Learning Response Time for WebSources using Query Feedback and Application in Query Optimization *

Jean-Robert Gruser
Netforce, Levallois-Perret, France
gruser@netforce.fr

Louiqa Raschid
Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
louiqa@umiacs.umd.edu

Vladimir Zadorozhny
Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
vladimir@umiacs.umd.edu

Abstract

The rapid growth of the Internet and support for interoperability protocols has increased the number of Web accessible sources, WebSources. Current optimization technology for wrapper mediator architectures needs to be extended to estimate the response time (delays) to access WebSources and to use this delay in query optimization. In this paper, we present a Multi-Dimensional Table (MDT), a tool that is based on learning using query feedback from WebSources. We describe the MDT learning algorithms, and report on the MDT learning for WebSources. The MDT uses dimensions Time of day, Day, and Quantity of data, to learn response times from a particular WebSource, and to predict the expected response time (delay), and a confidence in this prediction, for some query. Experiment data was collected from several WebSources and analyzed, to determine those dimensions that were significant in estimating the response time for particular WebSources. Our research shows that we can improve the quality of learning by tuning the MDT features, e.g., including significant dimensions in the MDT, or changing the ordering of dimensions. We then demonstrate how the MDT prediction of delay may be used by a scrambling enabled optimizer. A scrambling algorithm identifies some critical points of delay, where it makes a decision to scramble (modify) a plan, to attempt to hide the expected delay by computing some other part of the plan that is unaffected by the delay. We explore the space of real delay at a WebSource, versus the MDT prediction of this delay, with respect to critical points of delay in specific plans. We identify those cases where MDT overestimation or underestimation of the real delay results in a penalty in the scrambling enabled optimizer, and those cases where there is no penalty. Using the experimental data and MDT learning, we test how good the MDT is in minimizing these penalties.

1 Introduction

Architectures based on wrappers and mediators [27] have been proposed in [1, 6, 13, 17, 18, 19], to provide access to data in heterogeneous sources. In such an architecture, wrappers handle query processing on individual sources. The mediator solves the task of capability based rewriting (CBR) to determine the

*This research has been partially supported by the Defense Advanced Research Project Agency under grant 01-5-28838; the National Science Foundation under grant IRI9630102, and INRIA Rocquencourt, France.

(sub)query to be sent to a source, depending on the capability of each source. A mediator also handles query optimization for wrapper (sub)queries and the mediator (composition) query. The rapid growth of the Internet and Intranets, vendor support of database interoperability protocols such as JDBC [14], and OLE/DB [4], and the emergence of formats such as XML-Data [12], that facilitate the exchange of data via the WWW and the HTTP protocol, has dramatically increased the number of available Web accessible sources, WebSources.

Scaling a mediator architecture to deal with WebSources introduces the challenge of correctly estimating the response time, or delay, in accessing data from a WebSource. Developing a cost model for WebSources must deal with the following drawback: there is a lack of accurate statistics, e.g., selectivity estimates for queries; knowledge about load on the server, access paths, and the cost of physical algorithms executed on WebSources. There is also little knowledge about the impact that dimensions such as time of day, day, network topology, etc., can have on the time to transfer the results.

There has been some research on wide area traffic patterns for the Internet [21]. In [26], statistical models are applied to measurements of service requests at proxy servers, to detect failure patterns. Models using metrics such as number of hops, *ping* timing, and `http` request service times have been studied, to compare performance among replication servers [20]. Our research is in a similar spirit; we use learning based on query feedback, to predict response times from a particular WebSource. There is increasing interest in developing benchmarks to compare WebSource performance [8, 23]. The parameters that are used here are low level network and system parameters, and are specific to the server. They do not model the client or predict delays at the client. The *Network Weather Service*, NWS [28], is a general facility that provides dynamic resource performance forecasts for wide area networks. It uses intrusive resource monitoring; a distributed set of sensors gather data on current network and server conditions. Such data could also be used to predict response time (delay) at WebSources.

Several solutions have been proposed for mediator query optimization; however they have not considered the characteristics of WebSources. Research reported in [5, 7] assumes that calibration databases can be constructed on remote sources, i.e. they accept updates. A generic cost-model is calibrated by experiments on a calibrating database created in each source. Unfortunately, most WebSources do not accept updates. The DISCO project [22] contacts wrappers to get the cost of each plan. DISCO assumes that the wrapper for each source provides a description of the available physical operators and their corresponding costs. However, most WebSources do not model or communicate such information. Research reported in [25] also assumes that costs for accessing data from sources is known a priori. The approach used by the HERMES system [1] can be adapted to model WebSources, since their model uses only query feedback. However, they do not develop a robust model for learning and prediction that can handle the unpredictable nature of wide area networks. To summarize, all these solutions for developing mediator cost models have the drawback that they either expect unavailable information from WebSources, or they do not deal with the somewhat unpredictable behavior of WebSources, due to unpredictable loads on the source and network, noise, etc.

