Suffix Trees as Language Models

Casey Redd Kennington[1] **Martin Kay**[2] **Annemarie Friedrich**[3]

[1]Universität Bielefeld, Bielefeld, Germany, ckennington@cit-ec.uni-bielefeld.de
[2]Stanford University, Stanford, California, USA, kay@stanford.edu
[3]Universität des Saarlandes, Saarbrücken, Germany, afried@coli.uni-sb.de

Abstract

Suffix trees are data structures that can be used to index a corpus. In this paper, we explore how some properties of suffix trees naturally provide the functionality of an n-gram language model with variable n. We explain how we leverage these properties of suffix trees for our Suffix Tree Language Model (STLM) implementation and explain how a suffix tree implicitly contains the data needed for n-gram language modeling. We also discuss the kinds of smoothing techniques appropriate to such a model. We then show that our STLM implementation is competitive when compared to the state-of-the-art language model SRILM (Stolcke, 2002) in statistical machine translation (SMT) experiments.

keywords: language model, suffix tree, machine translation

1. Introduction

Suffix trees and suffix arrays can be used to index a text in such a way as to give access to a remarkable number of its properties. Not only can they be used to locate arbitrary substrings at very low cost, but they can be used, among other things, to find palindromes, n-grams of arbitrary size and long repeated substrings. Suffix trees can be used for data compression, since they make it convenient to allow the second and subsequent occurrences of repeated substrings to be replaced by a reference to the first. For some of these purposes, suffix arrays have sometimes been preferred in natural language processing (NLP) because they require less space and are thought of as easier to implement. In this paper, we explore how some properties of suffix trees naturally provide the functionality of an n-gram language model with variable n. Specifically, the efficient construction of suffix trees relies on *suffix links* which play a crucial role when using our STLM for evaluation.

One of the first uses of suffix trees was plagiarism detection (Monostori et al., 2000). Suffix trees provide a fast way of finding long strings which occur more than once, thus exposing parts of a text which may have been plagiarized. This is done simply by concatenating the text in question with the text that may have been plagiarized. After creating a suffix tree, non-terminal nodes which represent long substrings indicate overlaps.

Parameterized suffix trees have been used to find code duplication in software systems (Baker, 1997).

Suffix trees have been well used in information retrieval, more recently with Chinese documents (Huang and Powers, 2008). Because the vocabulary on which a suffix tree is based can consist of words or characters, suffix trees are especially useful when working with languages such as Chinese which do not use white-space as word boundaries. In a suffix tree, each character sequence is fully represented, unlike in other approaches which require error-prone word segmentation.

In this paper, we present an application of suffix trees as n-gram language models with variable n. n-gram language models are used to determine how probable it is for a string to belong to the language of a training corpus. From a list of candidate text representations that possibly represent an utterance, automatic speech recognition (ASR) uses language models to find the most probable text representation of that list. Machine Translation (MT) can produce a large list of possible translations of a text, all of which are checked against a language model to see how probable they are. Language modeling is discussed in more detail in Section 3.1.

The implementation of the STLM relies on a compressed version of a suffix tree, which is enriched by suffix links. Suffix links are edges that are added incrementally at building time to each node and that connect a node representing a string and the node representing its longest suffix. At evaluation time, we perform a tree traversal depending on the input string and leverage the suffix links for an elegant implementation of back-off. With each node, we keep the number of terminal nodes in its subtree as the node's suffix count. We show that these counts naturally provide n-gram counts of variable length, and add several versions of smoothing. Most language models limit the context in how the probabilities are calculated. We offer a language model that has no such limit. (Kneser, 1996) was successful in using unlimited context, but no evaluation was done in a practical domain, like ASR or MT.

We evaluate STLM within the setting of SMT and show state-of-the-art performance for some language pairs. For German to English and French to English, we achieve results better than the well-known state-of-the-art language model SRILM (Stolcke, 2002) when being trained using standard SRILM settings. SRILM outperforms STLM for several other language pairs, but STLM is still competitive.

The following section gives a brief explanation of suffix trees and then shows how they can be used as n-gram language models. We then discuss smoothing techniques and present the results of evaluating STLM and SRILM in machine translation experiments. We also speculate how our implementation allows for a dynamic language model which can learn by interaction.

2. Suffix Trees

In this section, we give an overview of suffix trees and explain how they can be built with limited time and space requirements.

