Selectivity Estimation for String Predicates: Overcoming the Underestimation Problem

Surajit Chaudhuri Venkatesh Ganti Luis Gravano
Microsoft Research Columbia University
{surajitc, vganti}@microsoft.com gravano@cs.columbia.edu

Abstract

Queries with (equality or LIKE) selection predicates over string attributes are widely used in relational databases. However, state-of-the-art techniques for estimating selectivities of string predicates are often biased towards severely underestimating selectivities. In this paper, we develop accurate selectivity estimators for string predicates that adapt to data and query characteristics, and which can exploit and build on a variety of existing estimators. A thorough experimental evaluation over real data sets demonstrates the resilience of our estimators to variations in both data and query characteristics.

1. Introduction

String-valued data has become commonplace in relational databases and so have complex queries with selection predicates over string attributes (e.g., *Author.name like %ullman%)*. Since query optimizers rely heavily on selectivity estimates, accurate selectivity estimation of string predicates is critical to define efficient query execution plans.

The most frequent class of string predicates—called *wildcard* predicates—are of the form *R.A like %s%*, where *A* is a string-valued (varchar) attribute of a relation *R*. Several techniques have been proposed for estimating the selectivity of wildcard predicates (e.g., [KVI96], [JKNS00]). These techniques build *summary structures* (e.g., *pruned suffix trees or Markov tables*) recording the "frequency" of carefully selected strings. The *frequency* of a string in a relation attribute is the number of attribute values that include the string. The set of string-frequency pairs retained varies with the summary structure. At run time, the estimation of the selectivity of a predicate *R.A like %s%* involves two parts: (i) *parsing* the query string *s* into possibly overlapping substrings s_1, …, s_k whose (exact) frequencies—and hence the associated selectivity of each substring predicate *R.A like %s_i%*—can be looked up in the summary structure, and (ii) *combining* the (exact) selectivities of the substring predicates to estimate the selectivity of the original query predicate. To combine the selectivity of the substring predicates, existing techniques mainly rely either on the *independence assumption* [KVI96] (the selectivity of the *R.A like %s_i%* predicate is independent of that associated with s_j, for all $j \neq i$), or on the *Markov* assumption [JKNS00] (the selectivity of the *R.A like %s_i%* predicate is independent of all *R.A like %s_j%* except when $j = i-1$).

The independence and Markov assumptions may not hold in many real scenarios where the selectivity associated with a string is close to that of some of its substrings. In other words, these assumptions lead to poor selectivity estimates for a predicate *R.A like %s%* if the real selectivity is close to that of *R.A like %s'%*, for a strict substring *s'* of *s*. For example, the selectivity of the string predicate *R.A like %seattle%* may be almost the same as that of the substring predicate *R.A like %eatt%*. In this case, estimators based on the independence or Markov assumptions tend to severely underestimate the true selectivity of the first predicate: these estimates over-compensate for the additional characters not in "eatt" and thus return a small fraction of the selectivity of *R.A like %seattle%*. This observation is orthogonal to the underlying summary structure (e.g., pruned suffix trees or Markov tables) employed. In this paper, we formalize the intuition behind the above example into the s*hort identifying substring* hypothesis. Informally, the hypothesis states that a query string *s* usually has a "short" substring *s'* such that if an attribute value contains *s'*, then the attribute value almost always contains *s* as well.

If we could guess a short identifying substring *s'* of a query string *s*, we could then produce good quality selectivity estimates for the query predicate involving *s*. For example, we can use existing estimators (e.g., an estimator based on the Markov assumption) and return the selectivity estimate for *R.A like %eatt%* as the selectivity of the original predicate *R.A like %seattle%*. Intuitively, this strategy would help overcome the underestimation problem of standard estimators by focusing on a shorter substring with (close to) the same frequency as the original, longer query string. However, a key step in this strategy is to correctly guess the shortest identifying substring, which is of course not possible if only limited statistics are available. But suppose that we know the *length L* of the shortest identifying substring of a query string. In this case, we could estimate the selectivity of all substrings of length *L* and return the minimum estimate as the selectivity estimate for the original predicate, exploiting the fact that the selectivity of a string cannot be larger than that of any of its substrings. Unfortunately, even this length *L* is not generally known when only limited frequency statistics are available. Therefore, our general approach is to *guess multiple*

candidate identifying substrings of a given string *s*, one of each possible length between 1 and |*s*|, and then *combine* their associated selectivity estimates. Our driving hypothesis here (which we confirm experimentally in Section 7) is that an appropriate combination of several reasonable estimates is more robust than any individual estimator. This hypothesis is similar in spirit to that behind the popular and successful *ensemble of models* approaches (bagging and boosting) in the machine learning literature (e.g., [Breiman96, FS96]).

To robustly combine substring selectivity estimates, we have to adapt to characteristics of string values both in the relation and in a query because the length of shortest identifying substrings relative to the original query strings usually vary with query and data characteristics. For example, if substrings of an attribute value in a given relation vary drastically across tuples, then the selectivity of a string predicate over this attribute is likely to closely correlate with that of some of its very short substrings. To adapt to variability in correlations between selectivities of strings and their substrings, we exploit representative query workloads to *learn* an appropriate combination model for the selectivity estimates of candidate identifying substrings over a particular database. The learnt model is then applied at run time to efficiently and accurately estimate the string predicate selectivity.

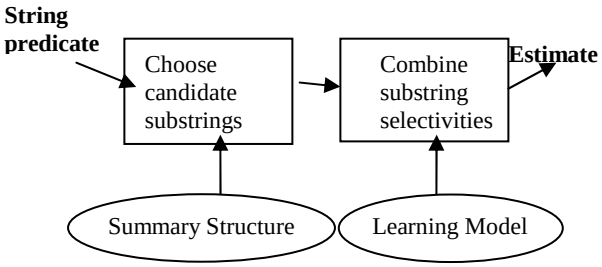

Figure 1: Estimation framework overview.

Figure 1 illustrates our general estimation framework. The choice of summary structure and learning model is largely orthogonal to that of the combination framework. In our evaluation prototype, we use Markov tables [AAN01] and regression tree models (e.g., [BFOS84, Loh02b]), respectively. Further details about these structures are discussed later.

The rest of the paper is organized as follows. Section 2 discusses related work. Then, Section 3 provides preliminary definitions and background necessary to describe our techniques. Section 4 introduces and validates the short identifying substring hypothesis. Section 5 describes our techniques for estimating wildcard predicate selectivities, and extensions to estimate range query predicates (e.g., *R.A between s_1% and s_2%*). Section 6 discusses how to build the summary structures and the learning model. Section 7 demonstrates the effectiveness of our techniques with a thorough experimental evaluation over real data sets. Finally, Section 8 concludes the paper.

2. Related Work

The problem of estimating selectivities for string predicates has started to receive attention recently. Existing approaches to address this problem differ in two aspects: (i) the structures and statistics used for summarizing string attribute values, and (ii) the techniques for deriving selectivity estimates from the attribute value summary.

