Query Size Estimation using Machine Learning

Banchong Harangsri  John Shepherd  Anne Ngu
School of Computer Science and Engineering,
The University of New South Wales, Sydney 2052, AUSTRALIA.
Telephone: +61 2 385 3980  Fax: +61 2 385 1813
Email: {bjtong,jas,anne}@cse.unsw.edu.au

Abstract
In a previous paper [6], we introduced the notion of using machine learning techniques to solve the problem of query size estimation in database query optimisation. In this paper, we build on this work by describing a new generic algorithm to correct the training set of queries for our machine learning method in response to updates. The training set correction algorithm is not only useful in the context of our machine learning approach, but is also useful for improving existing query size estimation methods whose performance deteriorates in the presence of high update loads. A by-product of our correction algorithm is that training sets can be fixed-size, allowing the error-level to be set in advance. Experimental results show that our machine learning technique performs well (and better than alternative methods) after the correction algorithm is applied.

Keywords  Query Size Estimation, Query Optimisation, Machine Learning

1 Introduction
A query optimiser for a database system aims to determine the most efficient execution plan for each query submitted to the system. Choosing an efficient plan relies on cost estimates derived from statistics maintained by the database system. Analysis in [9] showed that even very small inaccuracies in the cost estimates can cause the optimiser to choose a very poor (suboptimal) execution plan. Very accurate estimates for the cost of database operations are thus critical to the effective operation of query optimisers. This paper proposes a novel method to improve such cost estimation for a class of selection queries and provides experimental evidence that it is superior to existing approaches.

There has been a considerable amount of work on the issue of cost estimation for selection operations over one and a half decades [16, 3, 4, 14, 10, 8, 12, 13, 11, 5, 17, 2]. Previous work can be classified into four categories [17, 2]: non-parametric, parametric, sampling and curve-fitting. We briefly describe each of them here; more details can be found in the references given above.

Non-parametric methods are table- or histogram-based [14, 13]. A histogram may be built for an attribute in a relation by partitioning the attribute domain into intervals and counting the number of tuples in each interval. These methods require scanning an entire relation to build up attribute histograms. Histogram-based methods give more accurate results when the number of partitions is higher. Because histograms need to be stored, the quest for high accuracy can lead to considerable storage costs. These methods can deal with data changing over time, but at the cost of periodically rescanning relations.

Parametric methods [16, 3, 4] rely on assumptions about the underlying data distributions of attributes. These methods have low storage costs (simply the parameters of the distributions) and can produce estimates very quickly. These methods are very accurate if the actual data distribution follows the a priori assumptions well, but data distributions in real databases may not fit well with the assumed distributions. These methods also have problems if the data distribution changes over time as a result of database updates.

Sampling methods [11, 5] execute the queries to be optimised on small subsets (samples) of the real database, and use the results of these trials to determine cost estimates. These methods can give very accurate estimates for complex query plans, since they are effectively “pre-executing” plans to determine the costs. However, the accuracy of these methods depends critically on the sample size. Larger samples give more accurate estimates but, of course, increase the cost of computing the estimates. This is particularly problematic for complex (multi-step) queries where the cost of computing the estimates for the many possible plans can be a prohibitive overhead. The accuracy of these methods is not adversely affected by database updates, as the samples are determined
dynamically and always reflect the current data distribution.

Curve-fitting methods [17, 2] are based on polynomial regression to find the set of coefficients that minimises some least-squared error criterion. The actual criterion differs substantially from method to method.

The curve-fitting method proposed by [17], scans entire relations and uses regression to determine the distributions of attribute values in each relation. This approach is effective only for low-update database systems. That is, as long as the distributions of attribute values remain fixed, the method performs satisfactorily. However, if the distributions change, then the accuracy of the size estimates may deteriorate significantly. To repair the situation, re-scanning of relations is required.