2.1. Overview

A suffix tree is constructed on the basis of a text which we will refer to as its *corpus*. The size of the tree is a linear function of that of the corpus. It allows access to all substrings of the corpus in time that is a linear function of the length of the substring in question. The time required to build a suffix tree is also linear to the size of the corpus. Each arc of the structure is labeled, directly or indirectly, with a substring of the corpus, and the labels on the arcs from the root of the tree to a leaf, when concatenated, spell out a suffix of the corpus.

A suffix tree can be based on any unit - for instance characters or words - into which the corpus can be uniquely and exhaustively segmented. In our application, the units are words. However, we will base our initial examples on characters purely in the interest of saving space in our diagrams. The first words on the labels of the arcs leaving each node are pair-wise distinct. Consequently, the string of words that spells out the path from the root to any given node is unique and we will use the concatenation of the labels on the path as the *name* of the node. Before a suffix tree is constructed, a so-called *sentinel* word or character, which does not occur elsewhere in the corpus, is added to the end. This ensures that every suffix ends at a terminal node.

If an arc with a label consisting of two non-empty strings, say $\alpha\beta$, is replaced by a pair of arcs labeled α and β, with an intervening node, the resulting tree is equivalent to the original. However, it does not meet the definition of a suffix tree because it is required that there be at least two arcs out of each node (except the root) in a suffix tree. Arcs with multi-word labels are introduced precisely in order to meet this requirement. In fact, one way to construct suffix trees would be to first construct the equivalent structure with single-word labels (a *trie* in the sense of Fredkin (1960), see Figure 1) and then remove non-branching nodes, concatenating the labels on the arcs that enter and leave them. It is sometimes convenient to speak of the branching nodes as *explicit*, and of the locations between the words in an arc label as *implicit*, nodes. Clearly, implicit nodes can be named in the same way as explicit nodes in that the path to each of them from the root spells a unique string.

Note that whereas the number of arcs in a suffix *trie* for a corpus of m words is $\binom{m+1}{2}$, there can be no more than $2m - 1$ arcs in the corresponding suffix tree, for this is the number of arcs in a binary tree with m terminal nodes.

Figure 1 shows a simple suffix tree representing the text *mississippi$*.

2.2. Compression

In this section, we describe techniques that help to reduce the memory size required by a suffix tree, as proposed by Ukkonen (1995). These techniques give rise to a so-called *compressed tree*. We have already described the most important step in this process in which non-branching nodes are eliminated. However, this can clearly give rise to situations in which many arcs are labeled by very long strings. The second step therefore consists in replacing each of these strings with a pair of integers identifying the location in the corpus of their first occurrence. The first integer gives the position within the corpus where the string begins and the second gives either the position where it ends (a) or its length (b). We modify the standard scheme so as to eliminate the second of these integers. Observe that in case (a), there must be an arc emanating from the node at which the current arc ends with a label containing that position as its first integer, or the current arc ends at a terminal node. In the latter case, the second integer is simply the last position within the corpus (or the length of the corpus). For the former case, it suffices to ensure that we can locate the arc that carries the key piece of information, which we refer to as the *distinguished arc*. There are several ways of doing this. If the outgoing nodes are maintained as a list, for example, then it can be arranged that the distinguished arc always occupies its first position.

