Improved Count Suffix Trees for Natural Language Data

Guido Sautter *sautter@ipd.uka.de*
Researcher
Department of Computer Science
Karlsruhe Institute of Technology
Am Fasanengarten 5, 76128 Karlsruhe, Germany

Klemens Böhm *boehm@ipd.uka.de*
Full Professor
Department of Computer Science
Karlsruhe Institute of Technology
Am Fasanengarten 5, 76128 Karlsruhe, Germany

Abstract

With more and more text data stored in databases, the problem of handling natural language query predicates becomes highly important. Closely related to query optimization for these predicates is the (sub)string estimation problem, i.e., estimating the selectivity of query terms before query execution based on small summary statistics. The Count Suffix Trees (CST) is the data structure commonly used to address this problem. While selectivity estimates based on CST tend to be good, they are computationally expensive to build and require a large amount of memory for storage. To fit CST into the data dictionary of database systems, they have to be pruned severely. Pruning techniques proposed so far are based on term (suffix) frequency or on the tree depth of nodes. In this paper, we propose new filtering and pruning techniques that reduce the building cost and the size of CST over natural-language texts. The core idea is to exploit the features of the natural language data over which the CST is built. In particular, we aim at regarding only those suffixes that are useful in a linguistic sense. We use (well-known) IR techniques to identify them. The most important innovations are as follows: (a) We propose and use a new optimistic syllabification technique to filter out suffixes. (b) We introduce a new affix and prefix stripping procedure that is more aggressive than conventional stemming techniques, which are commonly used to reduce the size of indices. (c) We observe that misspellings and other language anomalies like foreign words incur an over-proportional growth of the CST. We apply state-of-the-art trigram techniques as well as a new syllable-based non-word detection mechanism to filter out such substrings. – Our evaluation with large English text corpora shows that our new mechanisms in combination decrease the size of a CST by up to 80%, already during construction, and at the same time increase the accuracy of selectivity estimates computed from the final CST by up to 70%.

Keywords: Selectivity Estimation, Count Suffix Tree, Pruning, Text Data

1. INTRODUCTION

With more and more natural language data stored in databases, query processing for this type of data becomes highly important. To optimize queries over this kind of data, the (sub)string estimation problem is vital. This is estimating the selectivity of natural language query predicates, usually term-based before the actual query execution, based on small summary statistics. The selectivity of a term (or of any substring) is the number of documents in the underlying collection it appears in. To estimate the selectivity of predicates of the kind considered here, Count Suffix Trees (CST) are commonly used [8]. According to [8], each CST node stores the selectivity of the string along the path from the root to the node. The selectivity of a string can be retrieved in a time linear to its length. A CST built over text data can efficiently solve the selectivity-estimation problem.

However, CST are computationally expensive to build and have high memory requirements. The space complexity of a CST is proportional to the size of the underlying alphabet and to the number of strings stored in the CST[1]. A CST built over a large amount of text data may well exceed 1,000,000 nodes. Since the statistics data structures used by the query optimizers of database systems have to fit in the data dictionary, i.e., in a very limited amount of memory, CST used for query optimization need to be reduced in size [4]. This is because they always have to be in physical main memory. If query optimization caused only a single page fault (i.e., the need to swap a memory page from on-disk virtual memory back into physical main memory), this would annihilate the performance advantage database systems gain from query optimization. One might think that the 1 KB limit from [8], published in 1996, is too strict nowadays, and that modern database servers can have statistics that are larger by orders of magnitude. However, not only the amount of memory available to database servers has rapidly grown since 1996. The amount of data to be handled has grown at a similar rate, both regarding the number of relations and of attributes per relation. Thus, the data dictionary has to accommodate significantly more statistics. Consequently, it is not unrealistic to assume that the statistics for an individual attribute still have to fit into 1 KB with today's commercial database servers [24].

To make the CST meet these restrictive memory requirements, a common solution is to apply pruning rules. Discarding some nodes, for instance those with the lowest selectivities [7, 8], saves space. But it also affects the accuracy of estimations. To deal with this problem, methods for estimating the selectivity of strings that are not retained in the Pruned CST (PST) any more have been proposed. Algorithms like KVI or MO [8, 7] alleviate estimation inaccuracies due to pruning to some degree, but do not rule out the problem. Pruning becomes even more problematic with non-static document collections, e.g., journal archives or the Blogosphere. Estimation errors may arise, due to incorrect node counts [1]. The only solution currently known is to rebuild the CST over the updated collection. Even though algorithms that reduce space and time complexity have been proposed [17, 18], CST construction remains computationally expensive and time consuming.

The goal of our work is finding other ways of reducing the size of the CST, i.e., filtering out suffixes. We focus on natural-language texts. Our core idea is to find linguistic criteria that let us decide, prior to insertion in the CST, which strings or suffixes are more likely to be queried. For the CST, we keep only the latter, and we deal with the rest separately to reduce the size of the CST during construction already, before the actual pruning. In particular, we apply syllabification, stemming, and non-word detection. The combination of these mechanisms allows for building a tree that requires significantly less memory than state-of-the-art CST. More specifically, the contributions of this paper are as follows:

Design of a New Approximate Syllabification Algorithm for CST-specific Data Preprocessing

We observe that letter-wise suffixes that do not start at a syllable border carry little semantic meaning. To filter out these suffixes we propose a fast *approximate* syllabification routine, based on the morphological structure of words. In order to avoid filtering too many suffixes, however, we have to find every syllable boundary, even at the cost of some false positives. Therefore, our routine is more aggressive than conventional ones. We will show that avoiding the insertion of suffixes that do not start at syllable borders reduces the size of the CST by much, not only for storage, but also during construction.

[1] Note that this mostly holds for languages in which words tend to consist of many letters, like the languages written in the Latin, Greek, Cyrillic, Arabic, Hebrew, or Thai writing system – these are the ones we deal with here For languages written in symbolic writing systems, like Chinese, Japanese, or Korean, words consist of far fewer symbols, and other data structures are better suited for selectivity estimation altogether, e.g. histograms.

Design of an Optimistic Suffix and Affix Stripping Algorithm
Stemming, that is conflating different inflections of a term to the same root form, further reduces the number of suffixes. We observe that traditional stemming algorithms like Porter's stemmer [16] are rather conservative, i.e., omit some conflations to avoid errors. We propose a new, more optimistic stemming procedure. It conflates more terms and thus reduces the number of suffixes to store. This procedure also includes prefix stripping, which moves the stem up front. The rationale is that the stem carries the most semantic meaning, and the nodes closest to the root of a CST are the last ones to be pruned. Note that linguistic errors may occur with this approach. But we show that their effect on estimation quality is likely to be insignificant.

Non-word Filtering
Experience shows that non-words and foreign words incur an over-proportional number of nodes in the CST. They also increase the memory consumption in the building phase of the tree. We therefore deploy a q-gram based algorithm for non-word detection to prevent inserting non-words in the CST. In particular, we exclude terms including very rare q-grams. To estimate the selectivity of non-words, we use a variant of what [4] refers to as *q-gram estimator* instead. Because we apply this technique only to words with highly distinctive q-grams, we do not encounter the errors reported in [4]. In contrast, our evaluation shows that the q-gram technique yields very good selectivity estimates for non-words.

New Node Labeling sStrategy
Traditional CST construction mechanisms suggest labeling each node with a single character and applying path compression when the tree is completely built. We propose a new node labeling mechanism which is syllable-based. It results in a more compact CST. In particular, we use syllables as the atomic node labels, instead of individual characters. While this increases the fan-out of the tree, it reduces its depth significantly, resulting in more compact CST.

Extensive Performance Experiments and Feasibility Demonstration.
We run a series of evaluations over large English document corpora. It turns out that syllable CST require significantly less space. For instance, the size of the CST built over datasets from the Aquaint Corpus [6] is reduced by up to 80%. We also show that the benefit of Syllable CST over traditional ones grows with the number of non-words and misspellings in the corpus, by introducing errors into it in a controlled way. Thanks to the reduced memory occupation, frequency-based pruning can then use lower thresholds. This results in more accurate estimations. We experimentally verify that, when pruned to meet the same size, Syllable CST provide significantly better selectivity estimates than standard ones. On average, the relative estimation error is reduced by up to 80%.

Although we use English language corpora for our experiments, the techniques presented in this paper are not restricted to English: They are applicable to any character-based language, provided that a small reference dictionary (for the non-word filtering) and a stemming and syllabification routine (for building the Syllable CST) are available.

Paper Outline
Section 2 reviews related work. Section 3 describes the syllabification and non-word detection techniques, Section 4 the design of the Syllable CST, and how to use it for selectivity estimation. Section 5 features our evaluation. Section 6 concludes. [23] is an shorter, preliminary version of this paper. This current article features a new node-labeling strategy, a more detailed description and discussion of our algorithms, and more extensive experiments which assess our techniques with documents containing spelling errors.

2. RELATED WORK

The Count Suffix Tree (CST) [8] is a data structure commonly used to estimate the selectivity of string predicates. Given a collection of documents, all strings and their suffixes are stored in the CST. Each node is labeled with a string and has a counter that stores the number of occurrences in the collection. Since CST tend to grow quickly in size when built on large text datasets, pruning strategies are essential to make the data structure meet memory constraints. Pruning requires estimating the selectivity of those terms whose node has been discarded. Krishnan et al. [8] has proposed three families of estimation methods. Among these, the KVI algorithm yields the most accurate results according to experiments. The MO (Maximal Overlap) algorithm, proposed by Jagadish et al. [7], outperforms KVI when the statistical short memory property holds for sequences of symbols. According to MO, the searched pattern is parsed in overlapping (when existing) substrings, which are considered statistically dependent. Since both KVI and MO tend to underestimate selectivities, Chaudhuri et al. [4] propose a new estimation model based on q-gram Markov tables and a regression tree. Bae [1] describes which estimation inaccuracies may arise in the presence of pruning and tries to overcome the problem by building a Count Q-gram tree. While it is useful for DNA data (alphabet size 5), [1] also shows that it is worse when the alphabet size increases. It is hardly applicable for natural language data (alphabet size 26).

Statistics on collections of text data [2] must be updated when new document are added to the collection. Once nodes have been pruned, however, there is no information left on the previous selectivity of removed strings. If they appear to be frequent in newly added documents, the node counts are incorrect. This is because they do not include the selectivities before pruning. As mentioned in Section 1, the only way to solve this problem is to rebuild the CST over the updated collection. Algorithms have been developed for constructing in-memory suffix trees in linear time, proportional to the number of strings stored in it, by Weiner [19] and McCreight [15]. Ukkonen [18] later designed an online version. The ever-increasing amount of available text data however calls for disk-based construction algorithms. The “Top Down Disk-based” strategy by Tian et al. [17], despite the fact that it runs in quadratic time, is faster than the linear in-memory alternatives, and its space consumption is lower than that of other algorithms described in literature. Even if disk-based strategies have significantly decreased time and space building overhead, the computational effort is still in the way of rebuilding the tree frequently. On the other hand, the final CST has to fit in main memory to let query optimizers estimate selectivities in short time. Thus, pruning and its drawbacks cannot be avoided. The goal of our work therefore is to find pruning strategies that incur less inaccuracy than the existing standard ones.