There are two main contributions of our research, to solve the problem of query optimization in a mediator with WebSources. The first contribution is the development of a tool, a Multi-Dimensional Table or MDT, that uses learning based on query feedback (response time) to predict the response time for a query in a *particular* WebSource. Unlike models based on low level network and server parameters, the MDT learning is at a high level, using dimensions Time of day, Day, and Quantity of data transferred, to predict the response time for a particular query on a particular WebSource, and to determine a *confidence* in this prediction. The MDT approach has some advantages over other learning based techniques such as regression techniques or neural networks. One advantage is the simplicity of the MDT prediction model. The second is the flexibility provided by the MDT to manipulate a number of parameters that control learning. The third advantage is that while the dimensions chosen may reflect the effects of source and network usage, a (lack of) confidence reflects the unpredictable nature of prediction for WebSources.

Query feedback was obtained from a number of WebSources. Statistical tests were used to determine those dimensions that were indeed significant in predicting the response time (delay) for a particular source. The MDT was then trained on the collected data. Our experimental study shows two significant results with respect to MDT learning. The first result is that the MDT does learn, and that as it is trained, the (cumulative) error decreases, and the confidence in the prediction increases. The second result is that we can improve the quality of learning by tuning the MDT features, such as including significant dimensions in the MDT, or changing the ordering of significant dimensions in the MDT.

The second contribution of our research is a study of how the delay or response time prediction by the MDT, and the confidence in the prediction, may be used to enhance a traditional query optimizer. We note that in this research, we do not distinguish between initial delay and response time (time to get the first answer); we hope to do so in future work. If an optimizer had perfect knowledge of the delay, then it will always identify the best plan. However, this knowledge is typically unavailable. A scrambling enabled optimizer makes optimization decisions during run-time, using estimated delays [24]. Such an optimizer has some *critical points* for each plan, where it makes a decision to scramble (modify) the plan, to attempt to cover the expected delay at a WebSource, by computing some other part of the plan that is unaffected by the delay. We explore the space of real delay versus the MDT prediction of this delay, with respect to critical points of delay in specific plans. We identify those cases where MDT overestimation or underestimation of the real delay is unsafe, and incurs a penalty by the scrambling enabled optimizer, and those cases where the prediction error is safe, and there is no penalty. Using the experimental data and MDT learning, we test how good the MDT is in minimizing these penalties.

This paper is organized as follows: In section 2, we describe the MDT structure and features to tune its learning. We then describe the MDT learning process; and the technique for predicting the response time and the confidence in the prediction. In section 3, we describe the experimental data collection task, and the analysis of the data, to determine those dimensions that are significant for particular WebSources. In section 4, we report on training the MDT, and the results of MDT learning on WebSources. We also compare with related work on learning and prediction. In section 5.1, we describe the technique of query scrambling, and discuss critical points during scrambling. In section 5.2, we discuss the impact of MDT overestimation and underestimation of the delay; the cases in which this can be done safely without incurring a penalty; and the cases in which there is a penalty. In section 5.3, we use the MDT prediction of delay, together with our experimental data, to determine how good the MDT is in minimizing these penalties. Section 6 concludes.

2 The Multi-Dimensional Table (MDT)

We use a parameterized *Multi-Dimensions Table* (MDT), to collect response times based on query feedback, and we use a simple learning technique to predict the response time for some query. For each prediction, the MDT will also determine the *confidence* in that prediction. We describe the structure and dimensions of the MDT, and the features that are used to tune the learning algorithm. We then describe the learning algorithm and explain how the predicted response time and the confidence in the prediction is determined.

2.1 Structure of the MDT and Features for Tuning

The structure of the MDT is determined by (1) a set of *dimensions*; (2) the *ordering* of these dimensions; and (3) the *ranges / scales* of these dimensions. The MDT that we use in this research has three dimensions, Quantity of data, Day and Time. The dimensions and their ranges / scale are as follows:

- The *Day* of the week. This dimension has a range of seven days, and the minimum scale for this dimension was chosen as one day.
- The *Time* of day. This dimension has a range of 24 hours, and the minimum scale for this dimension was chosen as one hour.
- The *Quantity* of data that is transferred. This dimension does not have a fixed range. Based on our experimental data, the minimum scale that we chose was multiples of 100 Kilobytes, and the range was from [0 - 800 Kilobytes].

The MDT structure is determined by the *ordering* of these dimensions and in our experiments in a later section, we will discuss the impact of the ordering on the MDT learning . The initial MDT consists of one *cell*, and the range of its dimensions correspond to the ranges described above. During the learning process, each query feedback qfb is represented by a value for Time, Day and Quantity, and response time $QryRT$. The values for Time, Day and Quantity will be matched to identify the corresponding *cell* of the MDT. A predicted response time $PredRT$ and a confidence $pred\_conf$ are also associated with that cell. The MDT learning process may then decide, based on the query response time, $QryRT$ of qfb, to either split this cell, into two or more cells, or to adjust the $PredRT$ and the confidence for this cell. Details of the learning algorithm will be described shortly. For simplicity, we assume that except in the case of the dimension *Day*,

Monday-Friday								Saturday-Sunday
8am-2pm				2pm-8pm			8pm-8 am	12am-12am
<200K	200K-400K	400K-600K	>600K	<200K	200K-400K	>400K	0-MAX	0-MAX

Figure 1: An Intermediate MDT Structure

the cell splitting will divide the range of the selected dimension into two equal ranges[1]. The minimum scale of each of the dimensions will determine the (final) MDT structure, when it can no longer split into more cells on any dimension. For example, when the Time dimension has been split into 24 cells, each with a one hour range, then no further splitting on that dimension is possible. An example of an MDT structure, at some intermediate point, is in Figure 1. Here the ordering is Day-Time-Quantity, where Day is most significant.