Krishnan et al. proposed the use of *suffix trees* for summarizing string values in a column [KVI96]. For a given relational attribute, they build a suffix tree—to maintain frequencies of all suffixes of attribute values—and *prune* it so that it fits in the allocated amount of space. The *pruned* suffix tree retains only the most frequent substrings of attribute values. For *estimating* the frequency of a query string *s*, they divide *s* into disjoint strings s_1, …, s_k such that each s_i occurs in the suffix tree. Assuming that an attribute value containing s_i as a substring is *independent* of it containing some other substring s_j, the estimated selectivity of the initial string is the product of the selectivities of the s_1, …, s_k substrings. They build upon this estimator and also consider weighted combinations of estimates of suffixes, where the weight of an estimate is proportional to its length. This is in contrast to our approach of driving the choice of substrings by the short identifying substring hypothesis and the query workload.

Jagadish et al. [JKNS00] improve the estimation step by relaxing the independence assumption, relying instead on the *Markovian* "short memory" assumption. According to this assumption, the probability of an attribute value *v* containing a substring s_{i+1} only depends on *v* containing substring s_i (and not on the earlier substrings). Furthermore, Jagadish et al. allow adjacent substrings to overlap, and also adapt the above ideas to multi-attribute string predicate estimation. Chen et al. in turn extend these techniques to estimate selectivities of queries involving string predicates connected in arbitrary Boolean expressions [CKKM00]. They also enhance the pruned suffix trees by maintaining *summary vectors* with each node. The summary vector of a node represents a "signature" of all tuples with the node's associated string as a substring. These summary vectors can help combine selectivity estimates of individual terms in a Boolean query predicate.

For the related problem of estimating the selectivity of simple XML path expressions consisting of XML tags, Aboulnaga et al. use *Markov tables* over XML tag sequences as the summary structure [AAN01]. In its basic setting, a Markov table of XML tags for an XML data set

records the selectivity of all possible sequences of tags of length not exceeding a pre-specified constant q. The value of q determines the amount of space required to store the Markov table. Aboulnaga et al. also propose techniques to prune the Markov tables so that they do not require more than some given amount of space. Lim et al. [LWP+02] further improve the pruning of the Markov tables–in a workload-aware way—by retaining the selectivity of frequently-used query substrings. The general idea of exploiting query workloads for selectivity estimation has been shown to be effective (e.g., [AC99, LWP+02]).

3. Preliminaries

In this section, we introduce definitions, notation, and background necessary for describing our selectivity estimation techniques.

3.1. Notation

We use *R.A* to denote the attribute *A* in a relation *R*, and *t[A]* to denote the value in attribute *R.A* of a tuple *t*. Let $\sum$ be a finite alphabet of size $|\sum|$ such that values in the attribute *R.A* are drawn from $\sum^*$. Let a symbol '%' *not* in $\sum$ denote the *wildcard* character, which is used for specifying predicates. Let *s* in $\sum^*$ be a string of length $|s|$.

Unit Predicates: A predicate of the form *"R.A like [%]s[%]"*, where *s* does not contain the wildcard character '%', is called a *unit predicate.* The presence of wildcard characters at the beginning and at the end of the predicate is optional (signified by enclosure within square brackets). A unit predicate whose first (last) character is not the wildcard character is called a *prefix (suffix) predicate*. That is, the predicate requires the query string to be at the beginning (at the end, for suffix predicates) of an attribute value. Also, we refer to *s* as the *query string.*

In Section 5.3 we discuss general wildcard predicates as well as range predicates. To unify the treatment of prefix and suffix predicates with that of unit predicates, we introduce the notion of *extended strings* and conceptually replace each attribute value *v* in *R.A* with *"#v$"*. Then, we regard a prefix predicate *"R.A like s%"* as equivalent to predicate *"R.A like %#s%"*, and a suffix predicate *"R.A like %s"* as equivalent to *"R.A like %s$%"*.

Extended String: Let *'#'* and *'$'* be two symbols *not* in $\sum$. Given a string *s*, the *extended string ext(s)* is obtained by prefixing *s* with '#' and suffixing it with '$'. For example, *ext("seattle")="#seattle$"*.

From now on, for simplicity we assume that all attribute values in *R.A* are replaced with their extensions, and that we transform prefix and suffix predicates as above. Also, we refer to a predicate *"R.A like %s%"* simply as *"%s%"* whenever the attribute *R.A* is clear from the context or unimportant for the discussion.

Predicate Matching: A tuple *t* is said to *satisfy or match* a unit predicate *"R.A like %s%"* if *s* is a substring of *t[A]*.

Frequency: The *frequency f(p)* of a unit predicate *p* in relation *R* is the number of tuples in *R* that match *p*. The *selectivity* of predicate *p* is equal to *f(p)/|R|*. For brevity, we also define the frequency *f(s)* of a string *s* over an attribute *R.A* as equivalent to *f("R.A like %s%")*. Similarly, we define the selectivity of a string *s* to be the selectivity of *"R.A like %s%"*.

Q-gram Table: Let *q* be a positive integer. Any string of length *q* in $(\Sigma \cup \{\$, \#\})^*$ is called a *q-gram*. A *q-gram table* $QT_q(R.A)$ for attribute *R.A* is a lookup table with the frequency $f(s_n)$ over *R.A* of each *n*-gram s_n where $1 \le n \le q$. That is, the *q*-gram table consists of the frequency of all *n*-grams of length *q* or less.[1]

Q-gram Sequence: The *q-gram sequence* $Q_q(s)$ of a string *s* with no wildcards is the ordered sequence of all (overlapping) *q*-grams that are substrings of *s*. For example, Q_3("seattle") is [sea, eat, att, ttl, tle].

3.2. Markov Estimator

In our context, a *Markov estimator* (ME) models the selectivity of a unit predicate *R.A like %s%* as the probability of observing the sequence of all *q*-grams in $Q_q(s)$ consecutively in *R.A* values. For *q=3* the selectivity associated with *%novel%* is the probability of observing the sequence Q_3*("novel")=[nov, ove, vel]*. The computation of this probability is simplified by making the Markovian "short memory" assumption, which states that the probability of observing a *q*-gram in the sequence depends only on the *q*-gram immediately preceding it, and is independent of all other preceding *q*-grams. More formally, let *%s%* be a query predicate. If the *q*-gram sequence of the string *s* is $Q_q(s)=[q_1, \ldots, q_k]$, the probability of observing q_{i+1} given $q_1, \ldots, q_i$ under the Markovian assumption is equal to the probability of observing q_{i+1} given q_i. Consequently, if $P(q_{i+1}|q_1, \ldots, q_i)$ denotes the probability of observing q_{i+1} immediately after $q_1, \ldots, q_i$, then the selectivity of *%s%* is computed as:

$$P(q_1) \cdot P(q_2 \mid q_1) \cdot \ldots \cdot P(q_{i+1} \mid q_1,\ldots, q_i) \cdot \ldots \cdot P(q_n \mid q_1,\ldots, q_{n-1})$$

$$= P(q_1) \cdot P(q_2 \mid q_1) \cdot \ldots \cdot P(q_{i+1} \mid q_i) \cdot \ldots \cdot P(q_n \mid q_{n-1})$$

[1] Aboulnaga et al. use the term *Markov table* instead of *q*-gram table [AAN01]. To detach estimation techniques from summary structures, we call them *q*-gram tables instead.