We will refer to further related work, in particular to algorithms from computational linguistics which we adapt and deploy in our context, in Sections 3 and 4.

3. SYLLABLE COUNT SUFFIX TREE

This section proposes a new variant of the CST data structure for selectivity estimation on collections of natural-language texts, the Syllable CST. Section 3.1 explains how syllabification reduces the size of the suffix tree. Section 3.2 presents our new syllabification routine. Sections 3.3 and 3.4 explain why Porter’s algorithm is not sufficient for our purpose, and why we implement another stemming routine. Section 3.5 illustrates the drawbacks of inserting non-words in the data structure and proposes a solution.

3.1 Syllabification

According to the original definition of suffix tree [19], inserting an index term in the tree implies generating all of its suffixes and inserting them as well. Given a string σ of length n, defined over the alphabet Σ and a string terminator symbol $ (not in Σ and lexicographically subsequent to any symbol in it), the i-th suffix is the substring starting with the i-th character of σ and terminated by $.

Example 1. Given σ = information$, its suffixes are: (*information$, nformation$, formation$, ormation$, rmation$, mation$, ation$, tion$, ion$, on$, n$, $*). □

Some of these suffixes are very unlikely to be ever addressed in a user query. The reason is that they convey very little semantic meaning, e.g., -*rmation*. Syllables are natural word building blocks in many languages. Using syllable-division points to compute suffixes reduces their number. The remaining ones carry an enhanced semantic message at the same time.

Example 2. Given the syllabified string σ = *in-for-ma-tion-$*, the set of its syllable suffixes is: (*in-for-ma-tion-$, for-ma-tion-$, ma-tion-$, tion-$, $*). □

Syllabification proves to be convenient to filter out suffixes that start with unusual combinations of letters. Since the number of suffixes that need to be stored in the tree decreases, its size is reduced as well. Figure 1 contrasts memory requirements of a standard CST built over the string 'information' with the equivalent Syllable CST. The number of nodes is more than halved. We expect a similar size reduction for larger datasets as well. Experimental evaluations will confirm this hypothesis (see Section 5).

Discussion. Syllabification-based filtering is not limited to English; it is applicable to any character-based language, provided that there is a syllabification routine. In this paper, we use English text for the examples and for the experiments.

Clearly, filtering out suffixes with the mechanism described here affects selectivity estimation for substrings that do not start at syllable boundaries. For instance, the Syllable CST in Figure 1 would not be able to estimate the selectivity of the predicate *LIKE '%nfo%'*. This is not a severe drawback (we argue). Namely, queries over natural language text, even if it is dirty, are likely to contain rather "natural" text fragments, e.g., *LIKE '%info%'* or *LIKE '%inform%'*, as opposed to *'%nfo%'*.

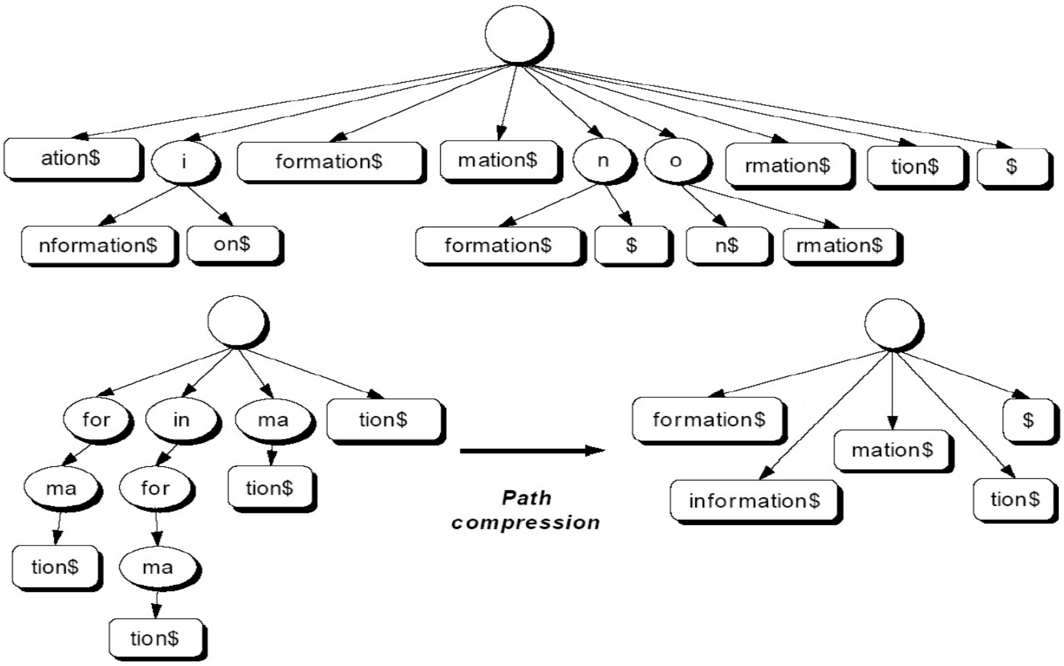


FIGURE 1: Syllable CST on the String *information$*

3.2 The Syllabification Routine

The problem of syllabification of written words is strictly tied to the hyphenation (or justification) task [13]. Algorithms for splitting words at syllable boundaries can be classified as rule-based or dictionary-based. The latter ones provide orthographically correct syllable division points by performing a lookup in a dictionary. Although they guarantee greater accuracy, there are several reasons why we did not pursue this option. First, since we want to minimize space requirements, the overhead of a dictionary is not tolerable. The dictionary size strongly depends on the content and the size of the corpus. The New York Times dataset of the Aquaint Corpus, for instance, contains 352404 different terms. The size of the dictionary should be of the same order of magnitude. Second, no matter the size of the dictionary, there will always be words that are not contained in it (foreign language words, proper names, etc.). Furthermore, a language keeps evolving, and new terms constantly enrich the dictionary. Third, since many syllabification rules are based on how the word is pronounced, there are differences regarding syllabification of the same term according to different English dictionaries.

Rule-based hyphenation systems, such as the LATEX algorithm [13], are an alternative. They are typically faster and require less storage space, but they are inherently subject to errors. The reason is that, even if a set of general rules has been defined, based on sophisticated linguistic literature, syllabification is not an “exact” process. Most of the rules are based on the sound of the spoken word and are not easy to implement. The so-called VV rule is an example [5]. It suggests separating two consecutive vowels in two distinct syllables if they do not form a diphthong, or considering them as part of the same syllable otherwise. Without any further phonological information, it is impossible to tell if adjacent vowels form a diphthong or not. Since our goal is exploiting syllabification to reduce memory requirements of the CST, however, we have adopted a rule-based solution anyway. It trades grammatical accuracy for a reduction of computational overhead and extra data required.

Example 3. The diphthong *ie* is split in *sci-ence*, since the vowels are pronounced separately. The same diphthong in *re-triev-al* belongs to one syllable. □

To achieve high-quality results, the hyphenation routine of LATEX [13] uses close to 5.000 rules in 5 levels. The representation of these rules alone would consume more than half of the memory available for a selectivity-estimation data structure. In addition, processing a term through 5 levels of rules causes more computational effort than acceptable to obtain selectivity estimates. Further, the LATEX hyphenation algorithm is trimmed towards missing some division points rather than dividing terms at incorrect positions. Because we remove suffixes not starting at syllable boundaries, this would result in too many suffixes sorted out. Consequently, we favor faulty hyphenation over missed division points to some extent. In other words, the design goal behind our hyphenation routine is different from the one of the routine used in LATEX.

Because we wanted to exploit syllabification to reduce memory requirements of the suffix tree, we choose a rule-based approach. To minimize the computation effort, we use a very small set of rules. To miss as few division points as possible, our rules are more aggressive than the ones in [5]. The basic idea of our syllabification routine is to determine syllabification points matching regular expressions over the consonant-vowel structure of the word. Function 1 lists the pseudo-code of the function computing the word structure. The output string is constructed by mapping each character to V, in case of a vowel (Line 3), and to *C* otherwise (Line 4). Function 2 contains the pseudo-code of the syllabification routine. First the algorithm checks the word length (Line 1): Words shorter than four characters are left unchanged (e.g., *box*, *cat*).

```
Function 1: computeWordStructure
Input: String word
Output: String wordStructure
1   for (I = 0; i < word.length; i++)
2     if (word[i] is vowel)
3       wordStructure[i] = ’V’;
4     else wordStructure[i] = ’C’;
5   return wordStructure;
```

In English, the number of syllables equals the number of vowel sounds. We assume that a vowel sound corresponds to a sequence of consecutive vowels. We compute the number of vowel sounds (Line 2). If it is less than two, the word is returned (Line 3).

```
Function 2: getSyllables
Input: String word
Output: String syllabifiedWord
1  if (word.length < 4) return word;
2  int vowelSounds = countVowelSounds(word);
3  if (vowelSounds < 2) return word;
4  wordStructure = computeWordStructure(word);
5  if (wordStructure contains consonantBlends) {
6    replace ”CC” with ”DD”;// Check patterns with blends
7    if (wordStructure.match(”VCDDV”))
8      replace ”VCDDV” with “VC-DDV”;
9    if (wordStructure.match(”CVDDV”))
10     replace ”CVDDV” with “CV-DDV”;
11 }
12 // Check common patterns
13 if (wordStructure.match(”VCCV”)
14   replace ”VCCV” with ”VC-CV”;
15 if (wordStructure.match(”VCCCV”)
16   replace ”VCCCV” with ”VC-CCV”;
17 if (wordStructure.match(”CVVCV”)
18   replace ”CVVCV” with ”CVV-CV”;
19
20 sylabifiedWord = getDivisionPoints(wordStructure);
21 return syllabifiedWord;
```

The following step is the construction of the string describing the structure of the word (Line 4). To find syllable boundaries, we test the word structure against 5 basic patterns (Lines 13-19). Table 1 shows the patterns and how we derive syllabification points from them; Table 1a illustrates the process for one word.