The ordering of dimensions and scale of the dimensions can be used to control (tune) the learning process. Three other features that are also used to control the learning are as follows:

- The allowed deviation dev of the error in response time. This value of dev is specified for each dimension. The error $err = \frac{|QryRT - PredRT|}{QryRT}$, where $PredRT$ is the MDT prediction for some cell, and $QryRT$ is the response time of some query feedback qfb that matches that cell. The allowed deviation for each dimension is used to determine if the matching cell should be split on that dimension, when qfb is received, and will be explained later.

- The precision $prec$ for each dimension. The $prec = 1 - \frac{Range.max - Range.min}{Cell.max - Cell.min}$. The smaller the range of the individual cell, compared to the range of the dimension, the greater is the precision. The initial precision for the initial single cell MDT is 0 for each dimension. The precision will be a factor in determining the confidence in the predicted response time for that cell.

- The confidence window $confwin$ which is selected for each dimension. Confidence is in the range [0.0 - 1.0] and the confidence window can be selected within that range. A typical window may be [0.3, 0.7].

2.2 Learning in the MDT based on Query Feedback

Figure 2 describes the simple learning algorithm for the MDT. We now describe the learning algorithm. Each query feedback qfb is described by a value for Day, Time, and Quantity. It also has a response time $QryRT$. The matching MDT $cell$ whose dimensions match qfb is identified; it is described by a current prediction $PredRT$, and current confidence $pred\_conf$, for each dimension. We determine the error, for that cell. Recall that the MDT is described by an ordering ord of the dimensions, and each of these dimensions is described by an allowed deviation dev and a precision $prec$ and a confidence window $confwin$.

[1] In our implementation, cell splitting is often more complicated and the two cells may not always have equal ranges after splitting.

Learning Algorithm(qfb[day,time,qty,QryRT])
For the cell[PredRT,pred_conf] of the MDT whose dimensions match Day, Time and Qty values of qfb
  For each dimension dim[dev,prec,confwin] in the ordering *ord*
    If Within_Deviation(qfb,cell,dim) correct(cell.$PredRT$, cell.$pred\_conf$)
    Else split_MDT(qfb,cell,dim)

Figure 2: The Learning Algorithm

Starting from the most significant dimension in *ord*, we compare the error *err* of that qfb, with the allowed deviation *dev*, for that dimension. Suppose *err* is more than *dev*, for that dimension. Then, if it is possible, i.e., the cell is not at the minimum scale for that dimension, the *cell* is split on that dimension. Recall that for simplicity, we assume that the split is into two cells, and each cell has equal range. Only one of the (split) cells will now match the dimensions of the qfb. The new *PredRT* for this new cell is set to *QryRT* and the new *pred_conf* is 0. The values of *PredRT* and *pred_conf* of the other split cell remain unchanged since the qfb no longer matches that cell.

If indeed there is a split on the dimension, then the learning algorithm will be called recursively, for each subsequent dimension, in decreasing order of *ord*. Again, depending on *dev* for each dimension, a decision must be made whether to split the new cell, on the next most significant dimension, into further new cells. Consider the MDT structure of Figure 1. Suppose there is a qfb on Saturday at 1am, and the calculated *err* is greater than *dev* for both Day and Time. Then, the learning algorithm will first split the cell `[(Saturday-Sunday),(12am-12am)]` on the dimension Day, and create two new cells, `[(Saturday),(12am-12am)]` and `[(Sunday),(12am-12am)]`. Next, the algorithm will split the dimension Time of the first new cell, and create two new cells `[(Saturday),(12am-12pm)]` and `[(Saturday),(12pm-12am)]`.

Suppose instead that *err* is less than the allowed deviation *dev* of some dimension of that cell, or that the cell cannot be split any further on that dimension. In this case, we adjust the *PredRT* and *pred_conf* for that cell to reflect the qfb. First, we calculate the confidence in the new *QryRT* of qfb.

Each MDT cell will have a buffer that stores the last N values of qfb that matched this cell. Using the value of *err* for the current dimension, we count the number of matching qfb in the buffer. A matching qfb is one such that the difference in *QryRT* between qfb in the buffer and the new qfb, normalized by *QryRT* for new qfb, $= \frac{|bufferqfb.QryRT - newqfb.QrtRT|}{newqfb.QryRT}$ is less than *err*. This count is $NumMatchingBuffer$. Then, the confidence in $QryRT = (NumMatchingBuffer * precision)$.