$P(q_i|q_{i-1})$ is the fraction of tuples containing the common substring $cs(q_{i-1}, q_i)$ of q_{i-1} and q_i as well as q_i. This fraction is computed using $f(q_i)/f(cs(q_{i-1}, q_i))$, where $f(cs(q_{i-1}, q_i))$ is the frequency of the common substring $cs(q_{i-1}, q_i)$. For example, the selectivity of the predicate *%novel%* is computed as follows. The 3-gram sequence of "novel" is *[nov, ove, vel]*. The selectivity of *%novel%* is then estimated as: $P(nov)\cdot P(ove \mid nov)\cdot P(vel \mid ove) = f(nov)/N \cdot f(ove)/f(ov) \cdot f(vel)/f(ve)$, where N is the number of tuples. Observe the "multiplicative" relationship between the selectivities of predicate *%s'%* and *%s%*, where *s'* is a strict substring of *s*. In other words, the selectivity of *%s%* is obtained by multiplying the selectivity of *%s'%* with conditional probabilities of observing the additional *q*-grams of *s* in sequence. Consequently, if the selectivities of *%s'%* and *%s%* are close, then the ME selectivity estimator of *%s%* is usually an underestimate. We refer to the selectivity estimated using the Markov estimator described above as the ***ME-Selectivity***.

R.A	**Extended *q*-gram sequences (*q*=3)**
novel	[#no, nov, ove, vel, el$]
article	[#ar, art, rti, tic, icl, cle, le$]
paper	[#pa, pap, ape, per, er$]
journal	[#jo, jou, our, urn, rna, nal, al$]
magazine	[#ma, mag, aga, gaz, azi, zin, ine, ne$]

Table 1: Attribute values and their *q*-gram sequences.

3.3. *QG* Estimator

We now describe the *QG estimator*, which relies on *q*-gram frequency tables to derive an upper bound on the selectivity of a unit predicate. The rationale behind this estimator is that the selectivity of a predicate *%s%* can never exceed that of *%s'%* for any substring *s'* of *s*. In particular, the selectivity of each *q*-gram of *s* is an upper bound on the selectivity of *%s%*. Therefore, the *QG* estimator returns the minimum selectivity of a *q*-gram of string *s* as (an upper bound on) the selectivity of *%s%*. For the example relation in Table 1, the ***QG-Selectivity*** of *%novel%* is $QG(\%novel\%)=min\{f(nov), f(ove), f(vel)\}/5 = min\{1, 1, 1\}/5=0.2$.[2]

4. Short Identifying Substring Hypothesis

We now discuss our hypothesis that query and attribute string values tend to have "short" substrings whose frequency in the underlying relation is close to that of the enclosing string. We now formalize this hypothesis by defining the notion of an identifying substring.

Definition [Identifying substring]: Consider a unit predicate *R.A like %s%*. We say that a substring *s'* of *s* is an (ε, β) *identifying substring*, for $0 \le \varepsilon < 1$ and $0 < \beta < 1$, if (i) the selectivity of *R.A like %s'%* is no larger than $(1+\varepsilon)$ times that of *R.A like %s%* *(i.e., the selectivity of s' is close to the selectivity of s)*, and (ii) $|s'| \le \beta . |s|$ (i.e., *s* is longer than *s'* by at least a factor of $1/\beta$).

For example, "ove" is a (0, 0.6) identifying substring of "novel" for the attribute values in Table 1: the selectivity of *%novel%* coincides with that of *%ove%* (hence $\varepsilon = 0$), *and "ove"* is a strict substring of *"novel"* of length $3 \le 0.6 \cdot 5$ (hence $\beta = 0.6$).

Incidentally, related hypotheses have been proposed for a number of different scenarios. A first example application is the design of "robust hyperlinks" for web pages: [PW00] claims that most web pages can be uniquely identified via a small subset of about 5 of their keywords. Another example application is speech recognition, where a spoken word can be identified via a partial sequence of correctly recognized "substrings," even in the presence of noise [MSHS99]. As a final example, compact tries that collapse intermediate nodes with only one child have been shown to be substantially smaller than their standard-trie counterparts when storing large sets of strings [Sus63]. These observations bear further testimony to our hypothesis that many strings have short identifying substrings.

We now experimentally illustrate the short identifying substring hypothesis using a variety of real data sets and queries. We use the following real data sets: (i) organization names column *(ON)* from a relation consisting of corporate customers, (ii) author names column *(AN)* of all papers in the DBLP database [Ley], (iii) paper titles column *(PT)* of all papers in the DBLP database. The sizes and the average numbers of tokens (words separated by white space characters) and characters per tuple in all three data sets are given in the table below. These statistics illustrate the variety in characteristics across data sets. The strings in *PT* are much longer than those in either *ON* or *AN*.

Data set	*Size*	*Average per tuple*	
		#Tokens	*#Chars*
Organization names (*ON*)	13,495	3.16	25.74
Author names (*AN*)	680,465	2.36	15.84
Paper titles (*PT*)	313,974	8.05	63.72

For each of these data sets, we generate query predicates by randomly selecting a word that occurs in any of the tuples. For example, if *w* is a word in attribute *A* of a tuple, we generate a query predicate "*A like %w%*." Further, we restrict the choice of words to "popular" words, with frequency of at least some threshold, say 100. We denote

[2] Krishnan et al. consider a similar estimator (CE_2) over pruned suffix trees [KVI96].

the set of query predicates obtained from the words in data set *X* with frequency *Y* or higher as *X_fY*. When the frequency threshold is 0 (i.e., when all words are eligible), we drop the suffix *"_f0."* We use the following query data sets: *ON* (386 queries), *AN_f100* (1863 queries), *AN_f500* (293 queries), *PT_f100* (2658 queries), and *PT*_f500 (667 queries). In addition, we also consider *AN-First* and *AN-Last*, involving queries over the first and last names of authors, respectively.

Validation Experiment: Table 2 shows the distribution of the shortest identifying substrings of query tokens when we set ε = 0.05, that is the selectivity of the identifying substring has to be within 5% of the selectivity of the query string. Given this value, we determine the smallest β value for which the query string has an identifying substring.

Query Set	Avg. length	β		3	4	5	6	7
		Mean	S.D.					
ON	7.82	0.59	0.17	21.9	64.0	91.9	98.5	99.4
AN-First	5.84	0.74	0.10	23.7	63.5	93.0	98.4	99.5
AN-Last	6.19	0.72	0.12	22.4	58.3	90.3	98.5	99.7
AN_f100	6.11	0.72	0.11	22.3	60.1	91.2	98.4	99.5
AN_f500	5.71	0.72	0.11	29.3	73.1	95.0	98.7	99.9
PT_f100	8.23	0.63	0.15	17.7	48.5	73.3	86.6	92.6
PT_f500	8.08	0.60	0.15	19.0	57.4	78.8	87.9	95.2

Table 2: Values of β for ε = 0.05.

The "average length" column in Table 2 is the average number of characters in each query. The next two columns are the average and standard deviation of β values given the ε value. Each of the subsequent columns is marked by a number (3, 4, etc.). A value *v* in the column marked by a number *n* indicates that *v%* of the query strings have a substring of length less than or equal to *n* whose selectivity is within 5% of the selectivity of the query string.

The *mean* and *standard deviation (S.D.)* of β values in Table 2 show that β values vary across data sets (the mean is around 0.6 for *ON*, around 0.7 for *AN*, and around 0.6 for *PT*). Around 50% to 70% of the queries we consider have identifying substrings (when $\varepsilon = 0.05$) of length less than or equal to 4, and around 70% to 90% of queries have unique substrings of length less than or equal to 5. Therefore, for high frequency query predicates, traditional estimators might return severe underestimates.