The curve-fitting method proposed by [2], called adaptive selectivity estimation (or ASE), also attempts to determine data distributions using regression. However, instead of scanning relations and computing distributions directly, it uses feedback from query execution. It assumes queries are of the form low \( \leq a1 \leq high \), where \( a1 \) is an attribute, and uses the result sizes of the queries as the basis for regression. As more and more queries are processed and more feedback becomes available for regression, the method gives more accurate query size estimation. To deal with updates, ASE uses "fading weights" to gradually reduce the significance of old query feedback. However, fine-tuning for the best set of fading weights turns out to be a difficult problem.

The method that we propose in this paper aims to overcome most of the difficulties mentioned above. Our overall approach is to derive size estimation functions using machine learning techniques. We proceed as follows:

1. use feedback from a training set of queries to construct a model tree (or regression tree) [1]; the model tree contains a collection of specialised, accurate cost estimation functions

2. to estimate the result size of a given query (\( q_u \)): determine the three queries \( \{q1, q2, q3\} \) in the training set that are most similar to \( q_u \); use the model tree and the result sizes for \( q1, q2 \) and \( q3 \) to estimate the result size of \( q_u \).

Experimental work in [7] has shown that, of the methods mentioned above, ASE and our machine learning produce the most accurate estimates, and so we consider only these two methods in the experiments in this paper.

Advantages common to ASE and our method:

no relation scan: We do not need to scan relations to collect statistics on which to base query size estimation. All of the methods above, except the parametric method, require scanning of relations.

adaptive: The estimation accuracy improves as more queries are processed and stored in the training set. In the ASE method, extra query feedback assists in adjusting the data distribution curve to better fit the actual distribution of attribute values. In our method, the extra feedback is used to help find a better \( \{q1, q2, q3\} \).

However, in the presence of very high update loads, the two methods use different approaches to maintain size estimation accuracy. We believe that our approach is more effective, and more widely applicable than the approach used by ASE.

Our method has these advantages over ASE:

**generic update algorithm:** The algorithm that we describe in section 4, can also be used (with minor modifications) with the size estimation methods proposed in [17, 2]. In other words, we can adapt size-estimation methods proposed for retrieval-only or retrieval-intensive environments for use with dynamic (high-update) databases. Details of how to achieve this may be found in [7].

**fixed-size list:** The scheme in [2] maintains a list of recent query feedback, but also needs to maintain outdated query feedback (i.e. feedback that was obtained prior to updates). Using our update algorithm, we can put a bound on the amount of query feedback that needs to be stored, since all of the feedback records can be maintained to reflect the current database state. In other words, after the length of feedback lists has reached a certain size and the error in query size estimation has dropped to a satisfactory level, then the amount of stored feedback remains constant.

The rest of this paper is structured as follows: In section 2, we define our terminology. Section 3 briefly summarises our approach to using machine learning techniques for query size estimation. Section 4 discusses the relationship between queries and attribute value frequency distributions and then describes our algorithm to correct the list of queries in a training set. In section 5, we compare the performance of our method to the ASE method. In section 6, we give some final remarks and address issues for future investigation.

2 Notation and Definitions

In this section, we define some terminology related to the query optimisation problem and to machine
In particular, the terminology for attributes comes from the machine learning domain and not from the database domain.

A simple query \( q_i \) is a selection query on a single attribute\(^1\) of the form: \( b \ \text{op}_{rel} \ x \), where \( b \) is an attribute of relation \( R \), \( x \) is a constant value, and \( \text{op}_{rel} \) is one of the relational operators in \{ \( <, >, =, \neq \) \}. Note that, with the addition of the standard set intersection and union operators, these four relational operators are sufficient to cover other types of simple query, such as \( b \leq x \), \( b \geq x \) or \( \text{low} < b < \text{high} \). For example, \( \text{low} < b < \text{high} \) can be replaced by \( (b < \text{high}) \cap (b > \text{low}) \).

We frequently use the term attribute to refer to a (syntactic) component of a query. For example, the simple query above consists of 3 attributes: \( b \), \( \text{op}_{rel} \) and \( x \). The term attribute will occasionally be used with its normal meaning “attribute of a relation”. The intended meaning is always clear from the context.

