Improved Count Suffix Trees for Natural Language Data

Guido Sautter
Universität Karlsruhe (TH)
Am Fasanengarten 5
76128 Karlsruhe
+49-721-608-4066
sautter@ipd.uka.de

Cristina Abba
Reply S.p.a.
Corso Francia, 110
10143 TORINO
cristina.abba@gmail.com

Klemens Böhm
Universität Karlsruhe (TH)
Am Fasanengarten 5
76128 Karlsruhe
boehm@ipd.uka.de

ABSTRACT

With more and more natural language text stored in databases, handling respective query predicates becomes very important. Optimizing queries with predicates includes (sub)string estimation, i.e., estimating the selectivity of query terms based on small summary statistics before query execution. Count Suffix Trees (CST) are commonly used to this end. While CST yield good estimates, they are expensive to build and require a large amount of memory to be stored. To fit in the data dictionary of database systems, they have to be severely pruned. Existing pruning techniques are based on suffix frequency or tree depth. In this paper, we propose new filtering and pruning techniques that reduce both the size of CST over natural-language texts and the cost of building them. The core idea is to exploit features of the natural language data, i.e., regarding only the suffixes that are useful in a linguistic sense. The most important innovations are (a) a new aggressive approximate syllabification technique to filter out suffixes, (b) a new affix and prefix stripping procedure that conflates more terms than conventional stemming techniques, (c) the deployment of state-of-the-art trigram techniques and a new syllable-based mechanism to filter out non-words (i.e., misspellings and other language anomalies like foreign words), which would cause an over-proportional growth of the CST otherwise. – Our evaluation with large English text corpora shows that our new mechanisms in combination decrease the size of a CST by up to 80% and shorten the build phase significantly. From a different perspective, if storage space remains unchanged, the accuracy of selectivity estimates computed from the CST increases by up to 70%.

General Terms

Algorithms, Measurement, Experimentation, Languages, Theory

Keywords

Query Optimization, Selectivity Estimation, Text Data, Count Suffix Tree, Pruning

IDEAS08 2008, September 10-12, Coimbra [Portugal]

Editor: Bipin C. DESAI

1. INTRODUCTION

With more and more natural language data stored in databases, query processing for this data type becomes highly important. To optimize such queries, the (sub)string estimation problem is essential, i.e., estimating the selectivity of natural language query predicates (usually terms) based on small summary statistics before the actual query execution. The selectivity of a term (or substring) is the number of documents in the underlying collection containing it. Count Suffix Trees (CST) are commonly used for this estimation. According to [7], each CST node V stores the selectivity of the string corresponding to the path from the root to V, retrievable in a time linear to the string length. Therefore, CST built over text data can efficiently solve the selectivity-estimation problem.

However, CST have high memory requirements and are expensive to build. The space complexity of a CST is proportional to the number of strings stored in it. A CST built over a large amount of text data may well exceed 1,000,000 nodes, i.e., 8.5 MB in the currently optimal implementation [20]. Since the statistics used by query optimizers have to fit in the data dictionary (a very limited amount of memory), CST used for query optimization need to be reduced in size [3]. To make the tree meet memory requirements, a common solution is pruning, i.e., discarding some nodes to save space, e.g., the ones with the lowest selectivities [6, 7]. But this also affects estimation accuracy: The selectivities of strings not present in the Pruned CST (PST) any more have to be estimated using algorithms like KVI or MO [7, 6]. This incurs considerable estimation inaccuracies. Pruning becomes even more problematic with non-static document collections, e.g., forums or the Blogosphere, because it makes updates impossible [1]. The only solution currently known is to rebuild the CST over the updated collection. Even though algorithms exist that reduce space and time complexity [17, 18], CST construction remains expensive.