As discussed earlier, whenever query predicates have short identifying substrings, then the Markov estimator tends to underestimate the true selectivities. If we believe that the Markov estimator is accurate for a short identifying substring *s'*, then the *ME-Selectivity ME(s'')* of any super-string *s''* of *s'* is less than *ME(s'): ME(s'')* is obtained by multiplying *ME(s')* with additional conditional probability factors. The margin of underestimation grows as β, the ratio of the lengths of the substring and the query string, decreases.

5. Estimation Algorithms

We now describe our algorithms for estimating selectivities of string predicates. First, we briefly review the regression tree models (which are shown to be effective and powerful data fitting models [BFOS84, Loh02b]) upon which our *combination estimator* relies. The discussion assumes that supporting structures like the *q*-gram table $QT_q(R.A)$ for an attribute *R.A* are available. In Section 6 we describe the construction of such supporting structures. Also, our discussion focuses on estimation algorithms for unit predicates of the form *R.A like %s%*. Later in this section we extend the estimators to cover general wildcard predicates, as well as range predicates.

5.1. Regression Tree Overview

We now briefly review regression tree models, which we use for modeling dependencies between selectivities of a string and its substrings. Please refer to [BFOS84, Loh02b] for a more detailed discussion. Consider a relation *R* with numerical attributes $X_1,\ldots,X_m$, *Y*, of which attribute *Y* is designated as the *dependent* attribute,[3] while attributes $X_1, \ldots, X_m$ are the *predictor* attributes. A *regression tree RT* on relation *R* is a tree-structured model for describing the dependent attribute *Y* in terms of the predictor attributes $X_1, \ldots, X_m$.[4] Each leaf node *n* in the tree is associated with a function $f_n(X_1, \ldots, X_m)$ that predicts the value of *Y* given those of $X_1, \ldots, X_m$. The nature of the function f_n may vary in complexity (e.g., possibilities include a constant function [BFOS84], linear combinations of predictor attributes [Loh02b], or a quantile regression function over predictor attributes [CL02]). Each edge *e* originating from a non-leaf node *n* has a predicate *p* associated with it. For any relation tuple, exactly one of these predicates evaluates to true at each node.

Given a tuple $[x_1,\ldots,x_m, NULL]$ whose *Y* value is unknown, we traverse the regression tree RT starting from its root until we reach a leaf node *n* by following edges whose associated predicates evaluate to true for the tuple. At *n*, $f_n(x_1, \ldots, x_m)$ is the RT predictor of the *Y* value for the tuple.

Figure 2 shows an example of a regression tree constructed from a relation with three attributes, Age, Salary, and

[3] In general, the relation may also have categorical attributes. However, since we only use numerical attributes in our estimation framework, we ignore categorical attributes.

[4] Unlike in decision trees, the dependent attribute *Y* is numeric in regression tree models.

Expenditure, where Age and Salary are the predictor attributes, and Expenditure is the dependent attribute. The input tuple [21, 29K, NULL] is "routed" to leaf node 4, so the predicted expenditure is 0.5·29K+100·21–10 = 16,590.

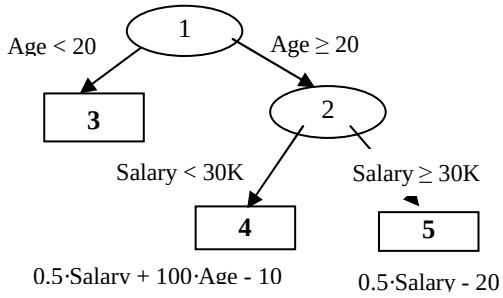

Figure 2: Regression tree for predicting expenditure.

For the selectivity estimation problem we consider in this paper, we attempt to minimize the amount of space required to store the *q*-gram tables and other auxiliary structures. Hence, smaller trees are preferred. Chaudhuri et al. showed that *quantile regression trees* (i.e., regression trees that employ quantile regression functions at leaf nodes) tend to be smaller in size and more accurate than other types of regression trees [CL02]. (Quantile regression models the quantile distribution of the dependent attribute with respect to the predictor attributes.) Because of this, we employ quantile regression trees for our estimates, but other models can be substituted with no change in our general strategy.

5.2. Regression Tree Combination Estimator

In this section we describe our selectivity estimation technique, whose rationale stems from the short identifying substring hypothesis. Ideally, we would like to correctly guess a "minimal" identifying substring (e.g., "eatt") of a query string *s* (e.g., "seattle"). An identifying substring of *s* is *minimal* if it does not strictly contain another identifying substring of *s*. Once such a minimal identifying substring is found, we can use existing estimators (e.g., the Markov estimator) to compute the selectivity associated with just this substring and return it as the selectivity of the original predicate, thus alleviating the selectivity underestimation problem. For example, the selectivity estimate of *R.A like %eatt%* can be returned as that of *R.A like %seattle%*. However, if we only have limited statistics on frequencies of substrings, correctly determining a minimal identifying substring for a given string predicate is not possible.

Our approach (as mentioned in Section 1) therefore is to guess *multiple candidate* identifying substrings, one for each value of substring length between *q* and |*s*|, and combine their estimated selectivities. Since exact selectivities of all *q*-grams are readily available from table $QT_q(R.A)$, we can *precisely* determine the best candidate identifying *q*-gram, whose selectivity is guaranteed to be at least as high as that of the query string. Therefore, we do not consider identifying substrings shorter than *q*. For each length between *q* and |*s*|, we find the substring of that length most likely to be an identifying substring. Our combination function assigns weights to these selectivities. Intuitively, the weight of a substring of length *L* depends on the probability that the length of the shortest identifying substring is equal to *L*. For the example predicate *%novel%* in Figure 3, *W0* depends on the probability that the length *L* of the shortest identifying substring of "novel" is 3, *W1* depends on the probability that *L*=4, and so on. We learn the weights associated with the chosen candidate identifying substrings for each length using a regression tree model.[5] This general approach requires that we specify (i) how to identify *candidate identifying substrings*, and (ii) how to define the *regression-tree combination function*. We now discuss these two issues.

Choice of Candidate Identifying Substrings

To estimate the selectivity associated with a string *s*, we choose one potential candidate identifying substring for each length "level" between *q* and |*s*|. *Level l* consists of all substrings of *s* of length *q+l*. Figure 3 shows the substrings for predicate *%novel%* organized by level for *q=3*: level 0 includes all substrings of length 3 (e.g., *nov*), level 1 has all substrings of length 4 (e.g., *nove*), while finally level 2 consists of the only substring of length 5 (i.e., *novel*).

At each level, we focus on the substring that is most likely to be an identifying substring of the original query string. For this, we build on the observation that the selectivity of any substring cannot be smaller than that of the query string. Therefore, we choose the substring at each level with the smallest (estimated) selectivity. The selectivities of level 0 substrings (i.e., of substrings of length *q*) can be derived precisely from the *q*-gram table $QT_q(R.A)$ for relation attribute *R.A*. Unfortunately, higher levels require that we resort to selectivity estimates, since we do not have exact frequency statistics for strings longer than *q* characters. Although traditional estimators are prone to underestimating true selectivities, as discussed in Section 4, this underestimation is substantially less severe for these short substrings than for the original (longer) string. To estimate substring selectivities, we can in principle exploit any selectivity estimation technique (e.g., Markov or *QG* estimators) that is consistent with our frequency statistics for this task. Our discussion below assumes the Markov estimator of Section 3, so we choose the substring with the smallest *ME-Selectivity* at each level. For the Table 1 example, Figure 3 shows in bold the substring that is picked at each level according to the Markov selectivity estimates.