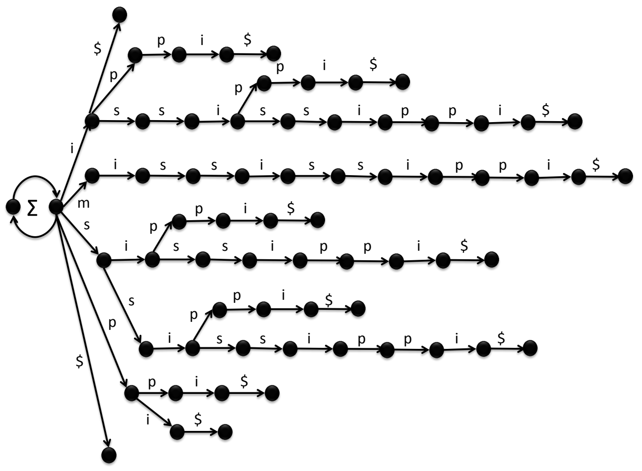

Figure 1: Suffix Tree for *mississippi$*

m	i	s	s	i	s	s	i	p	p	i	$
0	1	2	3	4	5	6	7	8	9	10	11

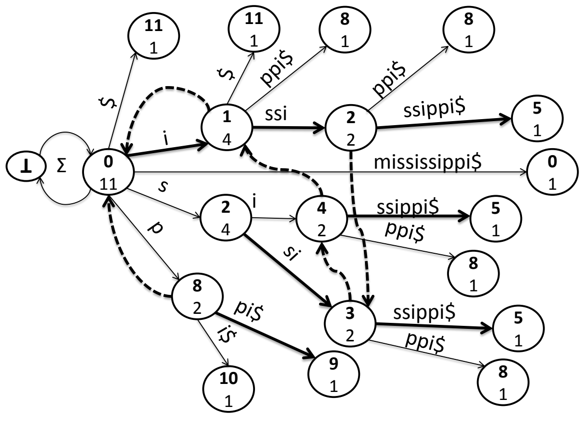

Figure 2: Compressed Suffix Tree for *mississippi$*

Figure 2 shows the suffix tree for the letters of the corpus *mississippi$*, with *$* being the sentinel character. Since the tree is compressed, the labels on the arcs are not explicit parts of the data structure but are shown only for convenience. The distinguished arc out of each nonterminal node is shown with a heavier line. The node labels, which *are* part of the data structure, are the upper numbers in the circles representing the nodes. The lower numbers in the circles are the suffix counts, which we will explain in section 3.2.. The root of the tree is shown in Figure 2 just to the right of the leftmost node, which we will refer to as the *base* of the tree. It is labeled $\perp$ and it has one arc, labeled Σ and ending at the root. This arc allows a transition from the base node to the root over any word, regardless of whether it occurs in the corpus. One might object to the inclusion of this node on the grounds that it is not strictly necessary and that it gives rise to a structure that is no longer strictly a tree. However, as we shall see, it is a convenience, both during the construction of the tree and in our proposed application.

2.3. Linear-Time Construction and Suffix Links

(Ukkonen, 1995) developed an algorithm for constructing suffix trees with a complexity that is linear in the length of the corpus. The dotted lines in Fig. 2 show explicit parts of the data structure know as *suffix links*. Only a few of them are shown. They play a crucial role in guaranteeing linear time bounds for the construction of the tree. In many applications, the suffix links can be discarded once the tree has been constructed. However, they are important for our application and we therefore retain them.

There is just one suffix link from each internal node in the tree. If the name of the node is $a\alpha$, where a is a character, and α is a, possibly empty, string of characters, then the suffix link from that node ends at node α. It is easy to see that such a node must always exist; if the corpus contains a string $a\alpha$ leading to an internal node, it must occur more than once in the corpus and be followed by different characters. If this is true of $a\alpha$, it must also be true of α. Explicit suffix links are stored only for explicit nodes. Note that they are defined also for implicit nodes. They can be found using suffix links in the following way. If $i = a\alpha\beta$ names an implicit node and $e = a\alpha$ is the longest prefix of this that names an explicit node, then the suffix link from the implicit node points to the node, implicit or explicit, that is found by following the suffix link from e, and moving forward from there over arcs whose labels spell β.

In Fig. 2 there are two paths of suffix links, both terminating at the root. In general, the suffix links themselves constitute a tree in which the links are oriented from the leaves towards the root.

3. n-gram Language Modeling using Suffix Trees

In this section, we explain n-gram language modeling and show how a suffix tree can be applied as a language model. This section ends with an explanation of smoothing, pointing out differences between our implementation and the current standard method.

3.1. n-gram Language Modeling

Language models provide an estimate of the probability that a given string belongs to the language of a particular reference corpus. The idea is to base the estimate of the probability of each word in a new text on the number of times that it was observed in a similar context in the reference corpus. In an attempt to minimize sparse data problems, the context of a word is routinely taken to be just the $n-1$ preceding words. The probability of the string $w_1, ...w_k$ is approximated by using the Markov assumption (here shown for n=2).

$$P(w_1, ..., w_k) \approx P(w_1) * P(w_2|w_1) * P(w_3|w_2) * ... \\ ... * P(w_k|w_{k-2}, w_{k-1})$$

The maximum-likelihood estimate of the probability of the word w_i given previous words $w_{i-n+1}, ..., w_{i-1}$ (the *history*) is computed as

$$P(w_i|w_{i-n+1}, ..., w_{i-1}) \approx \frac{C(w_{i-n+1}, ..., w_i)}{C(w_{i-n+1}, ..., w_{i-1})}$$

where $C(s)$ is the number of occurrences of the string s in the corpus. But cutting off the history to a pre-defined length n clearly does not eliminate the data sparseness problem entirely because many texts contain sequences of n words or less that occur nowhere in the training corpus. In these cases, a *back-off* to a lower-order n-gram model can be performed. On the other hand, when a matching string is found in the training corpus that is longer than n, there is a strong intuition that it could provide very valuable additional information. This is the intuition that motivates the present investigation.