The goal of our work is to find other ways of reducing the CST size, i.e., filtering out suffixes. We focus on natural-language texts. Our core idea is to find linguistic criteria that decide if a string or suffix is likely to be queried, prior to insertion in the CST. We insert only the suffixes that are likely to be queried and deal with the rest separately. This reduces the size of the data structures during construction already, before the actual pruning. In particular, we apply syllabification, stemming, and non-word detection. The combination of these mechanisms yields a CST that requires significantly less memory than state-of-the-art ones. More specifically, our contributions are as follows: (1) Because suffixes that do not start at a syllable border carry little semantic

meaning, we filter out these suffixes using a fast approximate syllabification routine based on morphological structure of words. To avoid filtering too many suffixes, our routine identifies syllable boundaries more aggressively than conventional ones. (2) Stemming, i.e., conflating different inflections of a term to the same root form, reduces the number of suffixes. Traditional stemming algorithms, like Porter's stemmer [14], are rather conservative, i.e., omit conflations to avoid errors. We propose a new, more aggressive stemming procedure, which conflates more terms and thus reduces the number of suffixes to store. Though linguistic errors may occur, we show that their effect on estimation quality is likely to be insignificant. (3) Non-words and foreign words incur an over-proportional number of nodes in the CST. We therefore deploy a q-gram based non-word detection algorithm to prevent inserting non-words in the CST. To estimate the selectivity of non-words, we use a variant of the q-gram estimator [3] instead. The combination of our mechanisms reduces the size of the CST by up to 70% without significantly affecting estimation accuracy. With the same tree size on the other hand, it reduces the relative estimation error by up to 80%.

Paper outline: Section 2 reviews related work. Section 3 describes the design of the Syllable CST, Section 4 the estimation model based on the Syllable CST. Section 5 features our evaluation. Section 6 concludes.

2. RELATED WORK

The Count Suffix Tree (CST) [7] is the data structure commonly used to estimate the selectivity of string predicates. Given a collection of documents, the CST stores all terms and their suffixes. Each node represents a suffix and has a counter that stores the number of occurrences in the collection. Since CST built over large text datasets are huge, pruning strategies are essential to keep it in memory. Pruning requires estimating the selectivity of the terms whose nodes have been discarded. [7] proposes three estimation methods. Among these, the KVI algorithm is the most accurate one. The MO (Maximal Overlap) algorithm [6] outperforms KVI. It parses the searched pattern in overlapping (when existing) substrings, which are considered to be statistically dependent. Since both KVI and MO tend to underestimate selectivities, [3] proposes a new estimation model based on q-gram tables and a regression tree. [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). – We refer to further related work in Sections 3 and 4, in particular to computational linguistics algorithms, which we adapt and deploy in our context.

3. SYLLABLE COUNT SUFFIX TREE

This section proposes a new variant of the CST, the Syllable CST, suited for selectivity estimation on natural-language data. Section 3.1 explains how syllabification reduces the tree size. Section 3.2 introduces a new aggressive stemming routine. Section 3.3 says how we deal with non-words.

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 \$.

According to the definition of the suffix tree [17], inserting an index term in the tree implies inserting all of its suffixes as well. Let s be a string of length n over an alphabet $S \cup \$$ (\$ is the termination symbol, $\$ \notin S$, $\$ > s \; \forall \; s \in S$). Example: Given s = *information$*, its suffixes are: *information$*, *nformation$*, *formation$*, *ormation$*, *rmation$*, *mation$*, *ation$*, *tion$*, *ion$*, *on$*, *n$*, and *$*. Some of these suffixes are unlikely to be ever queried by a user, e.g., *-rmation$*. Syllables are natural word building blocks in many languages. Using syllabification points to compute suffixes leaves only those carrying an enhanced semantic meaning. Example: Given the syllabified string s = *in-for-ma-tion-$*, its syllable suffixes are *in-for-ma-tion-$*, *for-ma-tion-$*, *ma-tion-$*, *tion-$*, *$*.

The Syllabification Routine. The problem of syllabification is strictly tied to the hyphenation task [12]. Syllabification algorithms can be rule-based or dictionary-based. The latter ones look up the syllable division points in a dictionary. But a dictionary would be too large for our purpose, and, no matter its actual size, will never contain all words (foreign language words, proper names, etc.). Rule-based hyphenation systems (e.g., the one in LATEX [12]) are typically faster and require less memory, but are inherently error prone. Most of the rules are based on the sound of the spoken word and are not easy to implement, e.g., the VV rule [4]. Since our goal is exploiting syllabification to reduce the size of the CST, we nevertheless adopt a rule-based solution.

