as a lattice, which can be interpreted as a finite state acceptor over target strings. In its most natural form, such an acceptor emits target phrases on each edge, but it can easily be transformed into a form with one edge per token, as shown in Figure 1a. This is sometimes referred to as a word graph (Ueffing et al., 2002), although in our case the segmented phrase table also produces tokens that correspond to morphemes.

Our goal is to desegment the decoder's output lattice, and in doing so, gain access to a compact, desegmented view of a large portion of the translation search space. This can be accomplished by composing the lattice with a **desegmenting transducer** that consumes morphemes and outputs desegmented words. This transducer must be able to consume every word in our lattice's output vocabulary. We define a word using the following regular expression:

$$\text{[prefix]* [stem] [suffix]* | [prefix]+ [suffix]+} \quad (1)$$

where [prefix], [stem] and [suffix] are non-overlapping sets of morphemes, whose members are easily determined using the segmenter's segment boundary markers.[3] The second disjunct of Equation 1 covers words that have no clear stem, such as the Arabic لـ *lh* "for him", segmented as *l+* "for" *+h* "him". Equation 1 may need to be modified for other languages or segmentation schemes, but our techniques generalize to any definition that can be written as a regular expression.

A desegmenting transducer can be constructed by first encoding our desegmenter as a table that maps morpheme sequences to words. Regardless of whether the original desegmenter was based on concatenation, rules or table-lookup, it can be encoded as a lattice-specific table by applying it to an enumeration of all words found in the lattice. We can then transform that table into a finite state transducer with one path per table entry. Finally, we take the closure of this transducer, so that the resulting machine can transduce any sequence of words. The desegmenting trans-

[3] Throughout this paper, we use "+" to mark morphemes as prefixes or suffixes, as in *w+* or *+h*. In Equation 1 only, we overload "+" as the Kleene cross: $X+ == XX*$.

ducer for our running example is shown in Figure 1b. Note that tokens requiring no desegmentation simply emit themselves. The lattice (Figure 1a) can then be desegmented by composing it with the transducer (1b), producing a desegmented lattice (1c). This is a natural place to introduce features that describe the desegmentation process, such as scores provided by a desegmentation table, which can be incorporated into the desegmenting transducer's edge weights.

We now have a desegmented lattice, but it has not been annotated with an unsegmented (word-level) language model. In order to annotate lattice edges with an n-gram LM, every path coming into a node must end with the same sequence of $(n-1)$ tokens. If this property does not hold, then nodes must be split until it does.[4] This property is maintained by the decoder's recombination rules for the segmented LM, but it is not guaranteed for the desegmented LM. Indeed, the expanded word-level context is one of the main benefits of incorporating a word-level LM. Fortunately, LM annotation as well as any necessary lattice modifications can be performed simultaneously by composing the desegmented lattice with a finite state acceptor encoding the LM (Roark et al., 2011).

In summary, we are given a segmented lattice, which encodes the decoder's translation space as an acceptor over morphemes. We compose this acceptor with a desegmenting transducer, and then with an unsegmented LM acceptor, producing a fully annotated, desegmented lattice. Instead of using a tool kit such as OpenFst (Allauzen et al., 2007), we implement both the desegmenting transducer and the LM acceptor programmatically. This eliminates the need to construct intermediate machines, such as the lattice-specific desegmenter in Figure 1b, and facilitates working with edges annotated with feature vectors as opposed to single weights.

Programmatic Desegmentation

Lattice desegmentation is a non-local lattice transformation. That is, the morphemes forming a word might span several edges, making desegmentation non-trivial. Luong et al. (2010) address this problem by forcing the decoder's phrase table to respect word boundaries, guaranteeing that each desegmentable token sequence is local to an edge. Inspired by the use of non-local features in forest decoding (Huang, 2008), we present an algorithm to find **chains** of edges that correspond to desegmentable token sequences, allowing lattice desegmentation with no phrase-table restrictions. This algorithm can be seen as implicitly constructing a customized desegmenting transducer and composing it with the input lattice on the fly.