3.2. Suffix Tree Language Model

In this section, we show how properties of suffix trees can be used to estimate the probability that an arbitrary text belongs to the language of the corpus, and how suffix links naturally provide back-off. As described in section 2., suffix trees provide the possibility to determine whether a given sequence of words occurs in a corpus. In the following, we show how probabilities for arbitrary sentences are computed from a language model based on a suffix tree. We assume that the elementary unit of the arc labels are words unless otherwise indicated.

In a suffix tree, the number of terminal nodes reachable from a given nonterminal node is the number of occurrences in the corpus of the string that names that nonterminal node. For example, the string *s* occurs four times in *mississippi$*, and four terminal nodes are reachable from the node that it names in Figure 2. The suffix counts are shown in the diagram as the lower numbers within the nodes. The strings *ss* and *ssi* each occur twice in the corpus and there are two terminals in the subtrees rooted at the nodes that they name. Note that the node named by *ssi* is an explicit node while the node named by *ss* is implicit. The number of terminal nodes reachable from each nonterminal node can be computed in a single walk of the tree and added to the representation of the nodes as *suffix counts*. The suffix count of leaf nodes is 1, while the suffix count of any

```
n = root; b = e = 0
p = 1
while e < length(text)
  m = n.get_branch(text[e])
  if m != NIL
     p *= m.suffix_count / n.suffix_count
     e += 1; n = m
  else
     n = suffix_link(n); b += 1
  end

end
```

Figure 3: Algorithm: STLM-Evaluation

other node is the sum of the suffix counts of its direct descendants. The count associated with a nonterminal node is simply the number of terminals encountered between the first and second times that the node is visited.

With the suffix count given, we can compute the probability of the first word on an arc as the suffix count of the destination node of the arc divided by the suffix count of the origin node. For example, the probabilities of $P(p|ssi)$ and $P(s|ssi)$ are both estimated as $1/2$. The probability $P(p|i)$ is estimated as $1/4$. By the same reasoning, the probability of edges starting at implicit nodes is always 1, e.g. $P(i|ss) = 1$.

Based on the suffix counts, our algorithm delivers the estimated probability of traversing an arc, given the label on the node at its origin. This estimated probability corresponds to traditional n-gram language model probability estimates.

The algorithm presented in Figure 3 for computing the probability of a sentence `text` assumes operating on the non-compressed tree where each node is represented explicitly.

Each time the variable e is incremented, a new substring, longer then the last one, is considered. When this happens, we will say that the algorithm *advances*. Each time b is incremented, a shorter substring is considered and we say that the algorithm *backs off*. We use the term *depth* of the process to refer to the current value of $e-b$, i.e the length of the current history. Notice that if the depth of the process is prevented from exceeding some fixed value, n, by forcing back-off, we have a classical n-gram model [1].

3.3. Smoothing

Discounting is easy to implement in this framework, and amounts to a simple initial smoothing approach. The model we have described is well behaved in the face of texts containing words that do not occur in the corpus, thanks to the base node. Failing to find the word at the root node, the algorithm repeatedly backs off until it reaches the base node, where it finds an arc that can be taken over any word whatsoever back to the root. Since the number of terminals reachable from the base and the root nodes are the same, namely $|\Sigma|$, the number of words in the corpus, the probability associated with this arc will be 1. This is clearly unsatisfactory. We can, however, easily arrange to associate

[1]This was convenient in comparing our variable-order model with standard fixed-order models.