[5] Besides regression trees, we also explored several other combination functions (Section 6).

Level 2		**novel**		*W2*
Level 1		**nove**	ovel	*W1*
Level 0	nov	**ove**	vel	*W0*

Figure 3: Substring levels for *"%novel%"*.

Regression-Tree Combination Function

As discussed above, if we knew the length L of a minimal identifying substring for a query predicate *%s%*, we could then just estimate the selectivity of the predicate as the estimate for the chosen substring of level *L-q*. Because this length L is not available, we derive the selectivity estimate for *%s%* as the weighted *geometric mean* of the selectivity estimate from each length level. Recall that the *ME-Selectivity* (defined in Section 3.2) of a string *s* is obtained by multiplying the *ME-Selectivity* of a smaller substring *s'* with the conditional probability that a value contains *s* given that it contains *s'*. Due to this non-linear dependence between substring estimates (say, $x_1,\ldots,x_n$), it is more effective to fit a linear model over the logarithms of estimates ($w_1 \cdot log(x_1) + \ldots + w_n \cdot log(x_n)$), which corresponds to the weighted geometric average ($x_1^{w1} \cdot \ldots \cdot x_n^{wn}$). Rather than assigning each level some constant (e.g., uniform) weight in this combination, we instead *learn* the level weights from the data sets and expected query workload, which is a set of string predicates and the true selectivities of all associated substrings. The rationale behind this decision is that different data set-query workload combinations might result in different average minimal identifying substring lengths. Therefore, our combination function adapts to the data characteristics and the correlations between the query string and substring selectivities.

A good combination function for the level selectivity estimates can be learnt from a representative training query workload by using a variety of machine learning tools. For concreteness, our discussion focuses on *regression trees* (Section 5.1), and we consider alternative weighted combinations in our experimental evaluation. (Section 6 discusses how to train a regression tree.) After training, a regression tree produces the selectivity estimate *CRT(s)* for a query string *%s%* from the *ME-Selectivity* values of the candidate substrings at each length level. The regression tree computes *CRT(s)* as a non-linear combination of the input *ME-Selectivity* values, weighting each level as determined during training.

Recall from Section 5.1 that a regression tree takes as input a fixed number of predictor attribute values and returns an estimate value for the dependent attribute. In our context, the number of *ME-Selectivity* estimates that are passed as input to the regression tree depends on the length of the query string. To handle this variability in input size, we "wrap" the regression-tree estimation module to accept a variable-sized estimate sequence. During training, we fix the number of predictor values that the tree will expect as the average length of the estimate sequences for the training queries. Then, our regression-tree wrapper pads shorter estimate sequences with 0's, while it "shrinks" longer estimate sequences by collapsing the selectivity estimates for the largest "levels" (e.g., by taking their maximum).

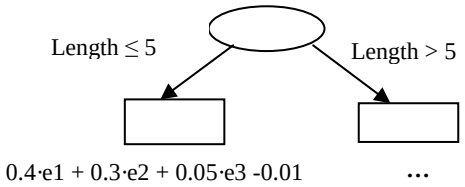

Figure 4: An example regression tree.

As an example, consider the regression tree combination in Figure 4 and the query predicate *%novel%*. As discussed earlier, the three chosen candidate identifying substrings for this query are *ove*, *nove*, and *novel*. Because the length of the original query string (i.e., *"novel"*) is 5, the final selectivity estimate for this query is exp(*0.4·log(ME-Selectivity(%ove%)) + 0.3·log(ME-Selectivity(%nove%)) + 0.05·log(ME-Selectivity(%novel%))-0.01)* when all substring selectivities are positive.

In general, as we demonstrate in Section 7, the estimates obtained using the regression tree combination function are very accurate. Further, like in all learning-based approaches, the improvements in accuracy can be validated by holding back a portion (say, 10%) of the workload as the test dataset over which to measure the accuracy of the built model. Only if the accuracy of the learnt function is better than that of traditional estimators (say, the Markov estimator) may we use it for estimation.

The use of regression trees introduces some overhead in the selectivity estimation process. Most significantly, we need to learn the combination model from a training set of representative example queries. Section 6 discusses this learning step in detail. In Section 7, we demonstrate empirically that the cost of actually traversing the regression trees—which tend to be shallow for our problem—while estimating selectivities at run time is negligible.

5.3. Handling Other Predicate Types

So far, we focused our discussion on unit predicates. We now outline selectivity estimation for a more general class of wildcard predicates, as well as for range queries.

Range Predicates: A predicate of the form *"R.A between s_1[%] and s_2[%]"*, where s_1 and s_2 do not contain the wildcard character '%', is called a *range predicate*.

A range predicate *"between s_1% and s_2%"* can be expressed as the sum of several prefix predicates, one for each string

between s_1 and s_2 in lexicographic order. Fortunately, these prefix predicates can be collapsed to a relatively small number of prefix predicates, the selectivity of many of which can be answered exactly from the *q*-gram table. Consider as an example the range predicate *"between abcz% and daab%"*. We can estimate the selectivity of this predicate as the sum of the selectivities of the (disjoint) prefix predicates *abcz%*, *abd%*, …, *abz%*, *ac%*,…, *az%*, *b%*, *c%*, and *daaa%*. This way, we collapse, for example, all strings starting with *"ac"* into the predicate *"ac%"*, whose selectivity can be derived precisely from the frequency of the *"#ac"* 3-gram.

As a variation, when we process a range predicate *"between s_1% and s_2%"* we can ignore all (longer) prefix predicates between s_1% and s_2% whose exact selectivity cannot be derived from the *q*-gram table. By not considering predicates at the ends of the query range, we hope to introduce less noise in the estimation process, as we confirm experimentally in Section 7. We refer to the resulting estimator for range predicates as *RG_APPX*. For the example predicate *"between abcz% and daab%"* and for *q=4*, *RG_APPX* returns the sum of selectivities of prefix predicates *abd%*, …, *abz%*, *ac%*,…, *az%*, *b%*, and *c%*.

Multi-unit Predicates: A predicate of the form *"R.A like [%]s_1%...%s_k[%]"*, where s_1, …, s_k do not contain the wildcard character '%', is called a *multi-unit predicate.*

The idea behind the extension to multi-unit predicates is similar to that behind the *QG* estimator: the selectivity of each predicate *%s_i%* is an upper bound on that of the original query predicate. We exploit this observation and simply return the minimum of all the *%s_i%* selectivity estimates as the estimated selectivity for *%s_1%...%s_k%*. For the example predicate *"R.A like %microsoft%corp%,"* we return the minimum of the selectivity estimates of *"R.A like %microsoft%"* and *"R.A like %corp%."*

6. Building Supporting Structures

We now discuss how to build the *q*-gram tables (Section 6.1) and regression trees (Section 6.2), as well as their associated space overhead.