Once this *qry_conf* is determined, then the *PredRT* and the *pred_conf* for the cell must be recalculated to reflect the new qfb values. We discuss several cases as follows:

- Both *pred_conf* and *qry_conf* are low and below $confwin$, the confidence window for the dimension. This typically occurs in the initial learning stages or when there is noise. The adjusted value of

$PredRT = \frac{pred\_conf * PredRT + qry\_conf * QryRT}{pred\_conf + qry\_conf}$ is calculated.

If err is large, then there is little confidence in the estimated value $PredRT$ or the new value $QryRT$. So the confidence for that cell is adjusted to the min($pred\_conf$,qry_conf).

However, if the err is small, then both $QryRT$ and $PredRT$ are close to each other, and the confidence in that cell's prediction should be increased. The confidence $pred\_conf$ is increased using a weighted average and $pred\_conf = \frac{pred\_conf * NumberOfValues + qry\_conf}{NumberOfValues + 1}$, where $NumberOfValues$ is the number of prior qfb values that were used to learn the old $PredRT$ and confidence. Using a weighted average slows MDT learning, but has the advantage of making the MDT less sensitive to noise.

- The qry_conf is below $confwin$ and $pred\_conf$ is in $confwin$ or higher. This occurs when there is some spurious noise, after some learning. We say there has been learning since $pred\_conf$ is high. The $PredRT$ and $pred\_conf$ are adjusted using the formula described previously, where err is small. We note that since qry_conf is below $confwin$, the adjusted $pred\_conf$ will be lower than before receiving qfb. However, if this new qfb is indeed noise, then few values in the buffer for that cell will match this new qfb and so $NumMatchingBuffer$ and qry_conf will be low. Thus, $pred\_conf$ will reduce slowly and $PredRT$ will also change slowly. However, if there has been a long burst of noise, then the $NumMatchingBuffer$ may be higher, and qry_conf may be low. This has the potential to interfere with MDT learning.

- Both $pred\_conf$ and qry_conf are high and within or above $confwin$. This is a refining stage of the learning and the new $pred\_conf$ will be increased. The $PredRT$ and $pred\_conf$ are adjusted using the formula described previously for small err.

- There are several cases that should not occur. For example, it cannot be the case that both $pred\_conf$ and qry_conf are high and within or above $confwin$, but err is large. This would indicate that the MDT has entered an unstable state. We do not provide details here but refer the reader to [9].

The learning behavior of the MDT was studied using simulation, based on various distributions for the response time of qfb. For lack of space we do not include the results here, and refer the reader to [9].

3 Experiment Data Collection and Analysis for WebSources

Experimental qfb were collected from various sources, to be used in an experimental evaluation of the MDT. Our experiment used the Java `URLConnection` class [15], and this class used the `http` protocol to download files from WebSources. We set the flag `setUseCaches` to false, so that the file would always be loaded from the WebSource. We used the Java class `Calendar` to time the experiment. We accessed two kinds of sources. One class was university servers; their country location, server URL, and the label to refer this WebSource is as follows: {Brazil `www.lbd.dcc.ufmg.br` (BR), Australia `broncho.ct.monash.edu.au` (OZ), Canada `www.cs.toronto.edu` (UT), France `www-rodin.inria.fr` (INR), USA `www.umiacs.umd.edu` (UM)}. For these servers, the impact of network load was typically more severe, compared to the number of users on the servers. Brazil and Australia had typically much longer response times. From these servers, we were able to request files of pre-selected size ranging from 100 Kilobytes to 800 Kilobytes. The second class of server was commercial servers. They included the NBC news server, `www.msnbc.com` (NBC), which is aliased to four servers on the same subnet; the news server for the Le Monde French daily, `www.lemonde.fr` (LeM); and a weather server, `www.weather.com` (WTH), which is aliased to two servers on different networks. Although these URLs were sometimes aliased to multiple physical servers, we made a decision to treat a server URL as a *single* WebSource, and estimate the response time at the client, for the single logical WebSource[2]. For the commercial servers, user load was also a factor that could affect response time. Since we were not able to place pre-selected files on these servers, we identified *gif* files varying in size from 50 Kilobytes to 800 Kilobytes. However, on some servers, e.g., `www.lemonde.fr`, we could not locate many large files, and this is reflected in our analysis of the data.

The data collection experiment was straightforward. At timed intervals, our data collection program randomly selected one or more files located at the source, downloaded the file, and recorded the response time, i.e., the elapsed time to download the page, using the `http` protocol, and to stream the page into program memory. The client was a machine within the University of Maryland network cluster `umiacs.umd.edu`. The data was collected over several weeks, from June - October 1998.

Figure 3 shows some sample response time data collected from `OZ` for some 3100+ qfb. It has been sorted by (increasing) Quantity. The emerging pattern indicates that the dimension Quantity is significant with respect to the response time.

We analyzed the collected data using the χ^2 contingency test for categorical data, to determine if the dimensions Day, Quantity and Time, were significant, with respect to the response time. We performed the test on the pairs of variables, Day and response time, Quantity and response time, etc. We briefly describe how the data was pre-analyzed to prepare it for testing, and then present the summarized results of our analysis in Figure 4.