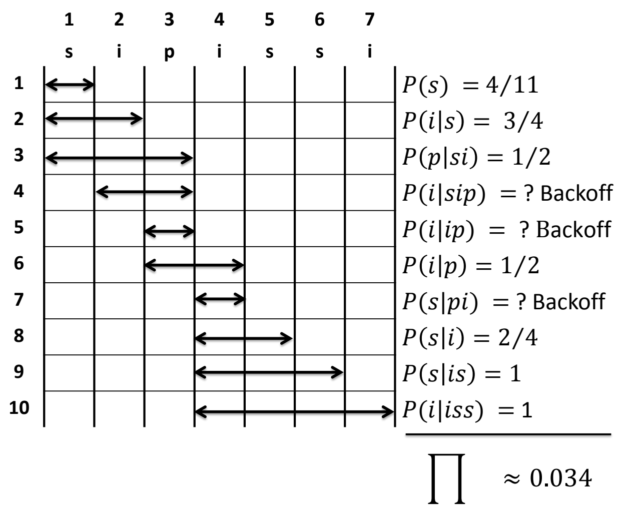

Figure 4: Matching and computation of the probability of a string

a much more reasonable probability with this arc. Suppose that, instead of $|\Sigma|$, we provide the base node with a suffix count of $|\Sigma^2|$. This will associate a probability of $1/|\Sigma|$ with the transition. If we do this, we should, strictly speaking, adjust the computations of all other probabilities so as to normalize them properly. But, as has frequently been observed, if the corpus is large enough to be useful for the present purpose, this correction will be too small to be worth making. However, normalization can be easily achieved by subtracting $1/|\Sigma|$ from the suffix count of each node in the tree. This kind of simple smoothing facilitates a model that is potentially dynamic, which is discussed in section 5.

STLMs are hospitable to most of the other variations in the basic n-gram scheme, notably Kneser-Ney smoothing (Kneser and Ney, 1995), in its original and modified variants (Chen and Goodman, 1998). The structure naturally allows for raw counts and back-off information. It also provides much of what is required to count histories so that they can be added to the structure along with, or in place of, the raw counts. Observe that the internal nodes of the suffix trees correspond exactly to the set of histories. It therefore suffices to walk the tree and, whenever a word w is encountered at depth d, add the pair $< w, d >$ to a file or list. Then sort this collection to bring all the records for a given word together and, within those, all the records for a given depth. Word w has k histories at depth d iff the file contains k records of the form $< w, d >$.

When the process has reached a depth d, it has the probability estimates for the last d words, which it can use to compute a probability estimate for the current substring. In principle, this can be done using standard interpolation methods. However, it requires that meaningful constants for use in the interpolation be available for substrings that are much longer than those considered in classical n-gram models. Our experiments gave better results when all of the distributions for different n were given equal weight. On the other hand, we found that a relatively crude way of assigning differential weights gave dramatically improved results.

The probability of a string $w_1, ..., .w_k$ is computed as $P(w_1) * P(w_2|h2) * ... * P(w_k|h_k)$ where h_i is the longest history found in the suffix tree for w_i, the next word in the test sequence. During the traversal of the tree, the length of history h_i plus the current word in question defines the current depth in the tree. As a modification (the *boosted model*), we propose the following formula, which rewards strings that were able to match longer sequences of the training corpus:

$$Score(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i|h_i) * (length(h_i) + 1)$$

Our implementation also incorporates a version of Kneser-Ney smoothing with boosting. The diversity of histories is interpolated with the raw counts at all depths. It made essentially no difference whether the Chen and Goodman discount values were computed at depth 2 or 6, at least for the English Europarl corpus. Diversity of history and raw counts are given weights of 0.05 and 0.95 respectively at all depths, which is quite different from standard Kneser-Ney. Interpolation as defined in Kneser-Ney was not used because it consistently made the model perform worse.

4. Evaluation

In this section we present our evaluation of the STLM using MT experiments. First, we show some properties of the STLM. We then determine the best settings for STLM and the SRILM, and use those in several MT experiments.

4.1. Information Represented in the STLM

In order to investigate some properties of the STLM, we trained the STLM using 100,000 lines of the English Europarl corpus. Key statistics are given in Table 1.

CORPUS		SUFFIX TREE	
Lines	100,000	Nodes	3,533,984
Words	3,005,366	Leaf	2,806,210
		% Leaf	20.6
		Internal	727,774
		% Internal	79.4

Table 1: Statistics for STLM instantiation using Europarl (EN).

The JRC-Acquis corpus (Steinberger et al., 2006) was used for the evaluation and the relevant statistics of this are given in Table 2. *Maximum depth* refers to the depth reached before any back-off was performed. Observe that there was at least one string of 15 words in the evaluation text that also occurred in the training text. The table also gives the average depth per word that the evaluation process reached. 5.2% of words led to an implicit node in the tree so that the following transition was assigned a probability of 1. Table 3 shows examples of these multi-word nodes found in Europarl. The left column gives the number of occurrences of the multiword expression in the training corpus. The number of words which did not occur in the training corpus (unseen) made up 3.4%, so they caused a transition from the base to the root node. In 91.4%, the probability of the next token was computed based on the suffix counts of the child and mother nodes.