6.1. Building *q*-Gram Tables

A *q*-gram table $QT_q(R.A)$ for a relation attribute *R.A* stores the frequency in *R.A* of each string of at most *q* characters from $\Sigma \cup \{\$, \#\}$. We can easily construct $QT_q(R.A)$ from a single scan of relation *R* (e.g., by processing the output of SQL query *"select R.A from R"*). If $C=|\Sigma \cup \{\$, \#\}|$, the *q*-gram table conceptually consists of $QL = C+C^2+...+C^q=(C^{q+1}-1)/(C-1)$ (not necessarily non-zero) entries.[6] Therefore, $QT_q(R.A)$ stores at most *QL* frequencies. We can structure these entries using, say, hash tables to store only the non-zero frequencies. Alternatively, we can use a dense representation in an array and avoid storing the actual *q*-grams. In this case, the frequencies are ordered lexicographically according to their corresponding *q*-grams, so that the entry associated with a given *q*-gram can be readily identified with a simple calculation.

To further reduce the size of the *q*-gram table, we can follow Aboulnaga et al. [AAN01] and Lim et al. [LWP+02], and maintain only the selectivity of "important" *q*-grams, while assuming a default (average) frequency for the remaining ones. The notion of "importance" of a *q*-gram may be tied to its selectivity: the higher the selectivity, the more important the *q*-gram [AAN01]. This notion may be further adapted so that *q*-grams are weighted according to their usefulness for deriving accurate selectivity estimates for a given query workload [LWP+02]. We do not discuss these space-reduction strategies further in this paper.

6.2. Building Regression Trees

To build a regression tree for a relation attribute, we need a *training set* consisting of a representative query workload of string predicates. Such a training set is typically easy to obtain, e.g., from a trace collected by the profiler tool available with most commercial database systems. (If no such workload is available, a sample of words or substrings from attribute values might be used instead.) Given a training set, we can then use standard regression tree construction algorithms such as the GUIDE algorithm developed by Loh et al. [Loh02b, CL02], whose associated software [Loh02a] we use in our experiments.

To prepare the training set, we compute the exact selectivity of all query predicates in the training workload by scanning the relation in question once. In fact, this selectivity computation can be piggybacked as part of the *q*-gram table computation. Then, for each query predicate we prepare an entry in the training set consisting of the following features:

Query String Characteristics: Figure 4 shows a hypothetical scenario in which the dependence between the selectivity of substrings and the query predicate is different for query strings of length less than or equal to 5 than it is for query strings of length greater than 5. The regression tree in the figure is able to model this by associating different combination functions to different leaf nodes, and splitting on the length of the query string. One of the predictor values that we associate with query predicates is then the *length* of the associated query string.

[6] *QL* is actually lower since the special characters # and $ only occur at the beginning and at the end of the *q*-grams. We ignore this detail to simplify our description.

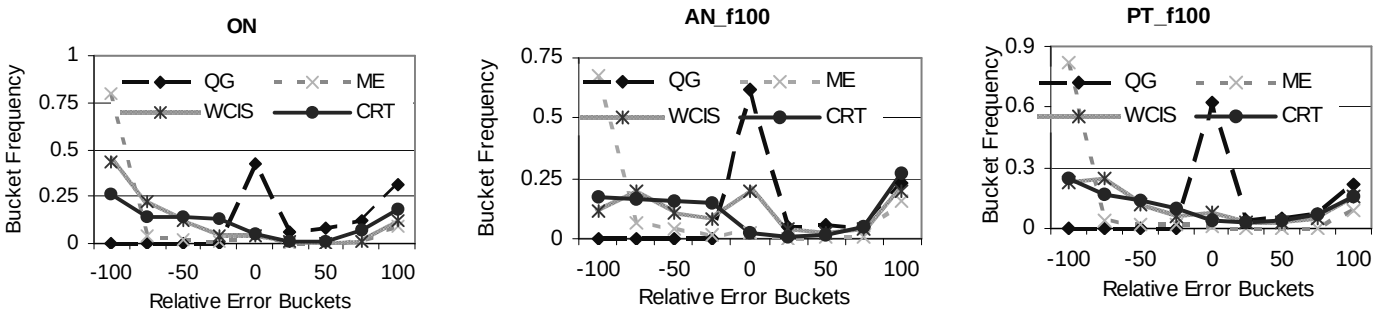

Figure 6: Quartile distribution of average relative error.

Query Substring Selectivity Estimates: As described above, we choose one substring per length "level" as the candidate substring. We include the logarithms of *ME-Selectivity* estimates of candidate substrings—one per level—as additional predictor values.

The space overhead introduced by using regression trees is negligible. Each non-leaf node of the tree needs to encode the predicates associated with its outgoing edges. In particular, quantile regression trees (as well as other typical classes of regression trees) are binary (i.e., each internal node has only two outgoing edges). Therefore, just two numbers—an integer for the attribute identifier and a real number for the attribute split value—need to be maintained for our setting. Finally, each leaf node simply maintains a set of weights defining a linear combination of the predictor values into the query selectivity estimate. In Section 7.4, we show experimentally that regression trees on various combinations of data sets and query workloads tend to be very shallow, consisting of at most 2 levels and hence requiring very little space. Relative to the space required by the *q*-gram tables (in the order of thousands of bytes), the additional space needed for the regression trees (in the order of hundreds of bytes) is negligible.

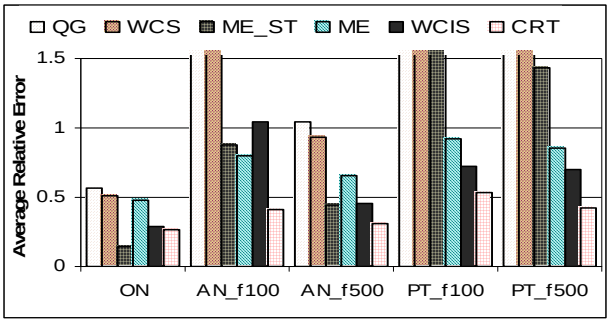

Figure 5: Average relative error.

7. Experimental Evaluation

We now evaluate our estimation techniques using real data sets to show (i) that simple non-adaptive combination functions are not accurate thus motivating the use of adaptive learning-based combination functions, and (ii) that the combination estimators are better than previous estimators for estimating string predicate selectivity. We first describe our experimental setup.

7.1. Setup

We first review the data sets, the training and test query workloads, which agree with those used in Section 4. We consider three data sets: (i) Organization names (*ON*), (ii) Author names (*AN*), and (iii) Paper titles (*PT*). Recall that the query workloads are: *ON* (386 queries), *AN_f100* (1863 queries), *AN_f500* (293 queries), *PT_f100* (2658 queries), and *PT_f500* (667 queries).

While evaluating the *CRT* regression-tree estimator of Section 5.2, we split the workload into a training set (consisting of either 250 queries or 50% of the workload, whichever is smaller) over which the regression tree combination is learnt, and a test set consisting of the remaining queries. In all the experiments on regression trees, we fix the number of predictor variables to 9: as explained, one variable is for the string length, while the remaining 8 are for substring selectivity estimates.

7.2. Evaluation Metrics

The first metric we consider is the *average relative error*, which measures the overall accuracy of an estimator as the ratio $|estimate\text{-}actual|/actual$, where *estimate* is the selectivity estimate and *actual* is the exact value of the selectivity. Unfortunately, the average relative error heavily penalizes errors with respect to small selectivities. We therefore adopt "corrections" for this metric from the literature that are aimed at lessening the impact of this problem [CKKM00, LWP+02]. Specifically, if the actual selectivity of a predicate over a relation R is less than $100/|R|$, we divide the absolute selectivity error with $100/|R|$, rather than with *actual*.