The hyphenation routine of LATEX [12] is not applicable to our problem right away, however. The representation of its 5,000 rules alone would consume half of the memory available for the CST, and applying them all causes too much computational effort. In addition, the LATEX hyphenation algorithm favors missing some division points rather than erroneously dividing terms. We, on the other hand, prefer faulty hyphenation to missed division points to some extent. This is because we remove suffixes not starting at syllable boundaries, and missing division points would sort out too many suffixes.

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 [4]. The core idea behind our syllabification routine is to determine syllabification points matching regular expressions over the consonant-vowel structure of the word. In particular, we split blocks of consonants between two vowels in the middle (even number of consonants), or right before the middle (odd number of consonants). To prevent dividing consonant blends and digraphs (couples of consonants that sound together, e.g., *th*), we use exception rules. Finally, we do not syllabify words shorter than four characters (e.g., *box*, *cat*). The word *information*, for instance, would be syllabified like this: *information* → *VCCVCCVCVVC* → *VC-CVC-CV-CVVC* → *in-for-ma-tion*.

Discussion. This approach is not limited to English; it is applicable to any character-based language, provided that there is a syllabification routine. Clearly, syllable-based filtering affects selectivity estimation for substrings that do not start at syllable boundaries: A Syllable CST on the word *information$* would not provide a selectivity estimate for the predicate *LIKE '%nfo%'*. This is not a severe drawback: Queries over natural language text are very likely to contain "natural" text fragments, e.g., *LIKE '%info%'* or *LIKE '%inform%'*, as opposed to '%nfo%'. Traces of web-search engines confirm this [21].

Compound words are a potential source of errors: They should be divided between the words they consist of, but the sole analysis of the word structure cannot locate the exact division point. The word *sandbox*, for instance, will be erroneously divided into *san-dbox*. Our experiments however show that these inaccuracies do not affect estimation accuracy by much.

3.2 Stemming

Morphological variants of the same term (plural, past tense, third person singular, etc.) let the CST grow considerably. A Syllable CST built on the string 'connect', for instance, has 4 nodes. It increases to 14 nodes when past tense and continuous forms are included. Stemming conflates inflected terms to their root form and thus reduces the number of suffixes. Conflating *connected* and *connecting* with *connect* is reasonable since they convey the same semantic message. Several stemming algorithms have been proposed in literature [8, 14, 13]; Porter's stemmer [14] probably is most popular. 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.). [8] features a detailed description of errors and wrong conflations. Furthermore, it does not deal well with compound suffixes. The adverb *increasingly*, for instance, is not stemmed because the suffix *–ly* is removed only when it inflects adjectives ending with *-ent* or *-al*. Not conflating *increasingly* with its stem incurs more suffixes, i.e., nodes.

Stemming Routine. Our algorithm invokes Porter's algorithm as a preprocessing step. It then removes common English suffixes that may not have been stripped by Porter iteratively, until it finds no more suffixes, or the rest of the term would be shorter than three characters.

Prefix Stripping. Traditionally, stemming only deals with inflectional or derivational suffixes, but rarely attempts to remove prefixes [10]. However, we observe that removal of prefixes would further reduce the number of nodes of a suffix tree. If we conflate *disconnect* with its stem *connect*, we save the space required by the additional suffix *dis-con-nect*. However, removing prefixes would incur a significant loss of information because they add a specific connotation to the meaning of the word. Instead, we move prefixes behind the stem, which further reduces the size of the CST. Example: From the word *undoubtedly*, the algorithm would produce *doubt-un-ed-ly*. The number of nodes of a CST built on *un-doubt-ed-ly* decreases from 8 to 7 thanks to this strategy: We can omit the tree branch generated by the prefix *un*. Moving prefixes behind the stem shows another benefit: In case of pruning, the stem is last to be pruned, preserving its distinctive semantics as long as possible.

3.3 Non-Word Detection

Typographical errors are a serious problem. They result in undesirable suffixes, i.e., CST nodes. The CST built on the sting *development*, for instance, has 12 nodes; adding the incorrect term *develop**e**ment* inflates it to 21 nodes. Thus, not inserting mistyped variants of index terms in the CST saves space. According to preliminary experiments, the benefit of non-word detection grows with the amount of noise in the text: The more misspellings, the more nodes there are. As long as misspellings are not repeated, these nodes are useless because they are pruned after the CST is built. Thus, it is beneficial to exclude them right away.