CORPUS		STLM EVALUATION	
Sentences	4,108	Max. depth	15
Av. length	31.37	Av. depth	3.38
Words	128,879	Back-offs	100,688
		implicit nodes	5.2%
		regular	91.4%
		unseen	3.4%

Table 2: Statistics for test corpus (JRC Acquis).

C	String
2	Substantially Less Interference by Members
3	overstepping the
4	callous disregard
4	five-legged sheep
4	nautical miles
5	non-economically active
6	oriental carpets
6	revitalization of the
7	sportsman or woman
24	Camp David

Table 3: Multiword expressions of arcs having implicit nodes in Europarl

Table 4 gives a breakdown of multi-word arcs leaving the root. The columns titled 'L' give the depth of the sequence and the columns titled 'C' the number of such nodes. Not surprisingly, the distribution fits Zipf's law fairly closely.

L	C	L	C	L	C	L	C
2	1237	9	14	16	3	24	1
3	359	10	8	17	3	25	2
4	142	11	9	18	3	26	3
5	69	12	9	19	1	27	4
6	44	13	6	20	3	28	1
7	27	14	3	21	5		
8	16	15	9	22	1		

Table 4: Multi-word arcs

We motivated our use of suffix trees by appealing to the intuition that long repeated strings, even if not repeated very often, could contribute valuable information to a variable-order language model that fixed-order models routinely throw away.

4.2. Statistical Machine Translation Experiments

The quality of language models is often evaluated using *perplexity*, which is a measure of the likelihood of a given test corpus. Perplexity is only meaningful when being calculated for a normalized model, which the boosted version

n	BLEU	Settings
7	10.40	kndiscount, interpolate, unk
5	10.37	kndiscount, interpolate, unk
4	10.45	kndiscount, interpolate, unk
4	**10.51**	**kndiscount, interpolate**
4	10.41	kndiscount

Table 5: SRILM Settings Comparison

of STLM is not. We therefore favored a *task-based* evaluation, which shows the quality of the language model in a real application setting.
We evaluated the STLM in the setting of machine translation using the Moses MT System (Koehn et al., 2007) and compared it to the SRI language model (SRILM) (Stolcke, 2002). Suffix array language models such as SALM (Zhang and Vogel, 2006) and (Stehouwer and Van Zaanen, 2010) were not compared due to time constraints, though a comparison certainly would have been interesting and informative. We also didn't compare against SALM implementations because they don't use Kneser-Ney smoothing in any variant, which is the current state of the art in smoothing for MT. SRILM and STLM both have a version of Kneser-Ney implemented. Another practical difference was that SRILM can be used directly with Moses (notably in the Experiment Management System), and we created a version of Moses that directly uses STLM in the same way as it uses SRILM. It suffices to compare STLM against the state-of-the-art SRILM to estalish its effectiveness in MT.
We used the BLEU (Papineni and Et al, 2002) and METEOR (Lavie and Denkowski, 2010) scoring metrics. Both metrics calculate a score which compares the MT output hypothesis with a reference text. BLEU calculates a modified form of precision using n-gram overlap, and METEOR uses the harmonic mean between precision and recall in unigrams, as well as stemming and synonymy matching. A useful script that produces these scores given the machine translation hypotheses along with the reference target translation is provided by Kenneth Heafield[2].
The established standard for SRILM in MT is to use Kneser-Ney smoothing with interpolation. The training corpora used were Europarl (Koehn, 2005), an evaluation set of JRC-Acquis (Steinberger et al., 2006), and the newstest2009 corpus. In all experiments, a phrase-based translation model was trained on the parallel parts of Europarl of the two languages in question, and the language models were trained using the target language corpus.
We ran some simple small-scale tests to establish the best settings for the two language models. Tables 5 and 6 show the BLEU scores for various settings using the first 200k lines of Europarl (German to English) for training, and the newstest2009 corpus for evaluation.
It shouldn't be surprising that the higher n-gram models don't work as well in practice. This seems like a major blow to the concept of variable length n-grams, but it is due to data sparsity. We return to this issue in Section 5.
In Table 6 the difference between the first and second lines show when boost was not used (line 1), and when boost