Vowel-Consonant-Vowel Structure	Derived Syllabification	Examples
VCV	V-CV	m**o-to**r, l**u-na**r
VCCV	VC-CV	**un-de**r, s**ub-wa**y
VCCCV	VC-CCV	**im-pri**nt, p**il-gri**m
VCCCCV	VCC-CCV	s**and-sto**ne
VCCCCCV	VCC-CCCV	h**igh-stre**et

TABLE 1: The Basic Syllabification Patterns

Step	Word Status	Word	Structure
Input	word	undoubtedly	-
Line 4	word structure	undoubtedly	VCCVVCCVCCV
Line 13	found division point (pattern VCCV)	un-doubtedly	**VC-CV**VCCVCCV
Line 13	found division point (pattern VCCV)	un-doub-tedly	VC-CV**VC-CV**CCV
Line 13	found division point (pattern VCCV)	un-doub-ted-ly	VC-CVVC-C**VC-CV**
Line 20	Transfer division points from structure to word	un-doub-ted-ly	VC-CVVC-CVC-CV

TABLE 1a: Syllabification Algorithm Applied to *undoubtedly*

Consonant blends and digraphs are couples of consonants that are sounded together and therefore should not belong to different syllables. However, the common syllabification pattern VC-CV would incorrectly syllabify words like *migration* to *mig-ra-tion* if we did not take the presence of the consonant blend *gr* into account. Therefore, we define a set of common English blends and bigraphs[2] and check if any of them appear in the word (Line 5). In case of a match, we modify the word structure (Line 6) by replacing the bigraph's consonants CC with DD. We then test the word structure against 14 specifically designed patterns (Lines 7-10), see Table 1b. If none of these patterns matches, we use the five basic patterns described above (Lines 13-19). Finally, we retrieve the division points from the position of the dashes among the word structure string (Line 20) and return the syllabified word.

Vowel-Consonant-Vowel Structure	**Derived Syllabification**	**Examples**
VDDV	V-DDV	f**a-the**r
VCDDV	VC-DDV	**im-pri**nt
VDDCV	VDD-CV	b**ank-ru**pt
VDDCCV	VDD-CCV	-
VCDDCV	VC-DDCV	**un-thri**lled
VCCDDV	VCC-DDV	**off-sho**re
VDDDDV	VDD-DDV	ba**nk-cr**ush
VDDCCCV	VDD-CCCV	
VCDDCCV	VCDD-CCV	
VCCDDCV	VCC-DDCV	
VCCCDDV	VCC-CDDV	**off-spri**ng
VDDDDCV	VDD-DDCV	
VDDCDDV	VDD-CDDV	h**igh-scho**ol
VCDDDDV	VCDD-DDV	w**orth-whi**le

TABLE 1b: The Special Syllabification Patterns

This approach has a few weaknesses. Compound words, for instance, are potential error sources. They should be divided between the words that make them up, but the sole analysis of the word structure cannot tell where the exact division point is located. The word *sandbox*, for instance, will be divided according to the syllabification rule that suggests splitting a sequence of three consonants after the first one (Line 16). This produces the incorrect *san-dbox*.

However, even if the algorithm does not always detect correct division points, we will experimentally verify that these inaccuracies hardly affect the quality of selectivity estimates. In Section 4.3, we will discuss the effect of inaccuracies in more detail.

3.3 Stemming

Morphological variants of the same term (plural, past tense, third person singular, etc.) are responsible for a considerable growth of the data structure. Figure 2 shows how the size of a Syllable CST built on the string *connect* increases when its past tense and continuous forms are included. A common practice to reduce the size of an index is stemming, i.e., conflating inflected terms to their root form. Conflating *connected* and *connecting* with *connect* is reasonable since they convey the same semantic message. Several stemming algorithms have been proposed in literature [9, 16, 14]. Porter's stemmer [16] probably is the most popular one. It adopts a longest-match suffix stripping strategy, through a series of linear steps. Since it is fast and does not require additional storage, we use Porter's algorithm as a preprocessing step. However, the number of stems can be further reduced. Porter's algorithm does not deal with some common suffixes (e.g., *-less, -ution, -ary*, etc.). [9] features a detailed description of errors and wrong conflations made by Porter's stemmer. Furthermore, it does not deal well with compound

[2] In particular, we check the following bigraphs: *bl, cl, dl, fl, pl, gl, br, cr, dr, fr, pr, tr, ch, gh, ph, sh, th, wh, kn, ck*

suffixes, namely suffixes obtained by concatenating more than one suffix. The adverb *increasingly*, for instance, is not stemmed. The reason is that the suffix *–ly* is removed only when it inflects adjectives ending with *-ent* or *-al*. Not conflating *increasingly* with its stem implies the generation of more suffixes, i.e., nodes.

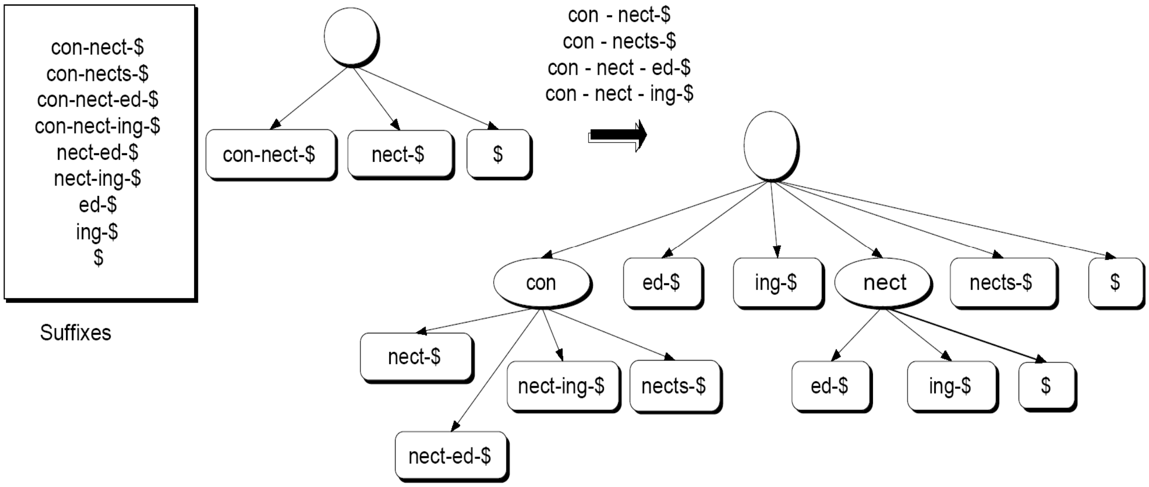


FIGURE 2: Effect of Morphological Variants of the same Word on the Syllable CST's Size

Traditionally, stemming only deals with inflectional or derivational suffixes, but rarely attempts to remove prefixes [11]. However, we observe that removal of prefixes further reduces the number of nodes of a suffix tree. If we conflated *disconnect* with its stem *connect*, we would save the space required by the additional suffix *dis-con-nect*. In Section 3.4, we will illustrate how we manage to reach this size reduction, without losing the information carried by the prefix. We propose a more aggressive stemming routine based on Porter's stemmer that lets us decrease the CST size, without significantly reducing the accuracy of selectivity estimations.

3.4 Stemming Routine

Function 3 lists the pseudo-code of our stemming routine. Our algorithm invokes Porter's algorithm (Line 3), but only as a preprocessing step. It then removes iteratively common English prefixes and suffixes that Porter's stemmer may not have stripped. The algorithm places a condition on the minimum length of the final stem. The affix stripping process stops if no affix is found, or if the removal of one affix would result in a stem shorter than three characters (Line 4).

We divide suffixes in four classes, depending on their inflectional role (adverbial, noun, adjective, verbal), and use an out-of-alphabet symbol to identify each of them (Line 6, suffixCategoryID). Each removed suffix is coded as a string obtained by concatenation of the category identifier and its length (Line 8-9). The reason is that a code is typically shorter than the suffix itself. This reduces the number of bytes to store it. We observe that there are not many suffixes belonging to the same category that can be attached to the same stem, therefore the probability of conflations due to the same code is low. To reduce it further, we use the information on the length. This way, even if *-less* and *-ful* are both adjective suffixes, we can distinguish them based on their length.

In contrast, we do not code prefixes, for the following reason. Dividing prefixes in classes to take advantage of a codified representation is not trivial. Prefixes add a specific connotation to the meaning of the word. Numerical prefixes, for instance *bi-*, *tri-*, give quantitative information; negative prefixes express the opposite meaning of a term. However, it is more difficult to state which message is carried by other prefixes and use it to divide them in classes, as we did with suffixes.

```
Function 3: Stemming algorithm
Input: word, suffixes[4], prefixes
Output: stemmedWord
1   Array suffixCodes; // Store coded suffixes
2   Array prefixes; // Store removed prefixes
3   word = porterStem(word)
4   while(affixFound and word.length > 3) {
5     if (affix is suffix and word.length-affix.length>2)
6     if (affix in any suffixes[]) {
7       // Check suffix category and code suffix
8       suffixCode = suffixCategoryID + suffixLength;
9       suffixCodes.add(suffixCode);
10    } else {
11      prefixes.add(affix);
12    }
13    word = affix; // Affix stripping
14  }
15  stemmedWord = word + prefixes + suffixes;
16  return stemmedWord;
```

By reordering prefixes and suffixes, we can further reduce the size of the CST. The algorithm changes the order of affixes as follows: We syllabify the stem and attach the prefixes and codified suffixes (Line 15).

Example 4. Suppose that adverbial and adjective suffixes are associated with the respective symbols. Given the word *undoubtedly*, Table 2 reports the steps of the algorithm and its output. Figure 3 shows how the number of nodes of the CST built on *un-doubt-ed-ly* decreases thanks to this strategy. We can omit the tree branch generated by the prefix. □

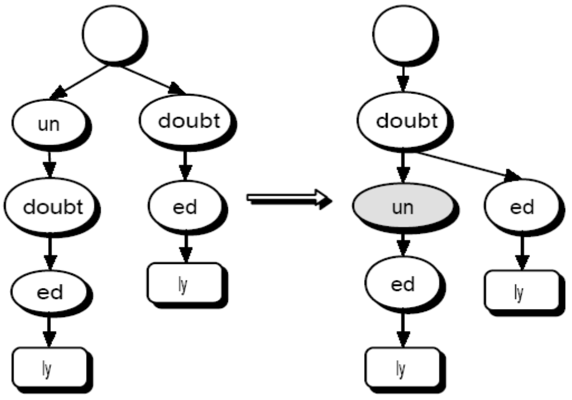


FIGURE 3: Moving the Prefix to the End of the Stem

Moving prefixes behind the stem shows another benefit. In case of pruning, the stem is last to be pruned. This preserves its distinctive semantics as long as possible. We do not move the prefixes behind the suffixes, however, because the prefix carries more semantic meaning than the suffixes and should thus not be pruned before them.

Step	Word Status	Word	Parts Split	
Input	word	undoubtedly		
Line 3	Porter stem	undoubtedli		
Line 6	Suffix found	undoubted	α2	
Line 6	Suffix found	undoubt	α2, β2	
Line 11	Suffix found	doubt	α2, β2	un
Line 14	String stemmed	doubtunα2β2		

TABLE 2: Stemming Algorithm Applied to *undoubtedly*

3.5 Non-Word Detection

Typographical errors are a serious problem regarding CST space requirements. Typos and misspelled words result in undesirable suffixes, i.e., CST nodes. Figure 4 shows how the CST grows if the incorrect term *developement* is added to a CST built on *development*. We observe that we can save space by not inserting mistyped variants of index terms in the CST, but storing them in another way, as explained below. The benefit of non-word detection grows with the number of misspellings in the text collection. This is because the more misspellings are present in the document collection, the more nodes are created while building the CST, similarly to the example in Figure 4. An observation of ours, based on our experiments, is that the probability of identical accidental misspellings is very low. Consequently, the CST nodes created for the suffixes of misspelled terms are relatively useless; Due to their low frequency, they will most likely be pruned after the CST is built. Nevertheless, they require a considerable amount of memory while building the CST, so it is beneficial to exclude misspelled words right away.