A common technique for detecting misspellings is n-gram analysis [10]. 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 techniques to detect non-words. Two of them, Trigram Analysis (TA) and Positional Trigram Analysis (PTA), use conventional trigram analysis, the other two, Syllable Analysis (SA) and Positional Syllable Analysis (PSA), are more recent and are similar to [2]. In order to determine if a given term is a non-word, we extract its trigrams (in TA, PTA) or syllables (in SA, PSA), and 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 table. Table 1 illustrates which strings are inserted in the dictionary table for the word *inform* according to each strategy.

Table 1: n-grams generated from the stem *inform*

Trigram analysis (TA; n=3)	Positional trigram analysis (PTA; n=3)	Syllable analysis (SA)	Positional syllable analysis (PSA)
inf, nfo, for, orm	inf_0, nfo_1, for_2, orm_3	in, form	in_0, form_1

N-grams characterize the morphological structure of a language well [18]. 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 such as *Albuquerque* or *Afghanistan*, which contain the uncommon trigrams *uqu* and *fgh* respectively, are considered invalid and are not inserted in the CST, no matter their selectivity. We therefore introduce a so-called **Invalid N-gram Table** to store invalid n-grams and their selectivity. This table lets us 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 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 to build the Syllable CST and the Invalid N-gram Table, how to estimate selectivity based on them, and how to prune them.

Building the Syllable CST. Prior to the insertion in the SylCST, we decompose every term in its trigrams or syllables, according to one of the strategies described in Section 3.3, and check if it is a non-word using the Dictionary Table. If it is, we store all invalid n-grams in the Invalid N-gram Table, together with their selectivity. 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 [3] refers to as a q-gram estimator. As opposed to [3], however, we do not expect severe overestimations because we deem the invalid n-grams distinctive. In particular, the more characteristic the n-grams of a non-word, the more accurate is the estimation: The out-of-dictionary trigram *fgh*, for instance, strongly identifies Afghanistan. We can reasonably assume that its selectivity is close to the one of the word itself. If a word is valid, in turn, we stem and syllabify it and finally insert the syllable suffixes in the CST.

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 one invalid n-gram, then its selectivity estimate is the minimum of the selectivities of its invalid n-grams.

Pruning. Since both the Invalid N-gram Table and the Syllable CST built over large text corpora have high memory requirements, we cannot do without pruning. We use common frequency-based pruning. Given the maximum size of a CST (in nodes), we iteratively remove nodes whose count is less than a threshold T. We increase T until the CST has the desired size. To estimate the selectivity of a valid string s that is not in the PST, we introduce a syllable-based variant of the MO estimator [6]. If s is syllabified as s_A-s_B-s_C, its estimated selectivity (ESel) is: ESel = $Sel(s_{AB}) \times (Sel(s_{BC}) / Sel(s_B))$, where $s_{AB} = s_A s_B$, $s_{BC} = s_B s_C$. 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.

Table 2: Corpora statistics

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

5. EXPERIMENTAL EVALUATION

We evaluate the performance of our Syllable CST both in terms of memory reduction and of selectivity-estimation accuracy. For our experiments we use four English newswire text corpora, Reuters-21578 (Reuters) [11] and three datasets of the Aquaint Corpus (APW, XIE, NYT) [5]. We tokenize the text to extract single words, filter out stop words, and convert all terms to lowercase. Table 2 contains statistics of our test data.

5.1 Effect of Syllabification

The Syllable CST requires significantly less memory than the CST. Table 3 shows that the size is roughly halved. The figures quantify size as the number of nodes. The actual memory footprint is implementation-specific; the currently optimal implementation [20] takes 8.5 KB per node. However, the size reduction means that we need less memory to build the tree, and that we can prune it at a lower threshold, resulting in higher estimation accuracy.

Table 3: CST size reduction (in nodes)

	CST's size	SylCST's size
Reuters	86772	41565 (52,1%)
APW	558633	308764 (44,7%)
XIE	633899	307001 (51,6%)
NYT	979383	526955 (46,2%)

5.2 Effect of N-gram Analysis

We initialize n-gram analysis with a small reference dictionary of common English words (69004 terms, 650 KB). We Porter stem each dictionary entry, compute its n-grams according to one of the strategies from Section 3.3 and store them in the Dictionary N-gram Table. We then process each index term, inserting out-of-dictionary n-grams in the Invalid N-gram Table. Table 4 lists the number of entries of each table.