[2]http://kheafield.com/code/mt/

BLEU	Settings
9.26	kndisc, interpolate, unk, no depth limit
11.55	kndisc, interpolate, unk, no depth limit, boost
11.96	**kndisc, interpolate, unk, depth limit 4, boost**

Table 6: STLM Settings Comparison

was used (lines 2 and 3). This shows that our implementation of Kneser-Ney did not work as well as SRILM's. The boost, however, resulted in a remarkable performance improvement.
With the best setting for each model established, we performed experiments for various language pairs. The entire Europarl corpus (1,540,549 sentences) was used for training and the JRC-Acquis corpus (4,108 sentences) was used for evaluation. Table 7 shows the results for German, French, and Spanish where English was the source language, and Table 7 where English was the target language.

LM	LANG	METEOR	BLEU
STLM	DE	**56.27**	**26.93**
SRILM	DE	51.34	21.92
STLM	FR	**66.65**	**37.29**
SRILM	FR	66.62	36.69
STLM	ES	64.08	33.73
SRILM	ES	**64.24**	**33.93**

Table 7: English as **target** language, training: full Europarl, test: JRC Acquis.

LM	LANG	METEOR	BLEU
STLM	DE	24.76	23.17
SRILM	DE	**25.48**	**23.96**
STLM	FR	20.45	33.61
SRILM	FR	**22.42**	**35.11**
STLM	ES	30.01	31.15
SRILM	ES	**30.32**	**32.35**

Table 8: English as **source** language, training: full Europarl, test: JRC Acquis.

STLM performs better in DE-EN and FR-EN in both evaluation metrics. SRILM outperforms STLM in both metrics for all other language pairs. These results show that STLM was perhaps over-tuned to work with DE-EN corpora, which was used for the intitial development, but it is still a competitive language model in our MT experiments when compared to SRILM, which is the result of many years of language modeling research and tuning.
Table 9 shows a final evaluation which was done on STLM using German-English, where the depth was limited to 4 as before (same as in Table 7), and where the depth was unlimited. The same training and evaluation data sets as before were used. Though the depth-4 model still performs better for BLEU, METEOR awards a slightly higher score for the unlimited depth STLM. Compared to the experiments related to the initial settings determination, we can see that more training data helps resolve the data sparseness prob-

Version	METEOR	BLEU
depth limit 4	56.27	**26.93**
unlimited	**60.03**	26.20

Table 9: German to English STLM

lem.

4.3. Memory Usage

It is important to compare the memory usage of the two language models. The STLM grows linearly over time during training, peaking at 32 times the corpus size. During evaluation it uses 21 times the corpus size. SRILM in contrast only uses 1.066 times the size of the corpus during training (for a 4-gram, which is the model we used) and runtime memory usage is about one-fourth the size of the corpus. However, as the SRILM training n-gram order increases, it requires more memory for training. For STLM, the memory usage stays constant, but for SRILM when the n-gram order is above 11, it uses more memory than STLM. During evaluation, however, the memory necessary for SRILM is still much lower than STLM. It should also be noted that suffix array language model implementations are known to use significantly less memory than suffix trees during training and evaluation.

5. Discussion

The STLM is able to accommodate unlimited n-gram lengths, but it was shown that a 4-gram model performs better overall than an unlimited model in some MT experiments. This mostly due to data sparseness (see (Kneser, 1996)). Figure 9 shows that an unlimited model regains ground when more training data is used, and that using longer histories is rewarded at least when comparing the METEOR scores.

Only the boosted version of the STLM can compete with SRILM, which points to the potential weaknesses of our implementation. Ideally, without boost, it should perform reasonably well against SRILM. It would be informative to see how well SRILM performs with similar boosting, or if it is something that is uniquely helpful to this kind of structure. Another criticism of the boost is that it potentially rewards longer sentences by including more integer multiplications in the sentence score. While this is a valid criticism, large boosts will only occur when longer word histories are present in the training corpus, which is the very motivation of the suffix tree as a language model. A long sentence which has words represented in the tree, but where none of the sequences are represented, will have no boost at all (each word probability is multiplied by 1), despite being a long sentence.