A common technique for detecting misspellings is based on n-gram analysis [10]. Given a string, an n-gram is any substring of length n. n-gram analysis requires a set of training words which must be sufficiently representative of the language. From these words, n-grams are extracted and inserted into a table (Dictionary Table). We investigate four strategies to detect non-words. Two of them, Trigram Analysis (TA) and Positional Trigram Analysis (PTA), are based on conventional trigram analysis, the other ones, Syllable Analysis (SA) and Positional Syllable Analysis (PSA), are more recent and are similar to [3].

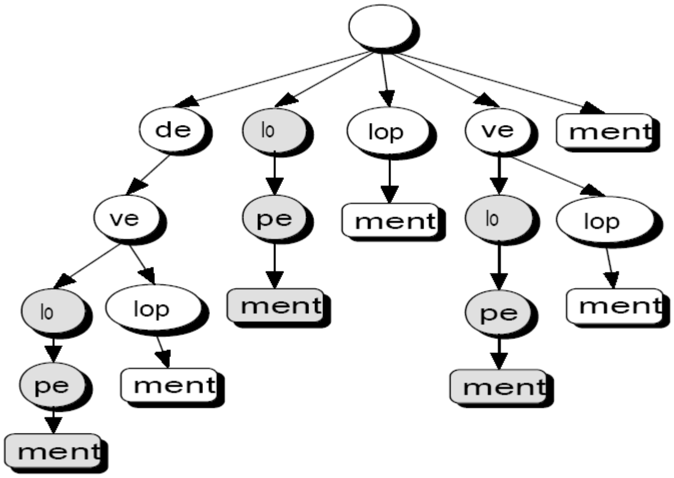


FIGURE 4: Inserting *developement* in the CST

Example 5. Table 3 illustrates which strings will be inserted in the Dictionary Table from the word *inform* according to each strategy. □

Trigram analysis (TA)	inf, nfo, for, orm
Positional trigram analysis (PTA)	inf_0, nfo_1, for_2, orm_3
Syllable analysis (SA)	in, form
Positional syllable analysis (PSA)	in_0, form_1

TABLE 3: N-Grams Generated from the Stem *inform*

In our case, the Dictionary Table is a simple set data structure. With TA, the Dictionary Table contains trigrams. In the positional case (PTA), we attach the information on the position of the trigrams within the words to the trigram string. Syllable analysis (SA) generates non-overlapping syllables, not n-grams; in the positional case (PSA), we store the syllable string together with its position within the word. In order to determine if a given term is a non-word, we extract from it the trigrams (in TA and PTA) or syllables (in SA and PSA). We then look up these parts in the Dictionary Table. We consider a term a non-word if it contains at least one trigram (or syllable, respectively) that is not present in the dictionary. The idea of using syllables to detect non-words is borrowed from [3]. They demonstrate that syllables are superior to state-of-the-art character n-grams, using Indonesian texts. However, the effectiveness of the method on English text is currently unclear. In our evaluation (Section 5), we will therefore test the strength of syllables in detecting non-words on English texts and compare it to trigram analysis.

n-grams characterize the morphological structure of a language well [20]. However, out-of-dictionary n-grams do not necessarily identify a mistyped word. Foreign language words, for instance, show a different morphological structure and could go as errors. Terms like *Albuquerque* or *Afghanistan*, which contain the uncommon trigrams *uqu* and *fgh* respectively, are considered invalid, therefore would not be inserted in the CST, no matter their selectivity. We therefore propose to deal with non-words as follows. We introduce the so-called Invalid N-gram Table, to store invalid n-grams and their selectivity. We use the information contained in this table to estimate the selectivity of non-words, as we will explain in Section 4. Memory requirements of this additional structure are significantly lower than the overhead of storing non-words and their suffixes in the CST. Note that the Dictionary Table does not require any memory after the building phase: It is a temporary data structure for testing the validity of index terms and is discarded after the CST has been completely built.

4. SYLLABLE CST CONSTRUCTION AND SELECTIVITY ESTIMATION

This section describes how we build the Syllable CST and the additional structures used for selectivity estimation. Section 4.1 says how we build the Syllable CST and the Invalid N-gram Table. Section 4.2 presents a new node-labeling strategy, which yields a more concise representation of the CST. Section 4.3 demonstrates how to estimate the selectivity of strings with our model. Finally, Section 4.4 presents selectivity estimation in case of pruning.

4.1 Building the Syllable CST

Function 4 is the procedure that inserts index terms in the Syllable CST (SylCST). Prior to the insertion in the SylCST, every term is decomposed in its trigrams or syllables, according to one of the strategies described in Section 3.5 (Line 1). The Dictionary Table is checked for the presence of each n-gram. If at least one is not in the Dictionary Table (Line 3), it is marked as invalid (Line 4). All invalid n-grams are stored in the Invalid N-gram Table, together with their selectivity (Line 5). The rationale is that we can identify a non-word with its invalid n-grams and use their selectivity to estimate the selectivity of the entire word. This is similar to the data structure [4] refers to as a *q-gram estimator*. As opposed to [4], however, we do not expect severe overestimations. This is because we expect the invalid n-grams to be rather distinctive. In particular, the more characteristic the n-grams of a non-word are, the more accurate is the estimation. The out-of-dictionary trigram *fgh*, for instance, strongly identifies *Afghanistan*, especially in combination with positional analysis. We can reasonably suppose that its selectivity will be close to the one of the word itself. At the expense of little estimation inaccuracies, we can save the space required by suffixes of non-words. This approach also prevents the generation of isolated tree nodes. Non-words suffixes are unlikely to share nodes with English words (e.g., no English word starts with *fghanistan*). Only words that contain no invalid n-grams (valid words) are inserted in the Syllable CST (Line 8). After stemming, they are syllabified, and syllable suffixes are generated (Lines 9-11). Finally, all suffixes are inserted in the SylCST (Line 12). This reduces the size of the CST already during construction. Thus, building the CST becomes less resource-intensive.

```
Function 4: BuildCST
Input: word, dictionaryTable
Output: invalidTable, SylCST
1   wordNGrams = generateNgrams(word);
2   for each nGram in wordNGrams {
3     if (nGram not in dictionaryTable) {
4       validWord = false;
5       invalidTable.add(nGram, wordSelectivity);
6     }
7   }
8   if (validWord = true) {
9     stem = stem(word);
10    syllabifiedStem = syllabify(stem);
11    syllableSuffixes = syllableSuffixes(syllabifiedStem);
12    SylCST.add(syllableSuffixes)
13  }
14  return SylCST, invalidTable;
```

4.2 Node Labeling Strategies

Standard CST construction mechanisms label each node with a single character and then apply path compression [19], i.e., collapsing unary children with their parent node. We refer to this mechanism as Standard Labeling (SL). We introduce a new node labeling strategy. When inserting syllable suffixes in the tree, we label each node with a syllable, instead of a single character, and then apply path compression. We will refer to our labeling mechanism as Atomic Labeling (AL). As we will explain later, this approach yields a more compact representation of the Syllable CST. Since the two structures have the same content, the difference in size is due exclusively to how syllables drive the path-compression mechanism.

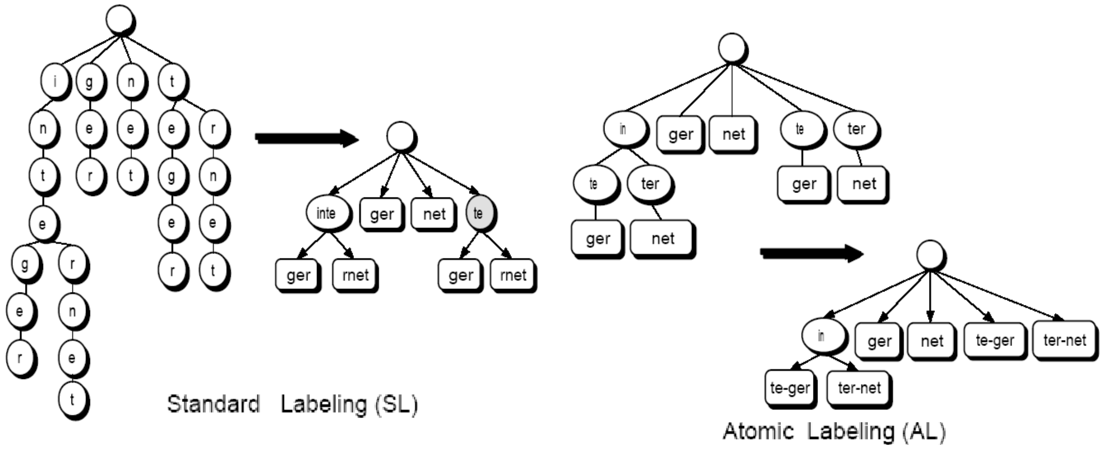


FIGURE 5: Labeling each Node with a Syllable Instead of a Single Character

Example 6. Consider a SylCST that is built over the strings *in-ter-net, in-te-ger*. Standard Labeling ignores internal syllable division points of syllable suffixes and inserts the strings *internet, ternet, net, integer, teger, ger*. Conversely, nodes created by Atomic Labeling store syllables and not single characters. Note that we use our optimization introduced before that uses only syllable suffixes instead of all suffixes, with both Standard Labeling and Atomic Labeling. The result is that path compression produces two different tree structures. Figure 5 illustrates this. □

The SylCST obtained following standard labelling has a small number of nodes in its first level, because the fan-out of a node is at most the alphabet size. On the other hand, it is relatively deep, since the depth is at most the length of the second longest suffix in the CST, and thus has many internal nodes, such as the one labeled with *te*. Conversely, Atomic Labeling produces a Syllable CST which is broader in its first level. The fan-out of the root is equal to the number of syllables in the tree, the fan-out of other nodes is at most this number. This CST is also shallower. This is because the depth is at most the number of syllables in the suffix with the second-most syllables in the CST. Thus, we hypothesize that Atomic Labeling can further reduce CST size, without affecting selectivity estimation. The reason is that, with Atomic Labeling, the paths of suffixes with common prefixes, but different syllabification points (e.g. ***in-te-**ger*, ***in-te**r-net*) split at the start of the first syllable they do not have in common. This leads to internal nodes with a high fan-out. With standard labeling, in contrast, the paths split at the first character they do not have in common, resulting in internal nodes with a lower fan-out. Mathematically, for a fixed number of leaves (each leaf represents one suffix), the number of internal nodes is negatively correlated with the average fan-out of the inner nodes: A lower fan-out results in a deeper tree with more internals nodes, a higher fan-out results in a shallower tree with less internal nodes. Experimental results will confirm this hypothesis.