Before describing the algorithm, we define some notation. An input morpheme lattice is a triple $\langle n_s, \mathcal{N}, \mathcal{E} \rangle$, where $\mathcal{N}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $n_s \in \mathcal{N}$ is the start node that begins each path through the lattice. Each edge $e \in \mathcal{E}$ is a 4-tuple $\langle \mathit{from}, \mathit{to}, \mathit{lex}, w \rangle$, where from, $\mathit{to} \in \mathcal{N}$ are head and tail nodes, lex is a single token accepted by this edge, and w is the (potentially vector-valued) edge weight. Tokens are drawn from one of three non-overlapping morpho-syntactic sets: $\mathit{lex} \in \mathit{Prefix} \cup \mathit{Stem} \cup \mathit{Suffix}$, where tokens that do not require desegmentation, such as complete words, punctuation and numbers, are considered to be in Stem. It is also useful to consider the set of all outgoing edges for a node $n.\mathit{out} = \{e \in \mathcal{E} | e.\mathit{from} = n\}$.

With this notation in place, we can define a **chain** c to be a sequence of edges $[e_1 \dots e_l]$ such that for $1 \leq i < l : e_i.\mathit{to} = e_{i+1}.\mathit{from}$. We denote singleton chains with $[e]$, and when unambiguous, we abbreviate longer chains with their start and end node $[e_1.\mathit{from} \rightarrow e_l.\mathit{to}]$. A chain is **valid** if it emits the beginning of a word as defined by the regular expression in Equation 1. A valid chain is **complete** if its edges form an entire word, and if it is part of a path through the lattice that consists only of words. In Figure 1a, the complete chains are $[0 \rightarrow 2]$, $[0 \rightarrow 4]$, $[0 \rightarrow 5]$, and $[2 \rightarrow 3]$. The path restriction on complete chains forces words to be bounded by other words in order to be complete.[5] For example, if we removed the edge $2 \rightarrow 3$ (*AlTfl*) from Figure 1a, then $[0 \rightarrow 2]$ ([*b+ lEbp*]) would cease to be a complete chain, but it would still be a valid chain. Note that in the finite-state analogy, the path restriction is implicit in the composition operation.

Algorithm 1 desegments a lattice by finding all complete chains and replacing each one with a single edge. It maintains a work list of nodes that lie on the boundary between words, and for each node on this list, it launches a depth first search

[4] Or the LM composition can be done dynamically, effectively decoding the lattice with a beam or cube-pruned search (Huang and Chiang, 2007).

[5] Sentence-initial suffix morphemes and sentence-final prefix morphemes represent a special case that we omit for the sake of brevity. Lacking stems, they are left segmented.

Algorithm 1 Desegment a lattice $\langle n_s, \mathcal{N}, \mathcal{E} \rangle$

```
{Initialize output lattice and work list WL}
n'_s = n_s, N' = ∅, E' = ∅, WL = [n_s]
while n = WL.pop() do
  {Work on each node only once}
  if n ∈ N' then continue
  N' = N' ∪ {n}
  {Initialize the chain stack C}
  C = ∅
  for e ∈ n.out do
    if [e] is valid then C.push([e])
  {Depth-first search for complete chains}
  while [e_1, ..., e_l] = C.pop() do
    {Attempt to extend chain}
    for e ∈ e_l.to.out do
      if [e_1 ... e_l, e] is valid then
        C.push([e_1, ..., e_l, e])
      else
        Mark [e_1, ..., e_l] as complete
    {Desegment complete chains}
    if [e_1, ..., e_l] is complete then
      WL.push(e_l.to)
      E' = E' ∪ {deseg([e_1, ..., e_l])}
return ⟨n'_s, N', E'⟩
```

to find all complete chains extending from it. The search recognizes the valid chain c to be complete by finding an edge e such that $c + e$ forms a chain, but not a valid one. By inspection of Equation 1, this can only happen when a prefix or stem follows a stem or suffix, which always marks a word boundary. The chains found by this search are desegmented and then added to the output lattice as edges. The nodes at end points of these chains are added to the work list, as they lie at word boundaries by definition. Note that although this algorithm creates completely new edges, the resulting node set $\mathcal{N}'$ will be a subset of the input node set $\mathcal{N}$. The complement $\mathcal{N} - \mathcal{N}'$ will consist of nodes that are word-internal in all paths through the input lattice, such as node 1 in Figure 1a.

Programmatic LM Integration

Programmatic composition of a lattice with an n-gram LM acceptor is a well understood problem. We use a dynamic program to enumerate all $(n-1)$-word contexts leading into a node, and then split the node into multiple copies, one for each context. With each node corresponding to a single LM context, annotation of outgoing edges with n-gram LM scores is straightforward.

3.4 Desegmentation Features

Our re-ranker has access to all of the features used by the decoder, in addition to a number of features enabled by desegmentation.