Attributes are described as continuous attributes if they have a natural ordering on their domain. The \( x \) attribute in our simple query schema is an example of a continuous attribute, and would typically be either a numeric or string constant. Numeric domains have a natural order. String values are converted to numeric values by sorting all strings in the training set in ascending lexicographic order and assigning a rank (i.e. 1,2,....) to each string.

Attributes are described as discrete attributes if they have no natural ordering on their domain. The \( b \) and \( \text{op}_{rel} \) attributes in our simple query schema are examples of discrete attributes.

A training set of queries \( Q \) is a set of simple queries and their result sizes. These are collected from previous user-submitted queries\(^2\). Each \( q_i \in Q \) is of the form \( q_i : (b, \text{op}_{rel}, x_{\text{scaled}}, \Sigma \text{sq}_i) \) where \( \Sigma \text{sq}_i \) is the result size of the query and \( x_{\text{scaled}} \) is an \( x \) value linearly scaled by:

\[
x_{\text{scaled}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}
\]

where \( x_{\text{min}} \) and \( x_{\text{max}} \) are, respectively, the minimum and maximum values in the domain of relational attribute \( b \).

An unseen query \( q_u \) is a simple query whose result size we want to estimate. After this query has been processed by the database system, and its actual result size measured, it may be added to \( Q \).

### 3 Size Estimation with Retrieval Queries

In this section, we summarise our method of using machine learning to solve the query size estimation problem; this method is described in detail in [6]. The machine learning mechanism that we use is a combination of model-based learning and instance-based learning originally proposed by Quinlan [15]. It involves construction of a model tree, a kind of regression tree originally proposed by Breiman et al. [1]. Our implementation of the method has been tailored for use in this application, the two major differences being that we have omitted the pruning and smoothing procedures from Quinlan's original implementation.

Under our scheme, we first build a model tree for one attribute of the relation using a training set of queries \( Q \). The model tree effectively partitions the set of queries for this attribute into subsets which have similar result sizes. The top level of partitioning is based on the relational operator \( \text{op}_{rel} \) used in the query, and the lower levels of partitioning are based on the constant value \( x \). Each leaf node in the tree contains a query size estimation function \( M(q_i) \), which is computed using linear regression on the set of queries \( Q_i \subseteq Q \) defined by that leaf node.

To estimate the size of an unseen query \( q_u \), we proceed as follows: (1) find the three queries \( \{q_1, q_2, q_3\} \) in the training set \( Q \) that are “most similar” to \( q_u \) according to a similarity measure \( \text{simval}_{q_i} \) (see [6]) (2) use the model tree to compute an estimate of the size of the result of \( q_u \) (i.e. compute \( M(q_u) \)) (3) produce the final size estimate \( \hat{S}_{q_u} \) by combining \( M(q_u) \) with the estimated sizes for each of \( q_1, q_2, q_3 \), according to the following formulæ:

\[
\hat{S}_{q_u} = S_{q_i} - (M(q_i) - M(q_u)) ; i = 1,2,3
\]

The detailed rationale for this approach can be found in [6].

With respect to performance, there are two aspects to be considered in this scheme: the cost of building the model tree, and the cost of using the model tree to compute query size estimates. Both of these costs are dependent on the size of the training set. If we use a very large training set, the tree may become very large and expensive to maintain. On the other hand, a larger training set gives more accurate results. Using the model tree is relatively cheap as it requires a simple traversal to a leaf node and calculation using the \( M(q_i) \) function. Computing a query size estimate also requires a complete traversal of the training set to find the three most similar queries. In practice, a training
Table 1: Effect of updates to a training set of queries