Clearly, the atomic labeling strategy prevents estimating substrings that do not start and end at syllable borders. However, this is not a problem if queries use keywords as predicates. This is likely for textual data. If other wildcard predicates were frequent, the standard labeling strategy would be advantageous.

4.3 Selectivity Estimation

Once the CST has been built, it can be used for selectivity estimation. The string in question is first decomposed in its n-grams. This is to determine if its structure respects the morphological profile described by n-gram analysis. This means searching for the presence of any of its n-grams in the Invalid N-gram Table. If no match is found, then the string, if present, must have been stored in the CST. The tree is traversed from the root to the node labeled with the string, and its count stores the selectivity sought. Conversely, if the string contains at least an invalid n-gram, then its selectivity estimate is the minimum of the selectivities of its invalid n-grams.

We expect some estimation errors due to syllabification errors. To illustrate, consider again the incorrect syllabification of the word *sandbox* (Section 3.2), which is divided according to the syllabification rule that suggests splitting a sequence of three consonants after the first one. This produces the incorrect *san-dbox*. However, this has no effect at all when estimating the selectivity of the term *sandbox*. But the selectivity of the term *box* will be slightly underestimated. This is because the occurrences of the suffix *box* in the term *sandbox* are not counted due to the incorrect syllabification. However, we do not expect this effect to induce severe errors. This is because we expect the selectivity of a basic (non-compound) word w to be much larger than the selectivity of a specific compound word w is part of. In addition, our experiments will show that, even if the impact of incorrect syllabifications is non-negligible, the benefit of syllabification outweighs it by far.

4.4 Pruning

Since both the Invalid N-gram Table and the Syllable CST still have high memory requirements when built on large text corpora, we cannot do without pruning. We have implemented the common frequency-based pruning strategy. Given the maximum size of a CST (maximum number of nodes), we iteratively remove nodes whose count is under a threshold T. In each iteration, we increase T until the CST has the desired size. To estimate the selectivity of a valid string that is no longer in the PST, we introduce a syllable-based variant of the MO estimator [7]. If the searched string σ is syllabified as σ_A-σ_B-σ_C, its estimated selectivity (ESel) is:

$$\mathrm{ESel} = \mathrm{Sel}(\sigma_{AB}) \cdot \frac{\mathrm{Sel}(\sigma_{BC})}{\mathrm{Sel}(\sigma_B)}$$

where $\sigma_{AB} = \sigma_A\sigma_B$, $\sigma_{BC} = \sigma_B\sigma_C$. If any of the previous terms is not in the CST because it has been pruned, then the selectivity of the string is estimated as the value of the pruning threshold T. Given a non-word, if the Invalid Table has to be pruned as well, and none of its invalid n-grams is found, its selectivity is set to the pruning threshold.

5. EXPERIMENTAL EVALUATION

We evaluate the performance of our Syllable CST. In particular, we first assess how syllabification, atomic node labeling and non-word filtering, reduce the size of the CST. Second, we compare the selectivity-estimation accuracy of the standard CST and the different variants of the Syllable CST. Third, we study how pruning affects the accuracy of selectivity estimates for the different CST types. Finally, we measure the impact of the number of errors in the data; In particular, we wonder if our linguistics-based optimizations still work well in the presence of many errors.

For our experiments, we use four English newswire text corpora, Reuters-21578 (Reuters) [12] and three datasets of the Aquaint Corpus (APW, XIE, NYT) [6]. Since the collections contain data in SGML format, we first parse them to extract text fields only. We then tokenize them to extract single words. Stop words are filtered out, and all terms are converted to lowercase. Table 4 contains statistics on the number of documents, the number of distinct terms, and the size of a complete CST (number of nodes) built on the collection.

	Documents	Tokens	CST size
Reuters	21578	32554	86772
APW	239576	207616	558633
XIE	479433	243932	633899
NYT	314452	352404	979383

TABLE 4: Corpora Statistics

5.1 Effect of Syllabification

The Syllable CST built on the collections significantly reduces memory requirements compared to the standard version. Table 5 shows that the size of the statistics data structure is more than halved. Note that the figures quantify size as the number of nodes in the CST. The memory footprint resulting from the number of nodes is implementation-specific; the currently optimal implementation [22] takes 8.5 KB per node.

	CST's size	SL SylCST	AL SylCST
Reuters	86772	41565 (52.1%)	37298 (57.0%)
APW	558633	308764 (44.7%)	234047 (58.1%)
XIE	633899	307001 (51.6%)	271018 (57.2%)
NYT	979383	526955 (46.2%)	399432 (59.2%)

TABLE 5: CST size (in nodes) and size reduction

Figure 6 shows graphically that a Syllable CST constructed according to the atomic mechanism (AL SylCST, see 4.2) always has smaller memory requirements than the tree built according to the standard algorithm (SL SylCST, see 4.2). This confirms the hypotheses from Section 4.1.

Memory requirements of a CST built on the NYT corpus are 40% of the initial size. This means both that we need less memory to build the tree, and that we can prune the tree at a lower threshold. The latter results in higher estimation accuracy.

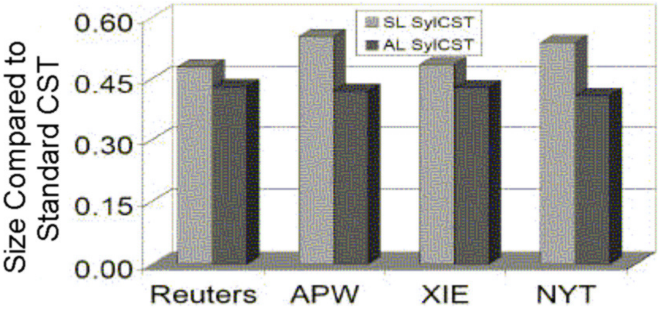


FIGURE 6: Atomic Labeling Yields a Syllable CST with Reduced Memory Requirements

5.2 Effect of N-gram Analysis

n-gram analysis is initialized on a small reference dictionary of common English words (69004 terms, 650 KB). Each dictionary entry is Porter stemmed, and its n-grams are computed according to one of the strategies from Section 3.5 and stored in the Dictionary Table. Each index term is then processed, and out-of-dictionary n-grams are inserted in the Invalid N-gram Table. Table 6 reports the number of entries of each table.

	TA	SA	PTA	PSA
Dictionary	5888	10305	22880	15101
Invalid Table Reuters	3954	9240	11868	11071
Invalid Table APW	6873	39728	41803	51726
Invalid Table XIE	7517	49179	45601	62277
Invalid Table NYT	8421	68914	63951	88623

TABLE 6: Dictionary and Invalid N-gram Table Size

The Invalid Table in turn is retained since we use it to estimate the selectivity of non-words. Table 6 shows that the greater the corpus size, the larger is the Invalid N-gram Table, and its memory requirements can become non-negligible. To limit its size, we set its maximum number of entries to an eighth of the tree size. This is roughly the acceptable size ratio proposed in [4] for the n-gram table. We followed the frequency-based approach proposed in [4] to prune the Invalid Table. I.e., we remove the entries with the lowest frequencies until the n-gram table has at most one eighth the number of entries as the CST has nodes. This increases the estimation error only insignificantly (by less than 0.1%) because the pruning threshold is very low, compared to that of the CST. The reason is that most entries in the Invalid Table are due to misspellings, which rarely have a frequency above 2. Only few entries represent proper names and therefore have a higher frequency. By keeping these high-selectivity n-grams, we can compute better estimates for non-words that turn out to be frequent. This is because of our hypothesis that the selectivity of invalid n-grams is rather strictly related to the selectivity of the words they have been generated from (see Section 4.1).

We evaluate the strength of each strategy (trigram and syllable analysis, with and without considering in-word position) regarding non-word detection and the impact on the size of the statistics data structure. Figure 7 shows that n-gram analysis alone considerably reduces the size of the CST, without syllabification in this experiment. An inspection of the corpora reveals that the size reduction increases with the number of non-words in the corpus: The reduction is highest for the Aquaint XIE corpus, which contains the most non-English words, and lowest for the

Reuters corpus, which is the cleanest. Thus, non-word filtering is particularly beneficial if the data is not very clean.

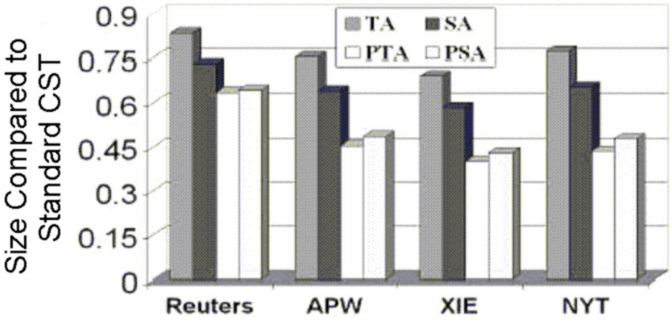


FIGURE 7: Size Reduction due to N-Gram Analysis

		SL SylCST	AL SylCST
Reuters	TA	34538 (-60.2%)	30514 (-64.8%)
	SA	29191 (-66.4%)	25630 (-70.5%)
	PTA	26454 (-69.5%)	23374 (-73.1%)
	PSA	25847 (-70.2%)	22721 (-73.8%)
APW	TA	239898 (-57.1%)	178799 (-68.0%)
	SA	197059 (-64.7%)	147959 (-73.5%)
	PTA	153005 (-72.6%)	110260 (-80.3%)
	PSA	154910 (-72.3%)	113454 (-79.7%)
XIE	TA	216907 (-65.8%)	190864 (-69.9%)
	SA	179375 (-71.7%)	156724 (-75.3%)
	PTA	126221 (-80.1%)	111010 (-82.5%)
	PSA	132886 (-79.0%)	116124 (-81.7%)
NYT	TA	419359 (-57.2%)	313492 (-68.0%)
	SA	340327 (-73.9%)	256467 (-73.8%)
	PTA	255629 (-65.3%)	183295 (-81.3%)
	PSA	261281 (-73.3%)	190746 (-80.5%)

TABLE 7: Syllable CST Size (in Nodes) and Size Reduction

We observe that the positional variants are better at detecting and removing non-words. In particular, positional trigram analysis performs better than the corresponding syllable strategy and also has a smaller Invalid Table (see Table 6). Figure 7 shows that, for the Aquaint corpora, more than half of the size of the CST without non-word filtering is attributable to non-words. The size of the Syllable CST built exclusively over valid words is reported in Table 7. These results demonstrate that syllable analysis is superior to state-of-the-art techniques in the non-positional case. It filters out more words and yields a smaller CST. In the positional case, it does not improve the results obtained with positional trigram analysis.

Figure 8 shows that positional trigram analysis and Atomic Labeling yield a very compact tree. More specifically, the size of CST built over all three Aquaint corpora is reduced to 20% of its initial size using positional trigram analysis to filter out non-words. With Standard Labeling, non-word filtering and syllabification still shrink the CST to at most 35% of its original size (see Table 7).