Desegmentation Score We use a table-based desegmentation method for Arabic, which is based on segmenting an Arabic training corpus and memorizing the observed transformations to reverse them later. Finnish does not require a table, as all words can be desegmented with simple concatenation. The Arabic table consists of $X \rightarrow Y$ entries, where X is a target morpheme sequence and Y is a desegmented surface form. Several entries may share the same X, resulting in multiple desegmentation options. For the sake of symmetry with the unambiguous Finnish case, we augment Arabic n-best lists or lattices with only the most frequent desegmentation Y.[6] We provide the desegmentation score $\log p(Y|X)$= $\log\left(\frac{\text{count of } X \rightarrow Y}{\text{count of X}}\right)$ as a feature, to indicate the entry's ambiguity in the training data.[7] When an X is missing from the table, we fall back on a set of desegmentation rules (El Kholy and Habash, 2012a) and this feature is set to 0. This feature is always 0 for English-Finnish.

Contiguity One advantage of our approach is that it allows discontiguous source words to translate into a single target word. In order to maintain some control over this powerful capability, we create three binary features that indicate the contiguity of a desegmentation. The first feature indicates that the desegmented morphemes were translated from contiguous source words. The second indicates that the source words contained a single discontiguity, as in a word-by-word translation of the "with his blue car" example from Section 2.2. The third indicates two or more discontiguities.

Unsegmented LM A 5-gram LM trained on unsegmented target text is used to assess the fluency of the desegmented word sequence.

4 Experimental Setup

We train our English-to-Arabic system using 1.49 million sentence pairs drawn from the NIST 2012 training set, excluding the UN data. This training set contains about 40 million Arabic tokens before

[6] Allowing the re-ranker to choose between multiple Ys is a natural avenue for future work.

[7] We also experimented on $\log p(X|Y)$ as an additional feature, but observed no improvement in translation quality.

segmentation, and 47 million after segmentation. We tune on the NIST 2004 evaluation set (1353 sentences) and evaluate on NIST 2005 (1056 sentences). As these evaluation sets are intended for Arabic-to-English translation, we select the first English reference to use as our source text.

Our English-to-Finnish system is trained on the same Europarl corpus as Luong et al. (2010) and Clifton and Sarkar (2011), which has roughly one million sentence pairs. We also use their development and test sets (2000 sentences each).

4.1 Segmentation

For Arabic, morphological segmentation is performed by MADA 3.2 (Habash et al., 2009), using the Penn Arabic Treebank (PATB) segmentation scheme as recommended by El Kholy and Habash (2012a). For both segmented and unsegmented Arabic, we further normalize the script by converting different forms of Alif ا آ أ إ and Ya ى ي to bare Alif ا and dotless Ya ى. To generate the desegmentation table, we analyze the segmentations from the Arabic side of the parallel training data to collect mappings from morpheme sequences to surface forms.

For Finnish, we adopt the Unsup L-match segmentation technique of Clifton and Sarkar (2011), which uses Morfessor (Creutz and Lagus, 2005) to analyze the 5,000 most frequent Finnish words. The analysis is then applied to the Finnish side of the parallel text, and a list of segmented suffixes is collected. To improve coverage, words are further segmented according to their longest matching suffix from the list. As Morfessor does not perform any orthographic normalizations, it can be desegmented with simple concatenation.

4.2 Systems

We align the parallel data with GIZA++ (Och et al., 2003) and decode using Moses (Koehn et al., 2007). The decoder's log-linear model includes a standard feature set. Four translation model features encode phrase translation probabilities and lexical scores in both directions. Seven distortion features encode a standard distortion penalty as well as a bidirectional lexicalized reordering model. A KN-smoothed 5-gram language model is trained on the target side of the parallel data with SRILM (Stolcke, 2002). Finally, we include word and phrase penalties. The decoder uses the default parameters for English-to-Arabic, except that the maximum phrase length is set to 8. For English-to-Finnish, we follow Clifton and Sarkar (2011) in setting the hypothesis stack size to 100, distortion limit to 6, and maximum phrase length to 20.