<table>
<thead>
<tr>
<th>qi</th>
<th>query</th>
<th>$S_q$</th>
<th>qi</th>
<th>query</th>
<th>$S_q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$b_1 &gt; 14$</td>
<td>9664</td>
<td>1</td>
<td>$b_1 &gt; 14$</td>
<td>9664</td>
</tr>
<tr>
<td>2</td>
<td>$b_1 &lt; 26$</td>
<td>269</td>
<td>2</td>
<td>$b_1 &lt; 26$</td>
<td>269</td>
</tr>
<tr>
<td>3</td>
<td>$b_1 = 20$</td>
<td>8</td>
<td>3</td>
<td>$b_1 = 20$</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>$b_1 &lt; 6$</td>
<td>9664</td>
<td>4</td>
<td>$b_1 &lt; 6$</td>
<td>9664</td>
</tr>
<tr>
<td>5</td>
<td>$b_1 = 41$</td>
<td>366</td>
<td>5</td>
<td>$b_1 = 41$</td>
<td>366</td>
</tr>
<tr>
<td>6</td>
<td>$b_1 &lt; 41$</td>
<td>399</td>
<td>6</td>
<td>$b_1 &lt; 41$</td>
<td>399</td>
</tr>
<tr>
<td>7</td>
<td>$b_1 = 43$</td>
<td>4</td>
<td>7</td>
<td>$b_1 = 43$</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>$b_1 &lt; 37$</td>
<td>9664</td>
<td>8</td>
<td>$b_1 &lt; 37$</td>
<td>9664</td>
</tr>
<tr>
<td>9</td>
<td>$b_1 &lt; 34$</td>
<td>333</td>
<td>9</td>
<td>$b_1 &lt; 34$</td>
<td>333</td>
</tr>
<tr>
<td>10</td>
<td>$b_1 &lt; 34$</td>
<td>333</td>
<td>10</td>
<td>$b_1 &lt; 34$</td>
<td>333</td>
</tr>
<tr>
<td>11</td>
<td>$b_1 = 34$</td>
<td>3</td>
<td>11</td>
<td>$b_1 = 34$</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>$b_1 &gt; 34$</td>
<td>9997</td>
<td>12</td>
<td>$b_1 &gt; 34$</td>
<td>9997</td>
</tr>
</tbody>
</table>

(a) Queries before an insertion  (b) Queries after an insertion

Table 2: Three categories of training set queries

<table>
<thead>
<tr>
<th></th>
<th>1st case</th>
<th>2nd case</th>
<th>3rd case</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_1 &gt; x_1$</td>
<td>$b_1 &gt; x_2$</td>
<td>$b_1 &gt; x_3$</td>
<td></td>
</tr>
<tr>
<td>$b_1 &lt; x_1$</td>
<td>$b_1 &lt; x_2$</td>
<td>$b_1 &lt; x_3$</td>
<td></td>
</tr>
<tr>
<td>$b_1 = x_1$</td>
<td>$b_1 = x_2$</td>
<td>$b_1 = x_3$</td>
<td></td>
</tr>
<tr>
<td>$b_1 \neq x_1$</td>
<td>$b_1 \neq x_2$</td>
<td>$b_1 \neq x_3$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Three categories of training set queries

Example in the previous section. We then provide an algorithm to correct the training set of queries in the presence of updates.

The input to our algorithm is a list $L$ of update records, which describe how records of relation $R$ are affected by updates. Consider one of these update records $r$:

$$(x_2, R.b_2, R.b_3, \ldots, R.b_m), \text{ update.type}(r)$$

where $x_2$ is an attribute value of $R.b_1$, $m$ is the total number of attributes of relation $R$ and update.type($r$) denotes the update operation for $r$ (either insert or delete). Assume $x_2$ lies between two other values in the domain of $R.b_1$ (i.e. $x_1 < x_2 < x_3$). There are three possible categories of queries in the training set with respect to $x_1, x_2$ and $x_3$; these are shown in Table 2. The underlined queries in the table are those whose result sizes are affected by any updates, while the sizes of the non-underlined queries are unchanged.

Based on this categorisation, figure 2 gives the algorithm to correct a training set of queries, given a list of update records. The first case in Table 2 is implemented in between lines 7-15 of the algorithm. The second and third cases are fulfilled in between lines 16-24 and 25-32, respectively.

In our example, the correction on the training set in Table 1 was due to the insert record $(x_2 = 34, 3365, 7747)$. The first four queries (1-4) in that table are the first case in Table 2, the second four queries (5-8) are the second case, and the third four queries (9-12) are the third case.