Table 4: Dictionary and Invalid N-gram Table size

	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

We retain the Invalid Table since we use it to estimate the selectivity of non-words. Table 5 shows that the greater the corpus size, the larger is the Invalid N-gram Table, and its memory requirements may 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 [3] for the n-gram table. We follow the frequency-based approach proposed in [3] to prune the Invalid Table. This increases the estimation error only insignificantly because the pruning threshold is very low, compared to that of the CST. It turns out that n-gram analysis alone reduces the size of the CST considerably. The size reduction increases with the number of non-words in the corpus. Thus, non-word filtering is particularly beneficial if the data is not very clean. We omit the numbers for n-gram analysis alone due to lack of space. Table 5 gives the size of the Syllable CST built exclusively over valid words. These results show that syllable analysis filters out more words and yields a smaller CST than state-of-the-art techniques in the non-positional case. In the positional case, it does not improve the

results obtained with positional trigram analysis. Table 5 further shows that positional non-word filtering and syllabification together shrink the CST to at most 35% of its original size. 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. The latter lets the MO algorithm better estimate the selectivity of pruned suffixes.

Table 5: Syllable CST size in nodes

Corpus	Non-Word Detection	SylCST
Reuters	TA	34538 (-60,2%)
	SA	29191 (-66,4%)
	PTA	26454 (-69,5%)
	PSA	25847 (-70,2%)
APW	TA	239898 (-57,1%)
	SA	197059 (-64,7%)
	PTA	153005 (-72,6%)
	PSA	154910 (-72,3%)
XIE	TA	216907 (-65,8%)
	SA	179375 (-71,7%)
	PTA	126221 (-80,1%)
	PSA	132886 (-79,0%)
NYT	TA	419359 (-57,2%)
	SA	340327 (-73,9%)
	PTA	255629 (-65,3%)
	PSA	261281 (-73,3%)

5.3 Accuracy of Estimations

We now report on experimental results on the selectivity-estimation accuracy of the Syllable CST. We follow the approach adopted in [7, 6, 3] and evaluate positive queries (i.e., terms that are contained in the corpus) and negative queries (i.e., terms with a 0 selectivity). Finally, we demonstrate that estimation inaccuracies due to pruning are less severe on the Syllable CST.

Evaluation Metrics. For positive queries, we use the average relative error (ARE) to measure estimation accuracy, as suggested in [3]. It is defined as: ARE = |ESel – Sel| / Sel, where ESel is the estimated selectivity and Sel the actual selectivity of a string. We correct this metric, as suggested in [3], to overcome the penalizing effect on low selectivity strings: 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|. Following again [3], we use the average absolute error and its percentage of the corpus size as evaluation metric for negative queries.

Positive Queries. We evaluate the accuracy of our estimator for positive queries by estimating the selectivities of corpus terms as described in Section 4. The average relative error for the Syllable CST, without non-word filtering, is minimal for Reuters (3.5%) and maximal for NYT (11%). These results indicate that conflations due to our stemming algorithm do not introduce significant errors. Non-word detection in turn does incur some errors: For all test corpora but Reuters, the average relative error increases to 13-17%. Further, there are more errors with n-gram analysis. Overestimations, due to multiple invalid words identified by the same invalid n-gram, penalize the estimation of non-words, especially with the non-positional techniques: Consider the terms *Albuquerque* and the German word *Unterbauquerträger*. They both are identified as non-words due to trigram *uqu*. Non-positional trigram analysis conflates these terms in the *uqu* bucket. In consequence, their selectivity is over-estimated, as the sum of their selectivities. Positional trigram analysis avoids this by taking the in-word position into account. However, the average relative error is always under 20%.