The decoder's log-linear model is tuned with MERT (Och, 2003). Re-ranking models are tuned using a batch variant of hope-fear MIRA (Chiang et al., 2008; Cherry and Foster, 2012), using the n-best variant for n-best desegmentation, and the lattice variant for lattice desegmentation. MIRA was selected over MERT because we have an in-house implementation that can tune on lattices very quickly. During development, we confirmed that MERT and MIRA perform similarly, as is expected with fewer than 20 features. Both the decoder's log-linear model and the re-ranking models are trained on the same development set. Historically, we have not seen improvements from using different tuning sets for decoding and re-ranking. Lattices are pruned to a density of 50 edges per word before re-ranking.

We test four different systems. Our first baseline is **Unsegmented**, where we train on unsegmented target text, requiring no desegmentation step. Our second baseline is **1-best Deseg**, where we train on segmented target text and desegment the decoder's 1-best output. Starting from the system that produced 1-best Deseg, we then output either 1000-best lists or lattices to create our two experimental systems. The **1000-best Deseg** system desegments, augments and re-ranks the decoder's 1000-best list, while **Lattice Deseg** does the same in the lattice. We augment n-best lists and lattices using the features described in Section 3.4.[8]

We evaluate our system using BLEU (Papineni et al., 2002) and TER (Snover et al., 2006). Following Clark et al. (2011), we report average scores over five random tuning replications to account for optimizer instability. For the baselines, this means 5 runs of decoder tuning. For the desegmenting re-rankers, this means 5 runs of re-ranker tuning, each working on n-best lists or lattices produced by the same (representative) decoder weights. We measure statistical significance using MultEval (Clark et al., 2011), which implements a stratified approximate randomization test to account for multiple tuning replications.

[8] Development experiments on a small-data English-to-Arabic scenario indicated that the Desegmentation Score was not particularly useful, so we exclude it from the main comparison, but include it in the ablation experiments.

5 Results

Tables 1 and 2 report results averaged over 5 tuning replications on English-to-Arabic and English-to-Finnish, respectively. In all scenarios, both 1000-best Deseg and Lattice Deseg significantly outperform the 1-best Deseg baseline ($p < 0.01$).

For English-to-Arabic, 1-best desegmentation results in a 0.7 BLEU point improvement over training on unsegmented Arabic. Moving to lattice desegmentation more than doubles that improvement, resulting in a BLEU score of 34.4 and an improvement of 1.0 BLEU point over 1-best desegmentation. 1000-best desegmentation also works well, resulting in a 0.6 BLEU point improvement over 1-best. Lattice desegmentation is significantly better ($p < 0.01$) than 1000-best desegmentation.

For English-to-Finnish, the Unsup L-match segmentation with 1-best desegmentation does not improve over the unsegmented baseline. The segmentation may be addressing issues with model sparsity, but it is also introducing errors that would have been impossible had words been left unsegmented. In fact, even with our lattice desegmenter providing a boost, we are unable to see a significant improvement over the unsegmented model. As we attempted to replicate the approach of Clifton and Sarkar (2011) exactly by working with their segmented data, this difference is likely due to changes in Moses since the publication of their result. Nonetheless, the 1000-best and lattice desegmenters both produce significant improvements over the 1-best desegmentation baseline, with Lattice Deseg achieving a 1-point improvement in TER. These results match the established state-of-the-art on this data set, but also indicate that there is still room for improvement in identifying the best segmentation strategy for English-to-Finnish translation.

We also tried a similar Morfessor-based segmentation for Arabic, which has an unsegmented test set BLEU of 32.7. As in Finnish, the 1-best desegmentation using Morfessor did not surpass the unsegmented baseline, producing a test BLEU of only 31.4 (not shown in Table 1). Lattice desegmentation was able to boost this to 32.9, slightly above 1-best desegmentation, but well below our best MADA desegmentation result of 34.4. There appears to be a large advantage to using MADA's supervised segmentation in this scenario.

Model	Dev	Test	
	BLEU	BLEU	TER
Unsegmented	24.4	32.7	49.4
1-best Deseg	24.4	33.4	48.6
1000-best Deseg	25.0	34.0	48.0
Lattice Deseg	25.2	34.4	47.7

Table 1: Results for English-to-Arabic translation using MADA's PATB segmentation.

Model	Dev	Test	
	BLEU	BLEU	TER
Unsegmented	15.4	15.1	70.8
1-best Deseg	15.3	14.8	71.9
1000-best Deseg	15.4	15.1	71.5
Lattice Deseg	15.5	15.1	70.9

Table 2: Results for English-to-Finnish translation using unsupervised segmentation.