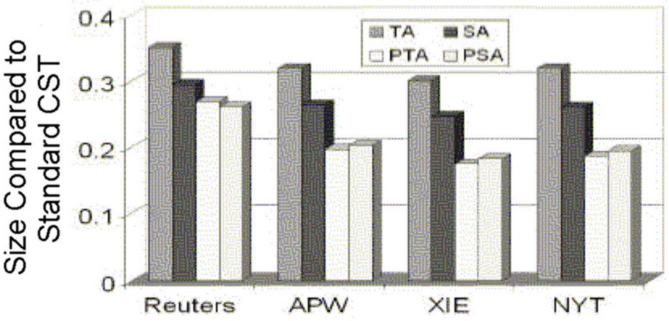


FIGURE 8: Size Reduction Obtained by Building the Syllable CST over Valid Words only

This means that, compared to existing techniques, (a) building the CST requires significantly less memory, and (b) for a given memory size, we can significantly lower the pruning threshold, compared to existing techniques. The latter lets the MO algorithm better estimate the selectivity of pruned suffixes. The experiments in the next sections will demonstrate this.

5.3 Accuracy of Estimations

The following sections report on experimental results on the accuracy of selectivity estimates computed on the Syllable CST. Section 5.3.1 presents the metrics used. We follow the approach adopted in [8, 7, 4] and evaluate positive queries (Section 5.3.2) and negative queries (Section 5.3.3). Positive queries are those that contain terms contained in the corpus, i.e., queries with a selectivity greater than 0, negative queries, in turn, have a 0 selectivity. Finally, Section 5.4 demonstrates that estimation inaccuracies due to pruning are less severe on the Syllable CST.

5.3.1 Evaluation Metrics

For positive queries, we evaluate the accuracy of our estimation model based on the average relative error, as suggested in [4]. It is defined as the ratio:

$$\text{Average Relative Error (RE)} = \frac{|\,\text{ESel} - \text{Sel}\,|}{\text{Sel}'} \qquad \text{(ARE)}$$

where ESel is the estimated selectivity and Sel the real selectivity of a given string. This definition of the average relative error metric includes the correction suggested in [4] to overcome the penalizing effect on low selectivity strings. More specifically, given a corpus of size C, if the actual selectivity of a string is smaller than 100/|C|, then the denominator is set to 100/|C|, formally:
Sel' := max(Sel, 100/|C|)

We consider the quartile distributions introduced by the same authors [4] to show how the accuracy of the estimator is biased. We bucketize the error distribution over the intervals

[-100%,-75%), [-75%,-50%), [-50%,-25%), [-25%,0%), [0,25%), [25%,50%), [50%,75%), [75%,100%), [100%,∞).

Estimates that fall in the interval [0,25%) are exact estimations and small overestimations, whereas the ones that fall in the first four buckets are underestimations.

Following again [4], we use the average absolute error and its percentage of the corpus size as evaluation metric for negative queries.

5.3.2 Positive Queries

Testing the CST against positive queries means estimating the selectivity of strings that are present in the collection. Unless the tree has been pruned, these strings are in the CST. To evaluate the accuracy of our estimator for positive queries, we take the corpus terms and estimate their selectivity as described in Section 4.3. Figure 9 shows the results. The average relative error for the Syllable CST, without introducing n-gram analysis to skip non-words, is minimal for Reuters (3.5%) and maximal for Aquaint NYT, where it is slightly over 10%. The different average errors for the different corpora are due to different average document sizes: The larger the documents, the more terms are frequent, and the higher are the document frequencies of the terms, for the same number of documents. This incurs larger absolute errors (numerator of the relative error) for the same number of documents (a hundredth of it is a lower bound of the denominator of the relative error, cf. Formula ARE) and yields a higher relative error. Average document size is highest in the Aquaint NYT corpus and lowest in the Reuters corpus, Aquaint XIE and APW lie in between. The average relative estimation errors behave accordingly.

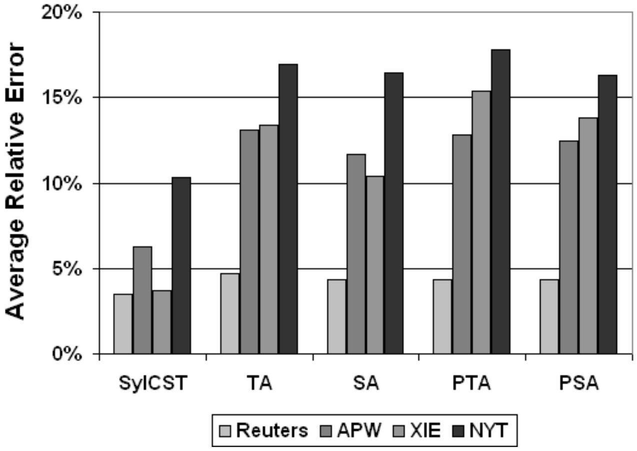


FIGURE 9: Average Relative Error

The experimental results indicate that conflations due to our stemming algorithm do not introduce significant selectivity-estimation errors. The benefits gained from non-word detection in turn come at the cost of some errors: The average relative error for the SylCST without non-word filtering (the leftmost data points in Figure 9) is lower than with any of the non-word filtering strategies enabled (the TA, SA, PTA, and PSA data points). Further, there are more errors with n-gram analysis (the TA and PTA data points) than with syllable analysis (the SA and PSA data points). Overestimations, due to invalid words identified by the same invalid n-gram, penalize the estimation of non-words (see Example 7). However, the average relative error is always under 20% even for Aquaint NYT, which contains the biggest percentage of non-words. For Reuters, errors are almost negligible.

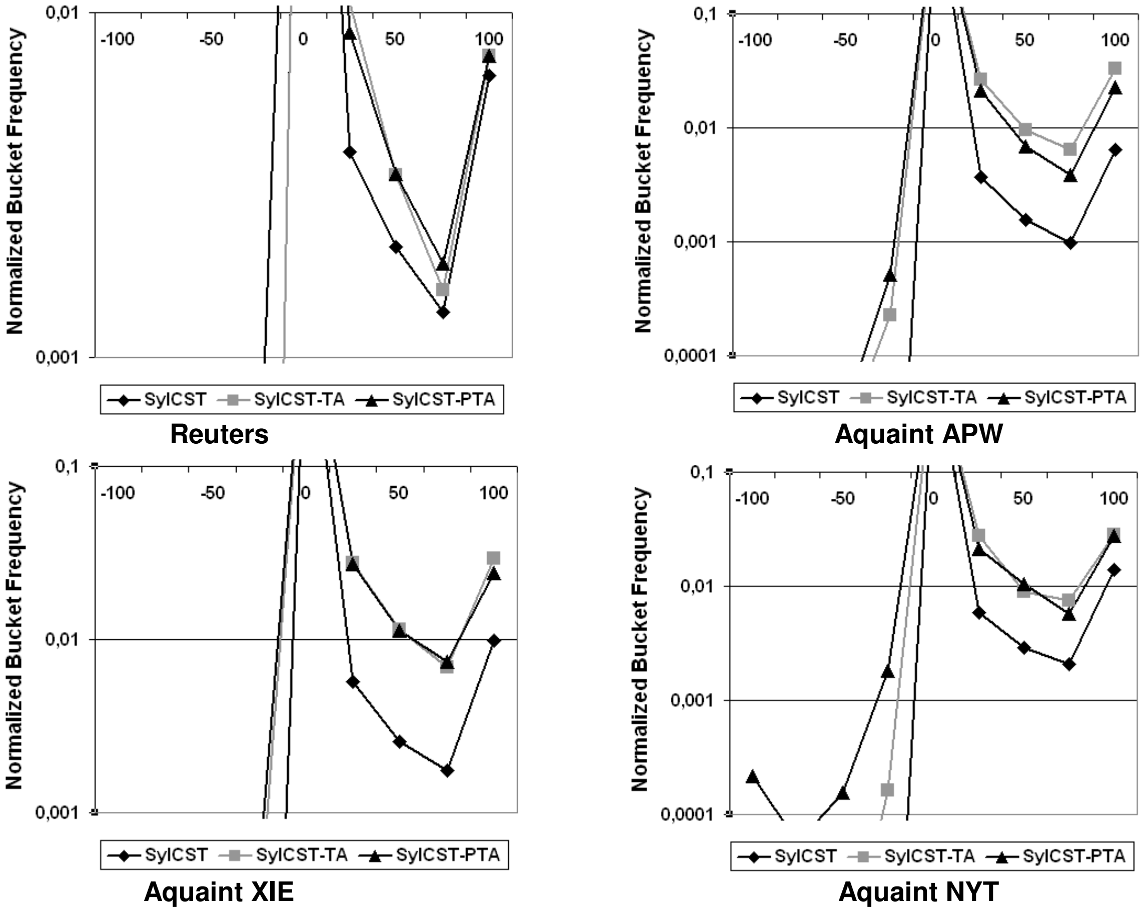
Example 7. Consider the two terms *Alb*uqu*erque* and *Unterba*uqu*erträger* (the latter being a German word from the civil engineering domain). They both will be identified as non-words due to trigram *uqu*. Non-positional trigram analysis conflates these terms in the *uqu* bucket. In consequence, the selectivity of both words is over-estimated as the sum of their individual selectivities. Positional trigram analysis avoids this by taking the in-word position into account: *Albuquerque* belongs to the *uqu_3* bucket, while *Unterbauquerträger* belongs to *uqu_7*. □

Quartile distributions are shown in Figure 10. Each graph refers to a corpus and plots the distribution of estimation errors according to each n-gram strategy. We normalize the absolute frequency to the total number of patterns tested and plot the distribution using a logarithmic scale for the y-axis. A linear scale would not reveal the differences between non-word detection strategies, since the number of overestimations and underestimations is always negligible. All the strategies produce an error that is much lower than 10%. The worst case is Aquaint NYT with non-positional trigram analysis. This is due to two reasons: (a) Non-positional trigram analysis incurs the most errors of all the non-word filtering strategies, and (b) the Aquaint NYT corpus has the largest average document size, which generally results in a higher error rate (see above).

However, 92% of estimations still fall in the central bucket. Non-positional trigram analysis yields less accurate results because it conflates more invalid trigrams (see Example 7). Their number is insufficient to identify non-words, and estimation errors are more likely. The number of estimations in the [100, ∞) bucket is interesting: It shows that if overestimations occur, they are likely to bear a very large error. We think that this effect has to do with the MO algorithm. It overestimates rare combinations of frequent substrings.

5.3.3 Negative Queries

We test our estimation model with negative patterns, which are strings that are not present in the indexed collection. The estimation should return selectivities close to zero. In our experiments, we generate a set of negative strings by randomly introducing errors into corpus words. Table 8 presents the results.