Table 6: Average Relative Error and Pruning Threshold for different CST sizes

Corpus	CST Type / Non-Words	CST Size (Nodes)			
		32000	16000	8000	4000
Reuters	CST	21,5% (7)	38,0% (29)	83,5% (109)	143,6% (332)
	CST / NW	19,7% (5)	31,9% (23)	55,8% (97)	89,8% (310)
	SylCST	10,0% (1)	10,0% (4)	24,0% (14)	46,2% (53)
	SylCST / NW	7,01% (0)	7,01% (2)	21,4% (10)	40,9% (46)
APW	CST	52,4% (61)	91,0% (214)	173,1% (666)	325,8% (1726)
	CST / NW	48,3% (44)	76,8% (179)	129,4% (607)	228,8% (1635)
	SylCST	16,6% (8)	52,6% (31)	104,4% (113)	164,2% (355)
	SylCST / NW	19,9% (4)	42,4% (22)	91,2% (95)	143,2% (319)
XIE	CST	33,3% (22)	50,5% (90)	98,4% (313)	186,8% (863)
	CST / NW	36,5% (14)	49,0% (68)	73,0% (266)	126,8% (779)
	SylCST	12,0% (3)	14,7% (11)	40,6% (44)	76,6% (152)
	SylCST / NW	17,7% (1)	18,8% (6)	40,3% (31)	72,6% (127)
NYT	CST	80,5% (122)	150,2% (413)	278,0% (1220)	562,4% (3016)
	CST / NW	71,8% (101)	124,8% (364)	223,5% (1142)	428,8% (2917)
	SylCST	49,4% (16)	125,3% (65)	208,3% (223)	307,4% (663)
	SylCST / NW	32,6% (10)	110,0% (54)	186,6% (199)	276,4% (628)

Negative Queries. The selectivity of negative patterns should be estimated as close to zero. We generate negative strings by introducing random errors into corpus words. The error ranges between 0,02% (Reuters) and 0.15% (NYT). This is one fourth of the 0.6% worst case reported in [4]. This shows that our model does not induce significant errors. We omit the result graphs here.

5.4 Pruning

Despite all reductions, the Syllable CST still occupies a lot of memory and thus requires pruning. Our experiments show that the pruning threshold is lower for a Syllable CST, compared to the standard CST, due to its inherently reduced size. As a result, estimations are significantly more accurate. We iteratively prune the CST and the Syllable CST to meet the same final size of 4000 nodes. Table 6 lists the average relative error and the respective pruning threshold (in brackets) for each tree size. For Reuters, the Syllable CST provides good estimates even with the minimum required size: about 40% average relative error. In general, the Syllable CST always gives the better estimations, due to the lower pruning thresholds: The value of the latter decreases by up to 80%, compared to standard CST. This leaves a more accurate basis for the MO algorithm: The relative estimation error is reduced by up to 70%, compared to the technique from [4].

6. CONCLUSIONS

Estimating the selectivity of query terms is essential for query optimization. For string predicates, estimation frequently relies on Count Suffix Trees (CST) [3, 6, 7]. While CST provide good estimates, their memory consumption is prohibitively high. Pruning tries to solve this problem, by trading in estimation accuracy. 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 is suitable to filter out suffixes with little semantic meaning. Aggressive stemming further reduces the CST size. Finally, a very concise n-gram data structure allows for (a) filtering out non-words during CST construction, and for (b) estimating their selectivity well. The various filtering techniques are independent from each other. They are applicable to other languages as well, provided that there is a stemming procedure, a syllabification routine, or a dictionary for the n-gram filtering. Since all the filtering takes place during CST construction, building the tree requires significantly less memory. For English text, the combination of the filtering mechanisms yields a CST 35% the size of the classical one, with the same estimation accuracy. 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 - Trans. Inf. Syst., E89-D(3):1148–1156, 2006.

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

[3] 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.

[4] D. W. Cummings. American English Spelling: An Informal Description. Johns Hopkins University Press, 1988.

[5] D. Graff. The aquaint corpus of english news text. Linguistic Data Consortium, Philadelphia, 2002.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] D. D. Lewis. Reuters-21578. http://www.daviddlewis.com/resources/testcollections/reuters21578/.

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

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

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

[15] Y. Tian, S. Tata, R. A. Hankins, and J. M. Patel. Practical methods for constructing suffix trees. The VLDB Journal, 14(3):281–299, 2005.

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

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

[18] 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.

[19] C. Silverstein, H. Marais, M. Henzinger, M. Moricz. Analysis of a Very Large Web Search Engine Query Log. ACM SIGIR Forum, Vol. 33, pp. 6 - 12, 1999, ISSN:0163-5840

[20] R. Giegerich, S. Kurtz, J. Stoye, Efficient Implementation of Lazy Suffix Trees, Software: Practice and Experience, Vol. 33, No 11, 2003