The test required us to prepare contingency tables, where one axis of the table was the response time and the other axis was the MDT dimension. Since the χ^2 test is applied to categorical data, we needed to identify appropriate categories for each dimension and for the response time. The number of categories for the MDT dimension Day was 7, and it was 8 for Time, where we considered contiguous 3 hour blocks from 12 am to midnight [3]. Determining the categories for Quantity was more complex. In some sites, we had 8 categories, [less than 100 Kilobyes, 100 to 200 KiloBytes, $\cdots$, greater than 700 Kilobytes]. In other sites, where all the files were less than 100 Kilobytes, we could not test the effect of this dimension.

The most critical task was identifying the categories for the response time, since the response times varied widely among all the sources. Our first step in identifying the categories was eliminating outlier data. This

[2] This appeared to be reasonable since most clients do not differentiate the various physical servers aliased to a URL.

[3] We chose a 3 hour block for the dimension Time in order to have a similar number of categories in each dimension.

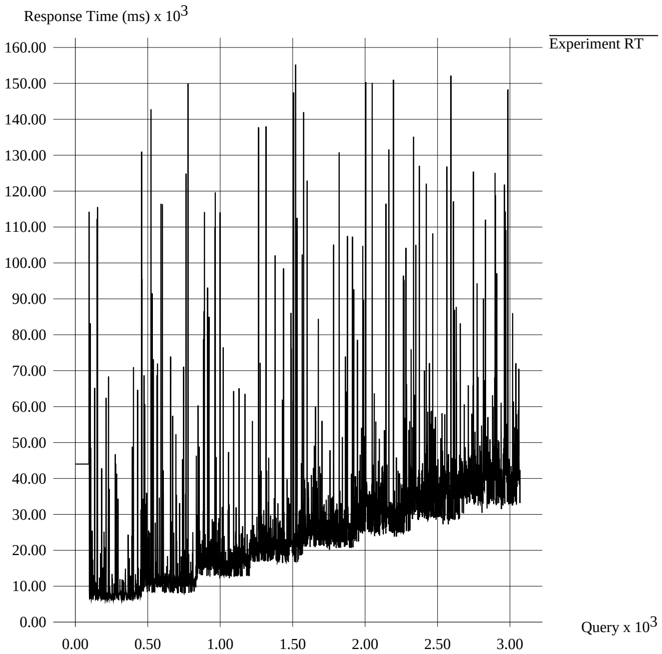

Figure 3: Response Time Sorted by Quantity for WebSource `OZ`

Source	Dimension					
	Day		Time		Quantity	
	O1	O2	O1	O2	O1	O2
BR	NO	NO	**YES**	**YES**	**YES**	**YES**
INR	?	**YES**	?	**YES**	**YES**	**YES**
LeM	**YES**	**YES**	?	**YES**	N/A	N/A
OZ	NO	**YES**	NO	NO	**YES**	**YES**
UM	NO	NO	NO	NO	**YES**	**YES**
UT	NO	NO	NO	**YES**	**YES**	**YES**
WTH	?	**YES**	**YES**	**YES**	N/A	N/A

Source	Dimension			
	Day		Time	
	O3	O4	O3	O4
BR	**YES**	NO	**YES**	**YES**
INR	**YES**	**YES**	**YES**	**YES**
LeM	N/A	**YES**	N/A	**YES**
OZ	**YES**	**YES**	?	NO
UM	NO	?	NO	NO
UT	?	NO	?	**YES**
WTH	NO	**YES**	**YES**	**YES**

? -- May be significant
N/A -- The sample data set is too small to determine significance

Figure 4: Table Summarizing the Statistical Analysis of Experiment Data

was either the very small or the very large values. We had a large number of cases where the `http` request timed out. The outlier data was eliminated recursively, starting from both the largest and the smallest values. Once the typical range [min, max] was identified for that source, we divided the response time into 3 categories, small, medium and large. The division of the range is identified by 3 integers, e.g., 30-30-40. Thus, the small response time are values in the range [0, min + (max-min)*.3], etc. We tested our data with a variety of ranges for the integers, e.g., 25-50-25, to determine sensitivity to this choice. The degrees of freedom for the contingency tables varied, depending on the dimension. We used an α of 0.01 to make a determination of significance, i.e., a level of 99% confidence in the test, and we tested that we had sufficient sample size in each cell of the contingency table.

We performed four analyses of the data, leading to four sets of observations, labeled *O1* to *O4* in Figure 4. In observations *O1*, we did not consider outlier data outside the typical range of [min,max] for the WebSource. However, in many cases, when the request timed out, it implied that the response time was indeed very large. Thus, in the observations labeled `O2`, we considered all the timed out instances as large response times. In Figure 4, we identify when a dimension is significant (YES) or is not significant (NO), in predicting the response time, for the source. A symbol `?` in the table indicates that we could not determine if the dimension was significant. A `N/A` value indicates that this test could not be performed due to lack of sufficient data[4].