Reuters

Aquaint APW

Aquaint XIE

Aquaint NYT

FIGURE 10: Quartile Distribution of Estimation Accuracy

For Reuters, the error is under 0.02%. We observe that errors tend to become larger the larger the documents in the corpus, for the reasons explained above. However, they remain below 0.15% even with non-positional trigram analysis to filter non-words. This is four times less than the 0.6% worst case reported in [4]. This demonstrates that, even though our model does not return a selectivity of zero, the error induced is not significant.

	Terms	SylCST	TA	SA	PTA	PSA
Reuters	32,554	2.2 (0.01%)	5.5 (0.02%)	4.2 (0.01%)	4.5 (0.01%)	4.0 (0.01%)
APW	207,616	49.5 (0.02%)	185.6 (0.09%)	126.4 (0.06%)	133.4 (0.06%)	123.6 (0.06%)
XIE	243,932	47.1 (0.02%)	349.7 (0.14%)	211.9 (0.09%)	265.5 (0.11%)	237.0 (0.05%)

TABLE 8: Absolute Error and Percentage of Corpus Size for Negative Queries

Corpus		**CST Type / Non-Word Filtering Strategy**					
	CST size (nodes)	**CST**	**CST-TA**	**CST-PTA**	**SylCST**	**SylCST-TA**	**SylCST-PTA**
Reuters	**32,000**	17,4% (7)	18,1% (6)	17,3% (5)	**6,56% (1)**	7,48% (1)	6,93% (0)
	16,000	17,8% (29)	18,2% (27)	17,3% (23)	**6,56% (4)**	7,47% (3)	6,92% (2)
	8,000	19,7% (109)	19,4% (104)	17,9% (97)	**6,56% (14)**	7,46% (12)	6,91% (10)
	4,000	29,4% (332)	26,1% (323)	21,8% (310)	**6,56% (53)**	7,42% (50)	6,87% (46)
APW	**32,000**	12,5% (61)	15,6% (55)	11,1% (44)	**8,89% (8)**	14,8% (6)	13,4% (4)
	16,000	48,3% (214)	42,2% (202)	26,5% (179)	17,7% (31)	21,1% (28)	**16,4% (22)**
	8,000	163,0% (666)	129,0% (645)	80,1% (607)	49,9% (113)	44,6% (106)	**30,0% (95)**
	4,000	452,0% (1726)	352,1% (1688)	227,3% (1635)	148,8% (355)	120,1% (341)	**77,2% (319)**
XIE	**32,000**	4,53% (22)	12,5% (18)	14,3% (14)	**4,59% (3)**	14,0% (2)	16,3% (1)
	16,000	19,0% (90)	21,8% (80)	19,1% (68)	**7,66% (11)**	15,8% (9)	17,0% (6)
	8,000	71,6% (313)	57,3% (293)	38,7% (266)	**20,7% (44)**	24,1% (39)	21,2% (31)
	4,000	216,6% (863)	159,1% (828)	100,2% (779)	62,7% (152)	52,7% (144)	**37,5% (127)**
NYT	**32,000**	27,6% (122)	27,2% (115)	18,7% (101)	**16,1% (16)**	21,0% (14)	19,5% (10)
	16,000	98,4% (413)	81,3% (396)	50,8% (364)	35,6% (65)	35,7% (61)	**27,5% (54)**
	8,000	303,4% (1220)	242,6% (1196)	152,8% (1142)	99,4% (223)	84,8% (215)	**56,5% (199)**
	4,000	833,5% (3016)	659,3% (2981)	413,6% (2917)	281,2% (663)	228,6% (649)	**147,4% (628)**

TABLE 9: Average Relative Errors and Pruning Threshold by CST Size (Best Value of Line in Bold)

5.4 Pruning

Despite all reductions, the Syllable CST for larger corpora still requires too much memory to fit in the data dictionary, see Table 7 for the exact numbers One might think that the memory available to database servers nowadays can easily accommodate the complete CST, and that limits such as the 1KB limit from [8] from 1996 is obsolete. However, not only the amount of physical memory has grown since 1996, but also the number of relations and attributes that database servers must handle. The memory available for the data dictionary has to accommodate statistics for significantly more attributes. Consequently, the memory available for an individual statistics data structure has grown less than the physical memory available in total. In commercial database servers, assuming 1 KB as a limit for the statistics for an individual attribute is not unrealistic [24]. All this means that we cannot do without pruning. Our experiments will show that the Syllable CST can be pruned at a lower threshold, compared to the standard CST, because of its inherently reduced size. As a result, the estimations are significantly more accurate.

We have pruned the CST and the Syllable CST iteratively to meet the same final size of 4,000 nodes. Table 9 contains the average relative error and the respective pruning threshold for each tree size. For readability, we restrict this table to the standard CST and the Syllable CST with standard labeling, and we give only the results for trigram analysis. Appendix A provides the complete results. The Syllable CST is always more accurate than the corresponding standard version when the tree is pruned. The atomic node labeling strategy gives an additional slight advantage.

The SylCST provides good estimation results even with the minimum required size for Reuters: The average relative error is slightly over 40%. In general, the Syllable CST always gives the best estimations. This is due to considerably lower pruning thresholds: The figures show that the value of the pruning threshold decreases by up to 80%, compared to standard CST. This leaves a more accurate basis for the MO algorithm that computes the estimates for pruned strings: The relative estimation error is reduced by up to 70%, compared to the technique from [4].

5.5 Noisy Data

Because the Syllable CST relies on linguistic features of the documents, it is susceptible to misspellings. It is unclear how the Syllable CST performs if the documents contain significantly more errors than a newswire corpus. Such noisy data occurs in the Blogosphere, for instance. To assess the behavior of the Syllable CST in the presence of many errors, we run experiments on documents containing errors. In order to control the error rate at arbitrarily fine granularity and to measure its effect, we use the same corpora as above and introduce artificial errors. To stress the algorithm, we describe experiments in which we have introduced random misspellings in 10% of the terms. The misspellings we introduce are equally distributed among removal, insertion, and replacement of a random character within the term.

The standard CST turns out impossible to build over the complete Aquaint corpora with so many misspellings: The CST grows out of memory due to the suffixes caused by the misspellings. In particular, the standard CST without non-word filtering grows so large that it exceeds the memory limits of a JVM running with 1.5 GB. This is by far more than a database server can allocate for building statistics data structures. The use of another programming language, e.g., C++ instead of Java, and a highly memory-optimized CST implementation might mitigate this problem, but will not do away with it. Thus, for this series of experiments, we only use the first 50,000 documents of each of our test corpora. Table 10 shows the results of a comparison between the standard CST and the Syllable CST with standard node labeling, each without non-word filtering and with non-positional and positional trigram analysis.

The numbers show that, somewhat expected, the benefit of n-gram analysis is very high when the data contains many misspellings: Positional trigram analysis reduces the size of the un-pruned CST by about 65%, i.e., 65% less memory is required to build the CST. Further, the results show that misspellings do not affect the benefit of syllabification to a significant degree: Syllabification still reduces the average relative error of selectivity estimates by about 50% in most of the cases. Only the error of the 4,000 node SylCST for the XIE branch is in the order of magnitude of the corresponding standard CST. Furthermore, n-gram analysis affects the average relative estimation error only negligibly.

In general, the average relative estimation error shows the same tendencies as for the complete Aquaint corpora without artificial misspellings: (a) Both syllabification and non-word filtering reduce the size of the un-pruned CST, (b) syllabification reduces the pruning threshold for the pruned CST, (c) due to (b), syllabification improves selectivity-estimation accuracy significantly, (d) non-word filtering can incur some estimation errors, but does not decrease overall accuracy by much. However, the advantage of the Syllable CST over the standard CST is higher for the 16,000 and 8,000 node trees, and lower for the 4,000 node tree. We attribute this effect to local skews in the distributions of the document frequencies of the terms. With a larger number of documents, as in the previous experiments, those skews level out and yield a rather predictable development of the relative estimation error. Conversely, these skews have a more significant effect in this current experiment, due to the considerably smaller number of documents.

Corpus	CST Type	CST Size	Average relative error at different CST sizes (pruning threshold)		
			16000	8000	4000
APW	CST	1710729	137.9% (1004)	219.3% (2405)	392.3% (4847)
	CST-TA	921349	125.6% (904)	186.7% (2231)	303.9% (4607)
	CST-PTA	491327	113.7% (795)	174.5% (2058)	263.0% (4389)
	SylCST	807641	28.4% (273)	65.8% (774)	201.7% (1887)
	SylCST-TA	449642	28.0% (219)	64.5% (677)	171.8% (1753)
	SylCST-PTA	227461	26.4% (156)	68.9% (572)	189.0% (1598)
XIE	CST	1065209	146.1% (522)	237.0% (1355)	419.1% (2897)
	CST-TA	586189	134.1% (464)	193.9% (1277)	310.3% (2759)
	CST-PTA	322911	127.5% (385)	178.2% (1169)	274.4% (2648)
	SylCST	493905	34.1% (116)	126.5% (378)	401.4% (1013)
	SylCST-TA	282024	26.5% (87)	86.3% (320)	225.1% (947)
	SylCST-PTA	148285	26.9% (56)	90.6% (252)	205.9% (854)
NYT	CST	2649161	125.2% (1925)	203.9% (4462)	362.5% (8482)
	CST-TA	1436417	118.4% (1799)	182.9% (4282)	310.7% (8212)
	CST-PTA	762014	109.2% (1655)	167.9% (4079)	283.2% (7908)
	SylCST	1261474	30.8% (558)	51.8% (1534)	115.2% (3688)
	SylCST-TA	704858	34.0% (486)	57.5% (1408)	130.7% (3505)
	SylCST-PTA	354332	34.4% (390)	54.6% (1272)	143.9% (3356)

TABLE 10: Experiment Results with Noisy Data

6. CONCLUSIONS

Estimating the selectivity of query terms is essential for query optimization and in other contexts. The estimates have to be available before the actual query processing and need to be based on small summary statistics. The memory limitations result from the need to permanently hold the statistics used for query optimization in physical main memory. If query optimization caused only a single page fault (i.e., the need to swap a memory page from on-disk virtual memory back into physical main memory), this would annihilate the performance advantage a database system gains from optimizing query execution.

Selectivity estimation for string predicates frequently relies on Count Suffix Trees (CST) [4, 7, 8]. While they provide good estimates, their storage requirements are prohibitively high. Pruning tries to solve this problem, by trading estimation accuracy for reduced memory needs. So far, pruning strategies are mostly based on frequency and tree depth. In this paper, we have proposed new techniques that reduce the size of CST over natural-language texts. We exclude suffixes that do not make sense from a linguistic point of view, regardless of their frequency. Syllabification has proven to be a suitable tool for generating suffixes that carry an enhanced semantic message, compared to letter-wise suffixes. A more aggressive stemming routine lets us further reduce the CST size, without affecting the quality of selectivity estimates by much. Further, a very concise n-gram data structure allows (a) for filtering out non-words during CST construction already, and (b) for estimating their selectivity precisely.