5 Experimental Results

In this section, we study the performance of the ASE method and our machine learning based approach. The machine learning technique described in Section 3 was implemented in a learning machine called M5 developed by Quinlan [15]. Throughout this section, we also refer to our overall query size estimation method by the name M5, even though the M5 machine learning is just one component.

Our experiments investigated two aspects of query size estimation. The first experiment aims to test the convergence rates of M5 and ASE by starting with a very small training set and building up to one that gives accurate size estimations. The second experiment aims to test the efficiency of M5 in the presence of a high update load.

To compare the performance of M5 and ASE with respect to diverse data distributions, we cre-
Figure 1: Relationship between cumulative frequency and query result sizes

(a) Shaded bars representing the result size of query $b_1 > 34$
(b) Shaded bars representing the result size of query $b_1 \leq 34$

Figure 2: Algorithm Update_Training_Set($Q, L$)

1. $Q$ is a training set of queries to be corrected
2. $L$ is a list of records influenced by updates
3. for each record $r \in L$ do
4.   comment
5.   record $r$ contains: $(x_1, R.b_1, R.b_2, \ldots, R.b_m)$
6.   update_type($r$) is either insert or delete
7.   endcomment
8. for each query $q(b_1, op_1, \ldots, b_m) \in Q$ do
9.   if $x < x_2$ then
10.      if relational operator $op_1$ is "$>$" or "$\neq$" then
11.         if update_type($r$) is "insert" then
12.            $x = x + 1$
13.         else
14.            $x = x - 1$
15.         endif
16.      else
17.         if relational operator $op_1$ is "$<" or "$\neq$" then
18.             if update_type($r$) is "insert" then
19.                 $x = x + 1$
20.             else
21.                 $x = x - 1$
22.             endif
23.         endif
24.      else
25.         if relational operator $op_1$ is "$=$" then
26.             if update_type($r$) is "insert" then
27.                 $x = x + 1$
28.             else
29.                 $x = x - 1$
30.             endif
31.         endif
32.     endif
33. else
34.     if $x > x_3$ then
35.         if relational operator $op_1$ is "$<" or "$\neq$" then
36.             if update_type($r$) is "insert" then
37.                 $x = x + 1$
38.             else
39.                 $x = x - 1$
40.             endif
41.         else
42.             if relational operator $op_1$ is "$=$" then
43.                 if update_type($r$) is "insert" then
44.                     $x = x + 1$
45.                 else
46.                     $x = x - 1$
47.                 endif
48.             endif
49.         endif
50.     endif
51. endif
52. endfor
53. endfor

We measure the performance of query size estimation using three separate measures of the discrepancy between estimated and observed values: residual, relative error and absolute error. The residual can be defined as:

$$\text{residual} = |\hat{S}_{q_i} - S_{q_i}|$$

$\hat{S}_{q_i}$ is the estimated result size of query $q_i$ approximated by M5 and ASE and $S_{q_i}$ is the actual result size of the query. The other two measures, relative and absolute errors, were also used in [2] to measure the performance of ASE. The relative and absolute errors can be defined respectively as:

$$\text{rel.err} = \frac{|\hat{S}_{q_i} - S_{q_i}|}{\hat{S}_{q_i}} \times 100$$

$$\text{abs.err} = \frac{|\hat{S}_{q_i} - S_{q_i}|}{\text{card}(R)} \times 100$$
### Notation Meaning

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N(\mu, \sigma) )</td>
<td>normal distribution with mean ( \mu ) and standard deviation ( \sigma )</td>
</tr>
<tr>
<td>( \chi(f) )</td>
<td>chi-square distribution with ( f ) degrees of freedom</td>
</tr>
<tr>
<td>( F(f_1, f_2) )</td>
<td>( F ) distribution with ( f_1 ) freedom for numerator and ( f_2 ) freedom for denominator</td>
</tr>
<tr>
<td>( B(\mu_1, \sigma_1, \mu_2, \sigma_2) )</td>
<td>bi-modal distribution between ( N(\mu_1, \sigma_1) ) and ( N(\mu_2, \sigma_2) )</td>
</tr>
</tbody>
</table>