5.1 Ablation

We conducted an ablation experiment on English-to-Arabic to measure the impact of the various features described in Section 3.4. Table 3 compares different combinations of features using lattice desegmentation. The unsegmented LM alone yields a 0.4 point improvement over the 1-best desegmentation score. Adding contiguity indicators on top of the unsegmented LM results in another 0.6 point improvement. As anticipated, the tuner assigns negative weights to discontiguous cases, encouraging the re-ranker to select a safer translation path when possible. Judging from the output on the NIST 2005 test set, the system uses these discontiguous desegmentations very rarely: only 5% of desegmented tokens align to discontiguous source phrases. Adding the desegmentation score to these two feature groups does not improve performance, confirming the results we observed during development. The desegmentation score would likely be useful in a scenario where we provide multiple desegmentation options to the re-ranker; for now, it indicates only the ambiguity of a fixed choice, and is likely redundant with information provided by the language model.

5.2 Error Analysis

In order to better understand the source of our improvements in the English-to-Arabic scenario, we conducted an extensive manual analysis of the differences between 1-best and lattice deseg-

Features	dev	test
1-best Deseg	24.5	33.4
+ Unsegmented LM	24.9	33.8
+ Contiguity	25.2	34.4
+ Desegmentation Score	25.2	34.3

Table 3: The effect of feature ablation on BLEU score for English-to-Arabic translation with lattice desegmentation.

	Lattice Correct	1-best Correct	Both Incorrect
Clitical	157	71	79
Lexical	61	39	80
Inflectional	37	32	47
Part-of-speech	19	17	11

Table 4: Error analysis for English-to-Arabic translation based on 650 sampled instances.

mentation on our test set. We compared the output of the two systems using the Unix tool *wdiff*, which transforms a solution to the longest-common-subsequence problem into a sequence of multi-word insertions and deletions (Hunt and McIlroy, 1976). We considered adjacent insertion-deletion pairs to be (potentially phrasal) substitutions, and collected them into a file, omitting any unpaired insertions or deletions. We then sampled 650 cases where the two sides of the substitution were deemed to be related, and divided these cases into categories based on how the lattice desegmentation differs from the one-best desegmentation. We consider a phrase to be correct only if it can be found in the reference.

Table 4 breaks down per-phrase accuracy according to four manually-assigned categories: (1) **clitical** – the two systems agree on a stem, but at least one clitic, often a prefix denoting a preposition or determiner, was dropped, added or replaced; (2) **lexical** – a word was changed to a morphologically unrelated word with a similar meaning; (3) **inflectional** – the words have the same stem, but different inflection due to a change in gender, number or verb tense; (4) **part-of-speech** – the two systems agree on the lemma, but have selected different parts of speech.

For each case covering a single phrasal difference, we compare the phrases from each system to the reference. We report the number of instances where each system matched the reference, as well as cases where they were both incorrect. The majority of differences correspond to clitics, whose correction appears to be a major source of the improvements obtained by lattice desegmentation. This category is challenging for the decoder because English prepositions tend to correspond to multiple possible forms when translated into Arabic. It also includes the frequent cases involving the nominal determiner prefix *Al* "the" (left unsegmented by the PATB scheme), and the sentence-initial conjunction *w+* "and". The second most common category is lexical, where the unsegmented LM has drastically altered the choice of translation. The remaining categories show no major advantage for either system.

6 Conclusion

We have explored deeper integration of morphological desegmentation into the statistical machine translation pipeline. We have presented a novel, finite-state-inspired approach to lattice desegmentation, which allows the system to account for a desegmented view of many possible translations, without any modification to the decoder or any restrictions on phrase extraction. When applied to English-to-Arabic translation, lattice desegmentation results in a 1.0 BLEU point improvement over one-best desegmentation, and a 1.7 BLEU point improvement over unsegmented translation. We have also applied our approach to English-to-Finnish translation, and although segmentation in general does not currently help, we are able to show significant improvements over a 1-best desegmentation baseline.

In the future, we plan to explore introducing multiple segmentation options into the lattice, and the application of our method to a full morphological analysis (as opposed to segmentation) of the target language. Eventually, we would like to replace the functionality of factored translation models (Koehn and Hoang, 2007) with lattice transformation and augmentation.

Acknowledgments

Thanks to Ann Clifton for generously providing the data and segmentation for our English-to-Finnish experiments, and to Marine Carpuat and Roland Kuhn for their helpful comments on an earlier draft. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the Ninth International Conference on Implementation and Application of Automata, (CIAA 2007)*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer. `http://www.openfst.org`.