For all of the sources, the dimension Quantity was significant [5]. In observation `O1`, Time was significant for `BR` and `WTH`. Day was not significant for any of the WebSources. When we also considered outlier data in `O2`, additional dimensions became significant. The dimension Day was significant for `INR`, `LeM`, `OZ` and `WTH`, and Time was significant for all WebSources except `OZ` and `UM`. The χ^2 test value for Quantity indicated that the significance of this dimension could overshadow the other dimensions. Thus, in `O3`, we only considered response times for large files, and in `O4`, we only considered response times for small files. The results indicate that when we minimized the effect of Quantity, the significance of Time and Day was more clearly identified, e.g, for INRIA, a may be significant result for dimension Day and Time in `O1` was proved to be a yes in `O3` and `O4`. For `OZ` a not significant result on Day in `O1` became significant in `O3` and `O4`.

Details of the χ^2 contingency test values from which we drew our conclusions, for observation `O2` are in Appendix A. Details of the complete data analysis are in [9].

4 Results of MDT Learning with Experimental Data

We describe the results of MDT learning, to predict the response time, for the WebSources described above. Our experimental study shows two significant results. The first result is that the MDT does learn, and that as it is trained, the prediction error decreases, and the confidence in the prediction increases. The second result

[4] We omit the analysis of data from `NBC` since the sample size was too small.

[5] For `LeM` we could not perform this analysis, since the files were small.

is that we can improve the quality of learning by tuning the MDT features. The MDT learning is improved with the inclusion of those dimensions that are found to be significant, for some source, determined in the previous analysis of the data, or a *good* ordering of those dimensions that are significant. Conversely, the MDT learning is worse when significant dimensions are not included, or with a *poor* ordering. We summarize the more interesting results, and details are in [9].

We use the (absolute) relative error $\frac{|QryRT - PredRT|}{QryRT}$ to track MDT learning. A better indicator to characterize the learning process is the cumulative mean squared relative error ($msre$). This expression is $\frac{\Sigma(\frac{QryRT - PredRT}{QryRT})^2}{N}$, where N is the number of predictions. We also use the confidence in the prediction $pred\_conf$, which was discussed previously.

4.1 MDT Learning

Our first result that the MDT does learn is shown in Figure 5a, which shows relative error, and in Figure 5b, which shows cumulative $msre$. The MDT is trained on approximately 3100+ experimental qfb from the WebSource labeled `OZ`, i.e., queries from the United States to Australia. Figure 6a has the corresponding confidence values for the MDT learning on these queries. As can be seen, both relative error and cumulative $msre$ are initially quite significant, as the MDT starts learning. The corresponding confidence in the prediction also swings very rapidly, indicating that the MDT predictions are not in a stable state. However, after about a 1000 qfb, the cumulative $msre$ gradually stabilizes. The confidence in the prediction correspondingly increases and does not vary as much. After about 2500 qfb, the confidence in the prediction varies between [0.90 - 0.95], indicating that the MDT is very confident in its prediction. The cumulative $msre$ has almost leveled off, indicating that learning is occurring very slowly at this point and the MDT learning has stabilized. Figure 6b shows cumulative $msre$ during MDT learning for WebSources `UT` and `UM`. A similar pattern of MDT learning is observed.

In the `OZ` experiment, the MDT used two dimensions, Quantity and Day. We note that an MDT with three dimensions, and the range and scale used in our experiments, has a maximum of 1344 cells, in the innermost dimension or the ordering, if the (deviation of the) value of qfb response time had led to maximum splitting on all dimensions. With a training set of approximately 3100+ qfb, as in `OZ`, there is an average of less than three qfb per cell, and this a sparse training set. Typically, the MDT does not split to the maximum number of cells.

4.2 Effect of Ordering of the Dimensions on Learning

We now show that a correct ordering of dimensions, that matches those dimensions found to be significant for some WebSource, can improve MDT learning. Omission of those dimensions that are found to be significant, or a poor ordering, has a negative impact on MDT learning.

We consider the WebSource `OZ`, for which Quantity and Day were the most significant dimensions. Figure 7a shows the MDT learning, using the cumulative $msre$, for ordering Q-D, i.e., when Quantity is the most