The various filtering techniques described here are mutually independent. They are applicable to other languages as well, provided that there is a stemming procedure, a syllabification routine, or a dictionary of the language for the n-gram filtering. Since all the filtering takes place during CST construction, significantly less memory is required to build the CST. The combination of these approaches, together with a new node labeling strategy, yields a much more compact CST: For English text, estimation accuracy is the same as with a classical CST, with only 20-30% of the nodes. From another perspective, with the same number of nodes, the new techniques reduce the average estimation error by up to 70%.

7. REFERENCES

[1] J. Bae and S. Lee. "Substring count estimation in extremely long strings". IEICE – Transactions in Information Systems, E89-D(3):1148–1156, 2006.

[2] R. Baeza-Yates and B. Ribeiro-Neto. "Modern information retrieval". Addison-Wesley Longman, 1. print. edition, 1999.

[3] S. Bressan and R. Irawan. "Morphologic non-word error detection". In Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA '04), pages 31–35, 2004.

[4] S. Chaudhuri, V. Ganti, and L. Gravano. "Selectivity estimation for string predicates: Overcoming the underestimation problem". In Proceedings of ICDE 2004, Boston, MA, USA, 2004.

[5] D. W. Cummings. "American English spelling: an informal description". Johns Hopkins University Press, 1988.

[6] D. Graff. "The Aquaint corpus of english news text". Linguistic Data Consortium, Philadelphia, 2002.

[7] H. Jagadish, O. Kapitskaia, and D. Srivastava. "One-dimensional and multi-dimensional substring selectivity estimation". The International Journal on Very Large Data Bases, 9(3): 214–230, 2000.

[8] P. Krishnan, J. S. Vitter, and B. Iyer. "Estimating alphanumeric selectivity in the presence of wildcards". In ACM SIGMOD International Conference on Management of Data, pages 12–13. ACM, 1996.

[9] R. Krovetz. "Viewing morphology as an inference process". In Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 191–203, 1993.

[10] K. Kukich. "Techniques for automatically correcting words in text". ACM Computer Surveys, 24:379–439, 1992.

[11] M. Lennon, D. Pierce, B. Tarry, and P. Willett. "An evaluation of some conflation algorithms for information retrieval". Journal of Information Science, 3(177–183), 1981.

[12] D. D. Lewis. Reuters-21578 [online] Available at: http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[13] F. M. Lian. "Word hy-phen-a-tion by com-put-er". PhD thesis, Stanford University, Stanford, August 1983.

[14] J. B. Lovins. "Development of a stemming algorithm". Mechanical Translation and Computational Linguistics, 11:22–31, 1968.

[15] E. M. McCreight. "A space-economical suffix tree construction algorithm". J. Assoc. Comput. Mach., 23(2): 262–272, 1976.

[16] M. F. Porter. "An algorithm for suffix stripping". Program, 14(3):130–137, 1980.

[17] Y. Tian, S. Tata, R. A. Hankins, and J. M. Patel. "Practical methods for constructing suffix trees". The International Journal on Very Large Data Bases, 14(3): 281–299, 2005.

[18] E. Ukkonen. “On-line construction of suffix trees”. Algorithmica, 14(3): 249–260, 1995.

[19] P. Weiner. “Linear pattern matching algorithms”. In Proceedings of the 14th Annual Symposium on Switching and Automata Theory, pages 1–11, 1973.

[20] E. M. Zamora, J. Pollock, and A. Zamora. “The use of trigram analysis for spelling error detection”. Information of Processing and Management, 17: 305–316, 1981.

[21] Z. Chen, F. Korn, N. Koudas, S. Muthukrishnan. “Selectivity Estimation for Boolean Queries”. In Proceedings of PODS 2000, Dallas, TX, USA, 2000

[22] R. Giegerich, S. Kurtz, J. Stoye. “Efficient Implementation of Lazy Suffix Trees”. Software: Practice and Experience, Volume 33, No 11, John Wiley & Sons Ltd., 2003

[23] G. Sautter, C. Abba, K. Böhm. “Improved Count Suffix Trees for Natural Language Data”. In Proceedings of IDEAS 2008, Coimbra, Portugal, 2008

[24] Personal communication with Torsten Grabs, Microsoft SQL Server development team.

8. APPENDIX

Corpus / CST Type	**CST Size (in Nodes)**			
Reuters	**32000**	**16000**	**8000**	**4000**
CST	17,4% (7)	17,8% (29)	19,7% (109)	29,4% (332)
CST - TA	18,1% (6)	18,2% (27)	19,4% (104)	26,1% (323)
CST - SA	17,5% (5)	17,5% (24)	18,1% (98)	22,3% (306)
CST - PTA	17,3% (5)	17,3% (23)	17,9% (97)	21,8% (310)
CST - PSA	17,2% (4)	17,3% (22)	17,7% (96)	21,1% (301)
SL SylCST	6,56% (1)	6,56% (4)	6,56% (14)	6,56% (53)
SL SylCST - TA	7,48% (1)	7,47% (3)	7,46% (12)	7,42% (50)
SL SylCST - SA	7,07% (0)	7,10% (2)	7,12% (11)	7,14% (47)
SL SylCST - PTA	6,93% (0)	6,92% (2)	6,91% (10)	6,87% (46)
SL SylCST - PSA	6,15% (0)	7,03% (2)	7,06% (10)	7,09% (45)
AL SylCST	6,73% (1)	7,04% (4)	8,11% (14)	11,1% (54)
AL SylCST - TA	7,91% (1)	7,91% (3)	8,65% (12)	10,7% (50)
AL SylCST - SA	6,22% (0)	7,41% (2)	7,83% (11)	9,45% (48)
AL SylCST - PTA	6,12% (0)	7,40% (2)	7,71% (10)	9,30% (46)
AL SylCST - PSA	6,14% (0)	7,32% (2)	7,69% (10)	8,95% (46)

Appendix A1: Average Relative Errors and Pruning Threshold by CST Size with Reuters

Corpus / CST Type	**CST Size (in Nodes)**			
APW	**32000**	**16000**	**8000**	**4000**
CST	12,5% (61)	48,3% (214)	163,0% (666)	452,0% (1726)
CST - TA	15,6% (55)	42,2% (202)	129,0% (645)	352,1% (1688)
CST - SA	11,8% (48)	31,3% (187)	96,9% (619)	274,5% (1645)
CST - PTA	11,1% (44)	26,5% (179)	80,1% (607)	227,3% (1635)
CST - PSA	10,9% (43)	25,8% (175)	77,5% (598)	218,7% (1608)
SL SylCST	8,89% (8)	17,7% (31)	49,9% (113)	148,8% (355)
SL SylCST - TA	14,8% (6)	21,1% (28)	44,6% (106)	120,1% (341)
SL SylCST - SA	12,7% (5)	17,0% (24)	34,2% (99)	91,8% (329)
SL SylCST - PTA	13,4% (4)	16,4% (22)	30,0% (95)	77,2% (319)
SL SylCST - PSA	13,1% (4)	16,1% (21)	29,1% (92)	74,6% (316)
AL SylCST	7,96% (8)	13,8% (31)	35,8% (113)	105,0% (355)
AL SylCST - TA	14,2% (6)	18,4% (28)	34,6% (106)	86,8% (341)
AL SylCST - SA	12,3% (5)	15,2% (24)	27,2% (99)	67,2% (329)
AL SylCST - PTA	13,1% (4)	15,1% (22)	24,3% (95)	56,6% (319)
AL SylCST - PSA	12,9% (4)	14,9% (21)	23,8% (92)	55,2% (316)

Appendix A2: Average Relative Errors and Pruning Threshold by CST Size with Aquaint APW

Corpus / CST Type	CST Size (in Nodes)			
XIE	**32000**	**16000**	**8000**	**4000**
CST	4,53% (22)	19,0% (90)	71,6% (313)	216,6% (863)
CST - TA	12,5% (18)	21,8% (80)	57,3% (293)	159,1% (828)
CST - SA	8,91% (16)	15,5% (73)	41,6% (276)	120,1% (789)
CST - PTA	14,3% (14)	19,1% (68)	38,7% (266)	100,2% (779)
CST - PSA	13,4% (13)	18,2% (66)	37,4% (258)	97,3% (761)
SL SylCST	4,59% (3)	7,66% (11)	20,7% (44)	62,7% (152)
SL SylCST - TA	14,0% (2)	15,8% (9)	24,1% (39)	52,7% (144)
SL SylCST - SA	10,8% (1)	12,1% (7)	17,9% (33)	39,4% (132)
SL SylCST - PTA	16,3% (1)	17,0% (6)	21,2% (31)	37,5% (127)
SL SylCST - PSA	15,3% (1)	16,2% (5)	20,3% (29)	36,2% (123)
AL SylCST	4,26% (3)	6,28% (10)	14,9% (43)	44,4% (150)
AL SylCST - TA	13,8% (2)	15,0% (9)	20,4% (38)	40,4% (140)
AL SylCST - SA	10,7% (1)	11,5% (7)	15,4% (34)	30,3% (134)
AL SylCST - PTA	16,2% (1)	16,6% (6)	19,3% (31)	30,5% (129)
AL SylCST - PSA	15,2% (1)	15,8% (6)	18,6% (30)	29,6% (126)

Appendix A3: Average Relative Errors and Pruning Threshold by CST Size with Aquaint XIE

Corpus / CST Type	CST Size (in Nodes)			
NYT	**32000**	**16000**	**8000**	**4000**
CST	27,6% (122)	98,4% (413)	303,4% (1220)	833,5% (3016)
CST - TA	27,2% (115)	81,3% (396)	242,6% (1196)	659,3% (2981)
CST - SA	20,8% (105)	61,0% (371)	185,7% (1146)	502,1% (2910)
CST - PTA	18,7% (101)	50,8% (364)	152,8% (1142)	413,6% (2917)
CST - PSA	16,9% (98)	47,8% (358)	146,7% (1121)	398,2% (2857)
SL SylCST	16,1% (16)	35,6% (65)	99,4% (223)	281,2% (663)
SL SylCST - TA	21,0% (14)	35,7% (61)	84,8% (215)	228,6% (649)
SL SylCST - SA	19,1% (11)	29,4% (56)	65,6% (202)	176,0% (636)
SL SylCST - PTA	19,5% (10)	27,5% (54)	56,5% (199)	147,4% (628)
SL SylCST - PSA	18,0% (10)	25,6% (52)	53,1% (194)	140,8% (619)
AL SylCST	14,1% (16)	27,6% (65)	71,5% (223)	199,4% (663)
AL SylCST - TA	19,6% (14)	29,8% (61)	63,4% (215)	162,9% (649)
AL SylCST - SA	18,1% (11)	25,3% (56)	50,6% (202)	127,0% (636)
AL SylCST - PTA	18,8% (10)	24,2% (54)	44,3% (199)	105,8% (628)
AL SylCST - PSA	17,3% (10)	22,6% (52)	42,0% (194)	102,0% (619)

Appendix A4: Average Relative Errors and Pruning Threshold by CST Sizes with Aquaint NYT