Ibrahim Badr, Rabih Zbib, and James Glass. 2008. Segmentation for English-to-Arabic statistical machine translation. In *Proceedings of ACL*, pages 153–156.

Ondřej Bojar. 2007. English-to-Czech factored machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 232–239, Prague, Czech Republic, June.

Colin Cherry and George Foster. 2012. Batch tuning strategies for statistical machine translation. In *Proceedings of HLT-NAACL*, Montreal, Canada, June.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proceedings of EMNLP*, pages 224–233.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of ACL*, pages 176–181.

Ann Clifton and Anoop Sarkar. 2011. Combining morpheme-based machine translation with post-processing morpheme prediction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 32–42, Portland, Oregon, USA, June.

Mathias Creutz and Krista Lagus. 2005. Inducing the morphological lexicon of a natural language from unannotated text. In *In Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR05*, pages 106–113.

Ahmed El Kholy and Nizar Habash. 2012a. Orthographic and morphological processing for English—Arabic statistical machine translation. *Machine Translation*, 26(1-2):25–45, March.

Ahmed El Kholy and Nizar Habash. 2012b. Translate, predict or generate: Modeling rich morphology in statistical machine translation. *Proceeding of the Meeting of the European Association for Machine Translation*.

Alexander Fraser, Marion Weller, Aoife Cahill, and Fabienne Cap. 2012. Modeling inflection and word-formation in SMT. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 664–674, Avignon, France, April. Association for Computational Linguistics.

Nizar Habash, Owen Rambow, and Ryan Roth. 2009. Mada+tokan: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization. In Khalid Choukri and Bente Maegaard, editors, *Proceedings of the Second International Conference on Arabic Language Resources and Tools*, Cairo, Egypt, April. The MEDAR Consortium.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic, June.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, pages 586–594, Columbus, Ohio, June.

James W. Hunt and M. Douglas McIlroy. 1976. An algorithm for differential file comparison. Technical report, Bell Laboratories, June.

Minwoo Jeong, Kristina Toutanova, Hisami Suzuki, and Chris Quirk. 2010. A discriminative lexicon model for complex morphology. In *The Ninth Conference of the Association for Machine Translation in the Americas*.

Philipp Koehn and Hieu Hoang. 2007. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 868–876, Prague, Czech Republic, June. Association for Computational Linguistics.

Philipp Koehn, Franz Joesef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of HLT-NAACL*, pages 127–133.

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 *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June. Association for Computational Linguistics.

Minh-Thang Luong, Preslav Nakov, and Min-Yen Kan. 2010. A hybrid morpheme-word representation for machine translation of morphologically rich languages. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 148–157, Cambridge, MA, October.

Einat Minkov, Kristina Toutanova, and Hisami Suzuki. 2007. Generating complex morphology for machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 128–135, Prague, Czech Republic, June.

Franz Josef Och, Hermann Ney, Franz Josef, and Och Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29.

Franz Joseph Och. 2003. Minimum error rate training for statistical machine translation. In *Proceedings of ACL*, pages 160–167.

Kemal Oflazer and Ilknur Durgar El-Kahlout. 2007. Exploring different representational units in English-to-Turkish statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 25–32, Prague, Czech Republic, June.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Brian Roark, Richard Sproat, and Izhak Shafran. 2011. Lexicographic semirings for exact automata encoding of sequence models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1–5, Portland, Oregon, USA, June.

Mohammad Salameh, Colin Cherry, and Grzegorz Kondrak. 2013. Reversing morphological tokenization in English-to-Arabic SMT. In *Proceedings of the 2013 NAACL HLT Student Research Workshop*, pages 47–53, Atlanta, Georgia, June.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*.

Andreas Stolcke. 2002. Srilm - an extensible language modeling toolkit. In *Intl. Conf. Spoken Language Processing*, pages 901–904.

Michael Subotin. 2011. An exponential translation model for target language morphology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 230–238, Portland, Oregon, USA, June.

Kristina Toutanova, Hisami Suzuki, and Achim Ruopp. 2008. Applying morphology generation models to machine translation. In *Proceedings of ACL-08: HLT*, pages 514–522, Columbus, Ohio, June.

Nicola Ueffing, Franz J. Och, and Hermann Ney. 2002. Generation of word graphs in statistical machine translation. In *Proceedings of EMNLP*, pages 156–163, Philadelphia, PA, July.