The second metric we consider is the *quartile* distribution of the relative error to illustrate the bias and the overall accuracy of an estimator. The quartile distribution *bucketizes* the relative error distribution into the following buckets: [-100%, -75%), [-75%, -50%), [-50%, -25%),

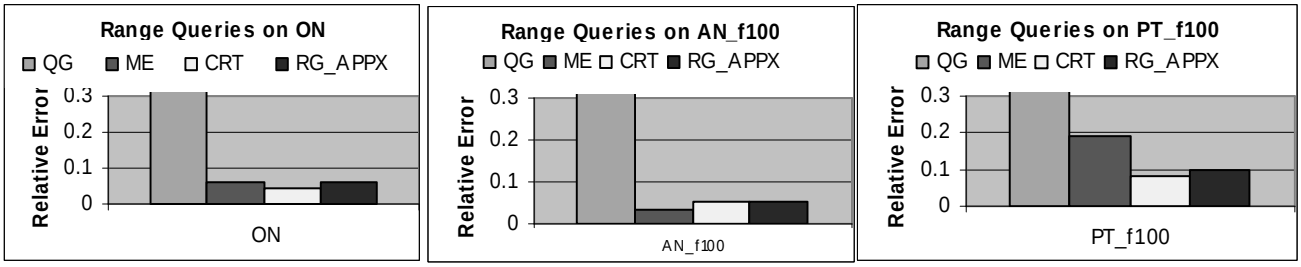

Figure 7: Accuracy of estimators for range queries.

[-25%, 0%), [0%, 25%), [25%, 50%), [50%, 75%), [75%, 100%), [100%, infinity). Estimates that fall into negative buckets indicate underestimates, and those that fall into positive buckets are overestimates.

7.3. Estimation Techniques Compared

We compare our new regression tree combination estimator, CRT, with several estimators from the literature, which we discussed above: Markov estimators over both *q*-gram tables (*ME*) and suffix trees (*ME_ST*)—referred to as the *maximal overlap* method in [JKNS00], as well as the *QG* estimator. We also consider two other weighted combination estimators over suffix trees: *WCS* and *WCIS*. Krishnan et al. [KVI96] proposed *WCS*, a weighted combination of estimates of all *suffixes* $[s_q, \ldots, s_{|s|}]$ *of the query string s* of lengths between *q* and |*s*|. This can be viewed as using—for each length "level"—the suffix of that length of *s*. The weight of the estimate for suffix s_i is proportional to its length $|s_i|$, and the sum of all weights is constrained to be equal to 1. We extend *WCS* to *WCIS* by considering the candidate identifying substrings at each level instead of suffixes. *WCIS* still uses the same weighting policy as *WCS*. Thus, the difference in accuracy between *WCS* and *WCIS* reflects the importance of choosing good candidate substrings.

Unless otherwise specified, we allocate *36.4 kilobytes of memory* for all summary structures: *q*-gram tables and, optionally, regression trees and suffix trees. This space can accommodate 3-gram tables (i.e., *q*=3) and approximately 3600 nodes in the suffix tree. We ran all experiments on a 2.2-GHz desktop PC with 512 MB of main memory running Windows XP Professional.

7.4. Experimental Results

We now present our experimental results on accuracy, and report on the time and space overheads of regression trees.

Accuracy of Estimates for Unit Predicates

We demonstrate the accuracy and robustness of the *CRT* estimator by plotting the average relative error and its quartile distribution.

Figure 5 plots the average relative error over several data sets and test queries. First, the *CRT* estimator is consistently the best for almost all data sets (except for the small *ON* dataset, whose unpruned suffix tree almost fits entirely within the allotted main memory), often by more than 30-50%. Second, *WCIS* is consistently better than *WCS*. Together, these imply that exploiting the identifying substring hypothesis yields significant improvements and, furthermore, that learning the appropriate combination helps improve accuracy. That *WCIS* is better than *WCS* shows the importance of choosing candidate substrings appropriately. Further, the importance of learning a robust adaptive combination function is highlighted in those cases (query sets *AN_f100*, *PT_f100*, and *PT_f500*) where *WCIS* is substantially worse than *CRT*. To illustrate the importance of using robust adaptive combination functions, we also experimented with two other functions for combining the estimated selectivities of candidate substrings: (i) the *geometric mean* and (ii) the *weighted geometric mean* where the weight for each level *L* is the probability *P(L)* that the shortest identifying substring of a query has length *L*. These probabilities are computed using the workload. *P(L)* is the proportion of queries in the workload whose shortest identifying substring is of length *L*. We are unable to present the plots due to space constraints. These combination functions yield substantially high errors (worse than *WCIS*) and vary widely across datasets, which highlights the importance of using sophisticated combination strategies.

Figure 6 plots the distribution of QG, ME, WCIS, and CRT estimates around the actual values for query data sets *ON*, *AN_f100*, and *PT_f100*. Each *[x, y)* "bucket" is represented by its left endpoint *x* on the *x*-axis. High values at the negative (or at the positive) end of the *x*-axis indicate bias towards underestimation (or overestimation, respectively). *ME* severely underestimates selectivities for a large percentage (sometimes, around 90%) of queries, thus supporting the motivation behind our approach. Even though *QG* and *WCIS* have a significant percentage of estimates around 0, they overestimate (with large relative errors above 100%) for a significant percentage of queries. Hence, their overall accuracy (as shown by Figure 5) is low.

The plots of quartile distribution and average relative error confirm the observation made by Krishnan et al. [KVI96] that *WCS* often results in drastic overestimates of the true selectivity. That the equivalent *WCIS* estimator fares much better underlines the validity and the usefulness of the short identifying substring hypothesis.

Accuracy of Estimates for Range Predicates

We now evaluate estimators for answering range predicates. In this experiment, we adapt the standard approach employed for evaluating the accuracy of histograms in answering range predicates on numeric attributes [IP95]. We generate a predicate *"between t_1% and t_2%"* by randomly picking two tokens t_1 and t_2 from among the set of all tokens occurring in the attribute values of *R*.A. (We swap the values appropriately so that $t_1 < t_2$ in lexicographical order.)

Figure 7 shows the results. Once again, *CRT* is usually the most accurate technique, though not by a significant margin all the time. Interestingly, the accuracy of *RG_APPX* (the estimator that ignores all intermediate prefix predicates that cannot be answered exactly; see Section 5.3) is comparable to that of *CRT*.

	ON	***AN***	***PT***
CRT	*77 (0.57%)*	*308 (0.05%)*	*312 (0.09%)*
ME	*4.93(0.04%)*	*30 (0.005%)*	*115 (0.03%)*
WCIS	*26 (0.19%)*	*658 (0.1%)*	*1787(0.057%)*

Table 3: Average absolute error (and as percentage of relation size) for negative queries.

Accuracy of Estimates for Negative Queries

We now consider "negative" queries that produce empty answers. For experiments over a particular data set, we use queries drawn from a different data set. For example, for a summary built upon the author name (*AN*) column, we generate queries from the paper title (*PT*) column. By construction, the actual selectivity of these queries tends to be very close to zero: 60-85% of queries have frequencies less than 5. Table 3 shows the *average absolute error* of the CRT, ME, and WCIS estimators along with the percentage of the relation size to which this error corresponds. (The results for other estimators are similar.) ME usually underestimates selectivities, so it performs particularly well for negative queries. The absolute errors for CRT are higher, but still small in absolute terms (on average less than 0.6% of the relation size for all data sets that we tried).