There is another benefit to using STLM as opposed to traditional n-gram language models. The STLM differs from traditional language models in that it is inherently dynamic when using simple smoothing. This is because it allows all of the back-off pointers and node counts to be up-to-date and usable at any given moment during training. This means the tree can be trained, used for evaluation, then given more data to improve the model in a dynamic way. Imagine a virtual agent that uses a language model to aid in utterance generation, then reads in a response from a human user, which is assumed to be a well-formed representation of the language. This new information can be used to improve the language model of the virtual agent by simply appending onto the STLM data structure, which would result in better counts and improved generated utterances. Normal n-gram models can be made to be used dynamically, but that ability is again implicit in this structure. It is arguably not dynamic in that, for the tree to be complete, it requires the final sentinel character, but in a language model application neglecting to add this final character at the end does not affect the counts and branching of the model to completely and accurately represent the corpus. Only the suffix counts need to be updated, which is normally done once by a simple walk of the tree after the tree is built. It would cost a lot computationally to always need to walk the entire tree even after a small addition of information. However, count updating can also be performed in an efficient way by updating the counts of any branch affected by newly added data by simply following parent nodes, updating each count, up to the root node.

6. Conclusions

Suffix trees contain properties that are useful when processing natural languages. Evidence of this was further established in this paper by a language model implementation. The STLM offers true variable-length n-grams as a language model. It is comparable in most language pairs in some MT experiments, and is significantly better for DE-EN. Another useful point of the STLM is that the data structure naturally holds its state such that it can be trained dynamically, which may be useful in some applications.

7. Future Work

In the future, we plan on improving the smoothing further and exploiting more properties of the data structure. Due to time constraints, the language model was trained on Europarl and tested in the news domain. Running in-domain MT experiments would be beneficial. We also hope to test the model in other applications such as speech recognition or information retrieval. We further desire to test the language model in a dynamic environment, such as on-line dialogue processing.

The STLM toolkit, written in C++, is available to anyone by e-mail request. We also have a version of the Moses decoder that can directly use the STLM.

Acknowledgements: This work was partially funded by the DFKI ACCURAT project. Thanks to Jia Xu for access to corpora and computing resources. Thanks also to the various reviewers.

8. References

Brenda S Baker. 1997. Parameterized Duplication in Strings: Algorithms and an Application to Software Maintenance. *SIAM J. Comput.*, 26(5):1343–1362, October.

Stanley Chen and Joshua Goodman. 1998. An Empirical Study of Smoothing Techniques for Language Modeling.

Technical Report TR-10-98, Computer Science Group Harvard University, August.

Edward Fredkin. 1960. Trie memory. *Communications of the ACM*, 3(9):490–499.

Jin Hu Huang and D Powers. 2008. Suffix Tree Based Approach for Chinese Information Retrieval. In *Eighth International Conference on Intelligent Systems Design and Applications*.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, volume I, page 181â[U+0080][U+0093]184, Detroit, Michigan, May.

R Kneser. 1996. Statistical language modeling using a variable context length. In *Proceeding of Fourth International Conference on Spoken Language Processing ICSLP 96*, volume 1, pages 494–497. Ieee.

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.

Philipp Koehn. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Conference Proceedings: the tenth Machine Translation Summit*, pages 79–86, Phuket, Thailand. AAMT, AAMT.

Alon Lavie and Michael Denkowski. 2010. The METEOR Metric for Automatic Evaluation of Machine Translation. In *Machine Translation*.

K Monostori, A Zaslavsky, and H Schmidt. 2000. Identifying Overlapping Documents in Semi-Structured Text Collections. Australasian Computer Science Conference.

Kishore Papineni and Salim Roukos Et al. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*.

H Stehouwer and M Van Zaanen. 2010. Using Suffix Arrays as Language Models: Scaling the n-gram. *Proceedings of the 22nd Benelux Conference on Artificial Intelligence BNAIC*.

Ralf Steinberger, Bruno Pouliquen, Anna Widiger, Camelia Ignat, Tomaž Erjavec, Dan Tufiş, and Dániel Varga. 2006. The JRC-Acquis: A Multilingual Aligned Parallel Corpus with 20+ Languages. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC'2006)*. LREC.

Andreas Stolcke. 2002. SRILM - An Extensible Language Modeling Toolkit. International Conference on Spoken Language Processing.

Esko Ukkonen. 1995. On-line construction of suffix trees. *ALGORITHMICA*.

Ying Zhang and Stephan Vogel. 2006. Suffix Array and its Applications in Empirical Natural Language Processing. Technical report, CMU, Pittsburgh PA, December.