(a) Distribution notations

<table>
<thead>
<tr>
<th>Distribution</th>
<th>( \sigma_{\text{min}}, \sigma_{\text{max}} )</th>
<th>( \text{card}(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N(200, 150) )</td>
<td>( [150, 150] )</td>
<td>10,000</td>
</tr>
<tr>
<td>( \chi(10) )</td>
<td>( [0, 1200] )</td>
<td>20,000</td>
</tr>
<tr>
<td>( F(10, 4) )</td>
<td>( [0, 800] )</td>
<td>10,000</td>
</tr>
<tr>
<td>( B(250, 150, 450, 50) )</td>
<td>( [150, 650] )</td>
<td>12,500</td>
</tr>
</tbody>
</table>

(b) Data distribution parameters

Table 3: Distribution notations and parameters for different distributions

#### 5.1 Convergence Rates

The aim of this experiment is to see how fast ASE and M5 can converge to a low error. We start by providing both ASE and M5 with very small training sets (to simulate conditions under which they would normally start). We then add test queries to the training sets to simulate the normal process of users posing queries which are executed and whose result sizes are inserted into the training sets.

Initially both ASE and M5 have a training set of 10 queries and 390 unseen test queries are used to test them. We get both systems to estimate the size of an “unseen” test query, and calculate the residual, relative and absolute errors in the estimates. This query then is added to the training sets of both. We repeat this process for all of the test queries until all 390 queries have been added to the training sets.

The results of this experiment are plotted as bar graphs in Figure 3, where each bar represents the average error (either residual, relative error or absolute error) in all of the test queries up to that point. In other words, we compute cumulative mean values for the errors:

\[
\sum_{i=1}^{\text{queries}} \frac{\text{error}}{\text{queries}}
\]

Each graph plots one kind of error for one particular data distribution. All figures are based on the same set of test queries. It is clear from these graphs that, whatever error measure you use, M5 produces a small error than ASE and achieves this with a smaller training set. Even in a pathological case such as subfigure 3(e), where M5 begins with a high relative error, it rapidly reduces this error to an acceptable level.

In addition to the three types of errors (residual, relative and absolute errors), we also plot the actual against estimated values of query result size in Figure 4. Clearly, in such graphs, the ideal is for all points to lie on the diagonal line where actual and estimated values are equal. These graphs emphasize the trend observed in the error plots, that the accuracy of the M5 method is better than the ASE method.

#### 5.2 Updates and Query Size Estimation

Updates can have a major effect on the accuracy of query size estimation. Our experience in using M5 has revealed that without the algorithm Update_Training_Set\((Q, L)\) to correct a training set of queries, the query size estimation will be very poor. The aim of this experiment was to demonstrate that the combination of M5 and our generic update algorithm still performs very well even in the presence of very high update loads.

We follow the update workload specification as used in [2]. The basic idea is to alter the distribution of values of the attribute of interest. An update is specified by:

\[
(i, N, D, [\text{min}, \text{max}], p)_{\text{INS}}
\]

where:

- \( i \) indicates that this update occurs immediately before \( i \)th query in a test set of queries.
- \( N \) is the number of records updated (either deleted or inserted).
- \( D \) is the distribution of values of the attribute of interest. Inserted or deleted records contain those values.
- \([\text{min}, \text{max}]\) is the minimum and maximum, respectively, of the attribute of interest generated by the distribution \( D \).
- \( p_{\text{INS}} \) is a probability with which a record is inserted. Thus a probability that a record is deleted is \( 1 - p_{\text{INS}} \). For instance, if \( N = 10 \) and \( p_{\text{INS}} = 0.2 \), then the number of records to be inserted is 2 \((0.2 \times 10)\) and that to be deleted is 8 \((0.8 \times 10)\).