Effect of *q* on Accuracy

In this experiment, we vary the amount of available memory for the summary structures, so that the value of *q* varies between 2 and 4. Figure 8 shows the average relative error for the *ME* and *CRT* estimators on several data sets, as we vary the value of *q*. As expected, the average relative error decreases as we increase *q* for all estimators. Moreover, the *CRT* estimator is consistently better than *ME* across these values of *q*, and is accurate even for *q*=2.

Data and Query Set	**Regression Tree Size (in bytes)**
ON	80 (0 *NL* + 1 *L*)
AN_f100	172 (1 *NL* + 2 *L*)
AN_f500	80 (0 *NL* + 1 *L*)
PT_f100	80 (0 *NL* + 1 *L*)
PT_f500	80 (0 *NL* + 1 *L*)

Table 4: Size of the regression trees for our data sets.

Regression Trees: Time and Space Overhead

We now demonstrate that the size and time overheads due to the use of regression trees are negligible. In our implementation, each internal or non-leaf (*NL*) node in the regression tree requires 12 bytes (attribute id and split value) and each leaf (*L*) node requires 80 bytes for encoding the linear combination of attributes: 8 bytes each for the weight of 9 predictor attribute variables and 8 bytes for a constant. Table 4 shows the sizes of the regression trees for all data sets and query workloads. Observe that even very small regression trees provide significant improvements in accuracy because they encode weighted combinations of substring selectivities that are learned from the workload. For example, the weights for the linear combination function for *ON* in Table 4 are [-0.009, -0.1, 0.2, -0.4, 0.4, -0.003, -0.08, 0.001]. Note also that the size of a *q*-gram table (using the array representation discussed in Section 6.1) is 35,600 bytes for *q*=3, and 3000 bytes for *q*=2. Therefore, the additional amount of space required for regression trees is negligible compared to that for *q*-gram tables.

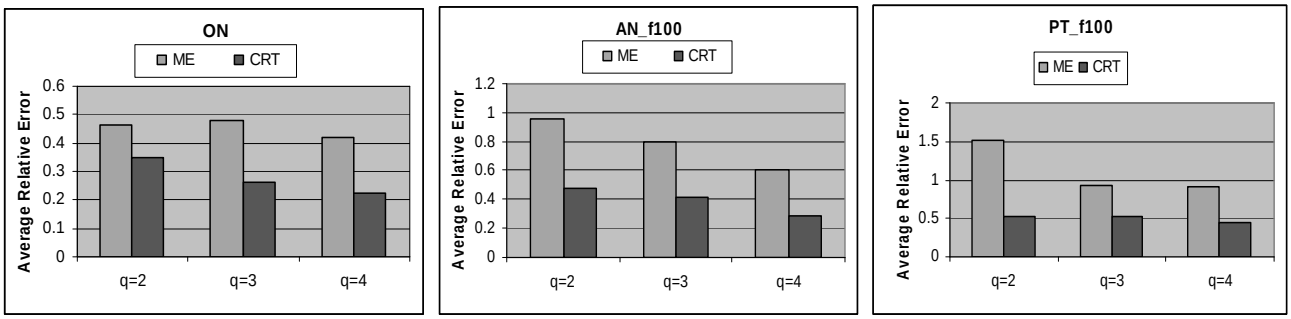

Figure 8: Average relative error versus *q*.

Table 5 shows the time required for building *q*-gram tables and the additional time for learning the regression trees—including both the preparation of the training data and the actual learning. As shown in the table, the overhead for learning regression trees is lower than that for building the *q*-gram tables. Note that the time for preparing training data can be reduced by "piggybacking" the table scan during the *q*-gram table construction, as suggested in Section 6.2. Our implementation, over which the numbers reported above were computed, does not use this optimization.

We also measured the average time required for estimating the selectivity of a predicate: the time variations turned out to be very small, at around *4 microseconds* for estimators *QG*, *ME*, and *CRT*. Thus, the overhead of using regression trees for combining estimates is negligible.

Data set	**Time (*in seconds*)**		
	***Q*-gram Tables**	***CRT***	
		Preparing Data	**Building**
ON	11	2.2	9
AN_f100	437	234.9	16
AN_f500	437	62.6	19
PT_f100	621	140.1	15
PT_f500	621	48.3	17

Table 5: Time for building regression trees.

In summary, our experiments over real data sets showed that our new estimator based on regression tree combination is more accurate and robust than traditional estimators, and that the associated time and space overheads are negligible.

8. Conclusions

In this paper, we developed accurate selectivity estimators for string predicates that adapt to data and query characteristics. Our estimators leverage and build on state-of-the-art estimation techniques, and are based on the hypothesis that query strings usually have relatively short substrings that almost uniquely "identify" them. We empirically showed that selectivity estimators that do not exploit this observation often severely underestimate selectivities. In contrast, our new estimation techniques exploit this hypothesis and learn how to weight the selectivity estimates for query substrings during a training phase over a representative query workload. Our experimental evaluation over real data sets suggests that our workload-aware estimates are accurate and robust, while at the same time imposing only low space and time overheads. Furthermore, the selectivity weights can be learned efficiently, so the combination model can be recomputed periodically to reflect workload or data changes.

Acknowledgements
We thank Zhiyuan Chen for providing us with source code of the suffix tree implementation.

References

[AAN01] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. Estimating the selectivity of XML path expressions for Internet scale applications. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 591—600, 2001.

[AC99] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proceedings of the 1999 ACM International Conference on Management of Data (SIGMOD)*, 1999.

[Breiman96] L. Breiman. Bagging predictors. *Machine Learning* 24, 123-140, 1996.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth, Belmont, CA, 1984.

[CKKM00] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. Selectivity estimation for Boolean queries. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 216—225, 2000.

[CL02] P. Chaudhuri and W.Y. Loh. Nonparametric estimation of conditional quantiles using quantile regression trees. *Bernoulli*, 8:561—576, 2002.

[FS96] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148—156, 1996.

[IP95] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 233—244, 1995.

[JKNS00] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. One dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal* (2000) 9, pages 214—230, 2000.

[KVI96] P. Krishnan, J. S. Vitter, and B. R. Iyer. Estimating alphanumeric selectivity in the presence of wildcards. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 282—293, 1996.

[Ley] M. Ley. DBLP. Computer science bibliography. http://www.informatik.uni-trier.de/~ley/db.

[Loh02a] W.-Y. Loh. The GUIDE system for building regression trees. http://www.stat.wisc.edu/~loh/guide.html.

[Loh02b] W.-Y. Loh. Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica, 12:361—386*, 2002.

[LWP+02] L. Lim, M. Wang, S. Padmanabhan, J. Vitter, and R. Parr. XPathLearner: An on-line self-tuning Markov histogram for XML path selectivity estimation. In *Proceedings of the 28th International Conference on Very Large Databases*, pages 442—453, 2002.

[PW00] T. Phelps and R. Wilensky. Robust hyperlinks and locations. *D-Lib Magazine*, 6(7), July 2000.

[MSHS99] J. Ming, D. Stewart, P. Hanna, and F. J. Smith. A probabilistic union model for partial temporal corruption of speech. In *Automatic speech recognition and understanding workshop*, Keystone, Colorado, December 1999.

[Sus63] E. Sussenguth. Use of trees for processing files. *Communications of the ACM*, 6(5):272--279, 1963.