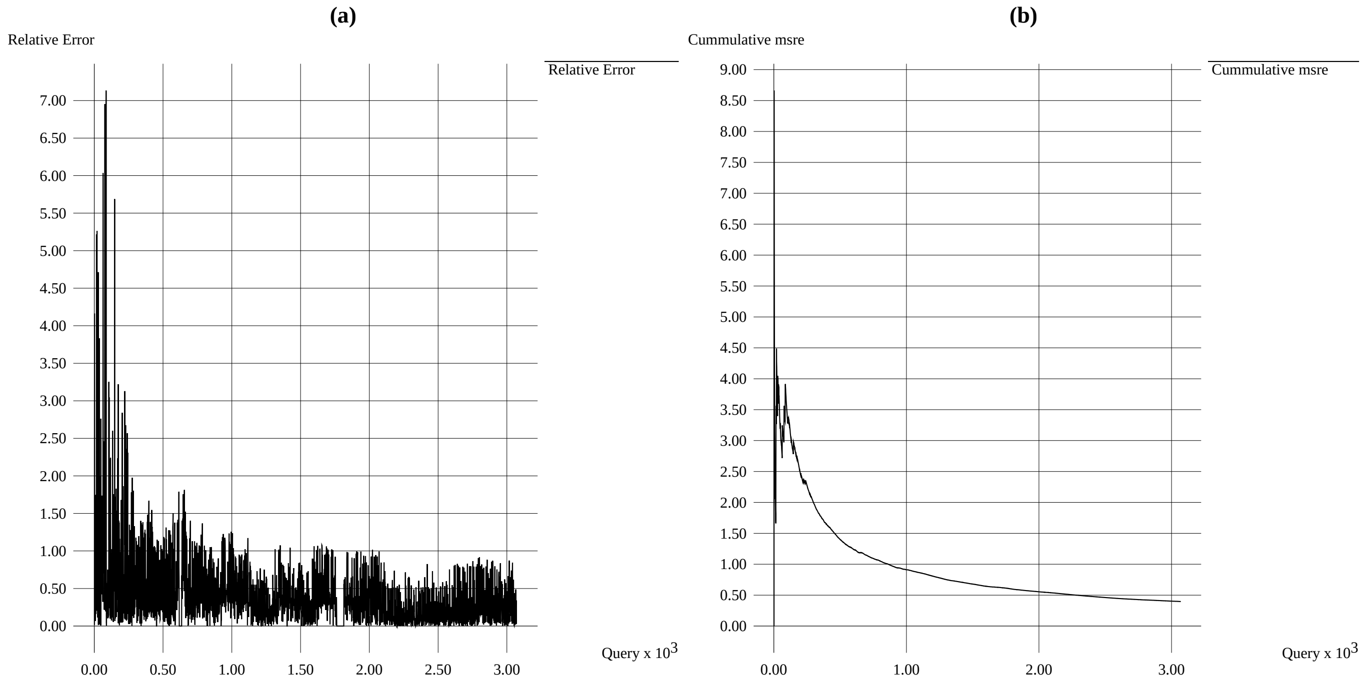

Figure 5: Relative Error and Cumulative *msre* during MDT Learning for WebSource OZ

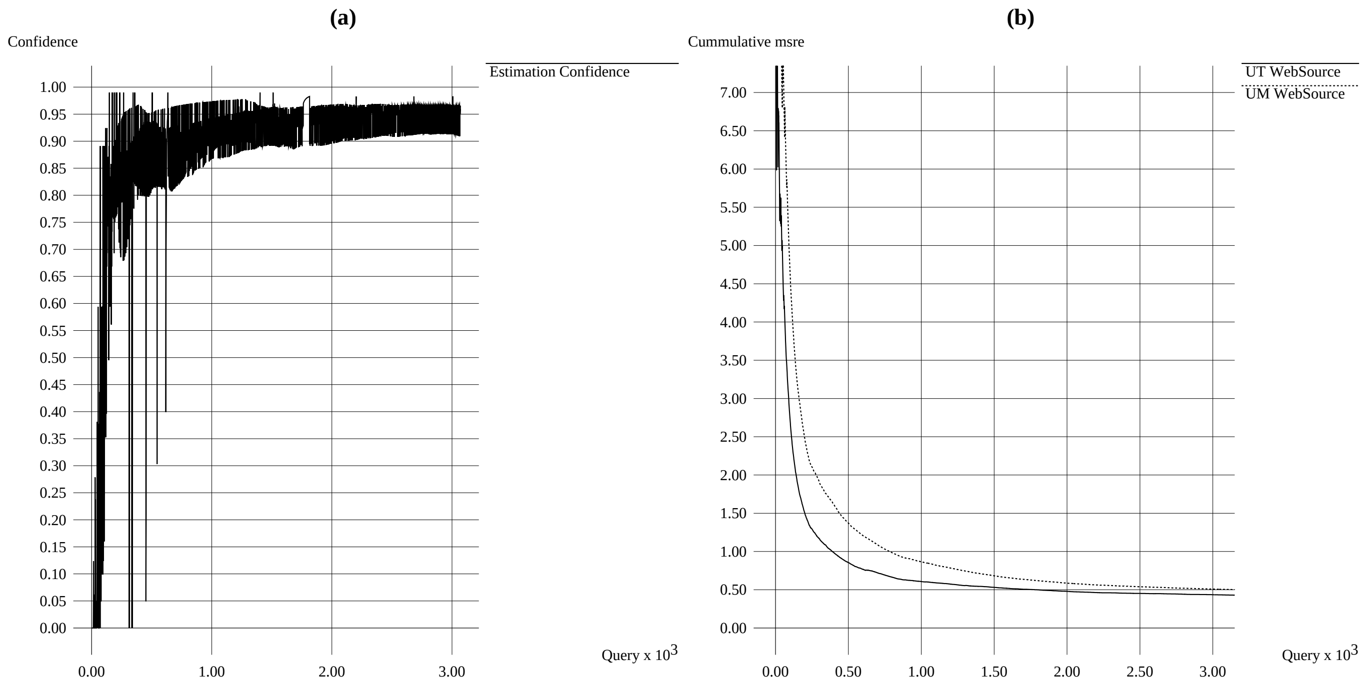

Figure 6: Confidence in MDT Prediction for WebSource OZ (a) and Cumulative *msre* for WebSources UT and UM (b)

significant dimension, and then for D-Q. As is clearly indicated, the ordering Q-D is superior. Our statistical analysis of the data indicates that Quantity is the most significant dimension, and has the greatest impact on predicting the response time, in particular for sources OZ and BR, where response times are often very large. This figure shows that the poor ordering D-Q, with respect to the significant dimension Quantity, had a negative impact on the MDT learning.