We used the following four workloads in this experiment:

- **LOAD1:** \((11, 4500, U(-50, 250), [50, 250], 1.0)\)
- **LOAD2:** \((11, 2250, U(-150, 550), [150, 550], 0.75)\), \((17, 2250, U(-150, 550), [150, 550], 0.75)\), \((23, 2250, U(-150, 550), [150, 550], 0.75)\), \((29, 2250, U(-150, 550), [150, 550], 0.75)\)
- **LOAD3:** \((11, 3000, N(-63, 50), [25, 200], 0.1)\), \((17, 1500, N(112, 40), [25, 200], 0.1)\), \((23, 2250, N(290, 60), [200, 375], 1.0)\), \((29, 2250, N(455, 50), [375, 550], 0.4)\)

We used the following test queries in each experiment:

- **LOAD1:** \((11, 4500, U(-50, 250), [50, 250], 1.0)\)
- **LOAD2:** \((11, 2250, U(-150, 550), [150, 550], 0.75)\), \((17, 2250, U(-150, 550), [150, 550], 0.75)\), \((23, 2250, U(-150, 550), [150, 550], 0.75)\), \((29, 2250, U(-150, 550), [150, 550], 0.75)\)
- **LOAD3:** \((11, 3000, N(-63, 50), [25, 200], 0.1)\), \((17, 1500, N(112, 40), [25, 200], 0.1)\), \((23, 2250, N(290, 60), [200, 375], 1.0)\), \((29, 2250, N(455, 50), [375, 550], 0.4)\)
Figure 3: Convergence rates

LOAD4: (11,2250,U(-150,550), [-150,550], 0.8),
(17,2250,N(150,80), [100,300], 0.7),
(23,2250,x(15), [-100,200], 0.5),
(29,2250,F(10, 6), [75,325], 0.6)

The first three were used in [2] and each uses the same data distributions (i.e., uniform distributions for all or normal distributions for all), whereas the last was designed to see how M5 performs in the presence of update workload over a range of different data distributions.

U(z, y) refers to the uniform distribution in the range [x, y], N(μ, σ) the normal distribution with mean μ and standard deviation σ, χ(f) the chi-square distribution with f degrees of freedom, and F(f_1, f_2) the F distribution with f_1 and f_2 degrees of freedom for numerator and denominator, respectively.

Four experiments with the workloads specified above were conducted using the relation with the normal distribution (see Table 3(b)). We used a training set containing 400 queries. Another 40 queries in the test set (the same queries as used in [2]) were used to measure the performance of M5.

The scatter plots of the actual against estimated values of query result size are displayed in Figure 5. One can see from the results that M5 continues to perform well in the presence of updates.

The following work remains to be done to improve our method:

- We need to extend the current machine learning technique to deal with more complex selection criteria and with joins. Several possible approaches to this are presented in [7].
- We also need to demonstrate experimentally that an extension of our method produces accurate query size estimates for these more complex kinds of queries. Initial experiments

6 Conclusions
In this paper, we have achieved the following:

- We have proposed a machine learning technique to solve the problem of query size estimation for a class of selection queries. Using simulation studies, we have demonstrated that its performance is generally superior to the best existing method (ASE) for solving this problem.
- We have proposed a simple generic update algorithm to maintain the training set of queries for our machine learning method. This algorithm can also be used to maintain data structures in other query size estimation methods: the distinct-value-frequency list (zi,f(zi)) in [17] and the query feedback list (li,hi,si) in [2]. This update algorithm improves the performance of all of these methods in the presence of updates.

The following work remains to be done to improve our method:

- We need to extend the current machine learning technique to deal with more complex selection criteria and with joins. Several possible approaches to this are presented in [7].
with more complex selection criteria suggest that our method still performs effectively [7].

Acknowledgements

We owe a debt of thanks to the following: Andrew Taylor pointed us to the M5 learning machine; Zi-jian Zheng provided valuable insight into how M5 constructs a model tree; Stephan Böttcher spent valuable time (when he could have been writing his PhD thesis) to modify his lineno.sty package to enable the line numbering in our algorithm figures.

References


Figure 5: Actual & Estimated Query Sizes for Update Experiment