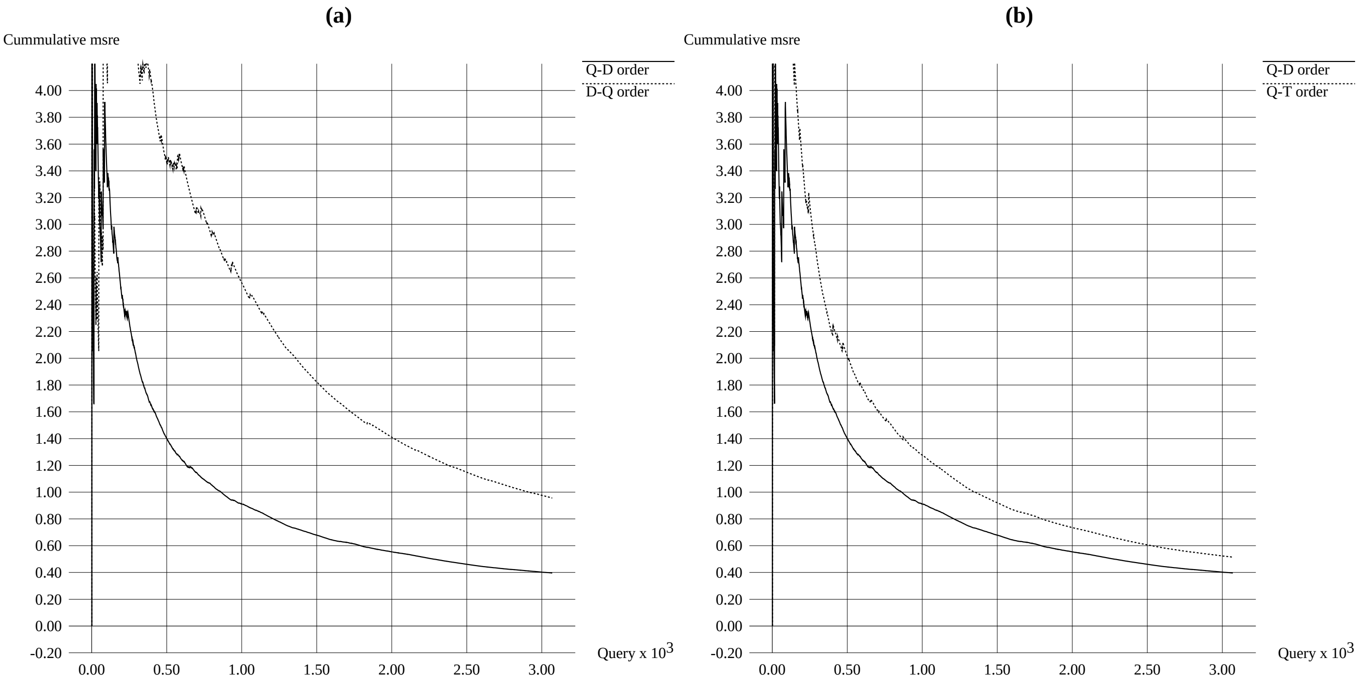

Figure 7: Comparison of Ordering Q-D, D-Q (a) and Q-D, Q-T (b) on MDT Prediction for OZ

Figure 7b shows the MDT learning for the ordering Q-D and compares it with the ordering Q-T, using the cumulative *msre*. Recall that Day was also a significant dimension for the dataset OZ. Thus, the prediction of ordering Q-T, which does not consider the dimension Day, is somewhat poor, compared to the MDT with ordering Q-D, which does consider the dimension Day. This indicates that omitting a significant dimension, Day, has a negative impact on the MDT learning. We further note that compared to Figure 7a, the MDT with ordering Q-T performs better than the MDT with ordering D-Q. This shows that the omission of a significant dimension, Day, appears to have less impact, compared to the error of a poor ordering D-Q, with respect to the very significant dimension, Quantity. This is consistent with our understanding of significance of the dimensions in prediction.

Finally, in Figure 8, we display the MDT with orderings Q-D-T, Q-T-D and Q-D. The MDT with ordering Q-D-T performed better than Q-D. While our statistical tests did not provide a clear indication that the dimension Time was significant for OZ, our MDT learning seems to indicate that the inclusion of this dimension does provide a slight benefit in the learning. We also see the effect of a poor ordering. While the ordering Q-T-D includes the two significant dimensions Quantity and Day, there is a poor ordering of

Day with respect to Time, i.e., Q-T-D instead of Q-D-T, since Day is more significant. This has a negative effect on the learning. Thus, the MDT with ordering Q-D performed better than the MDT with ordering Q-T-D, since the ordering Q-D is superior to Q-T.

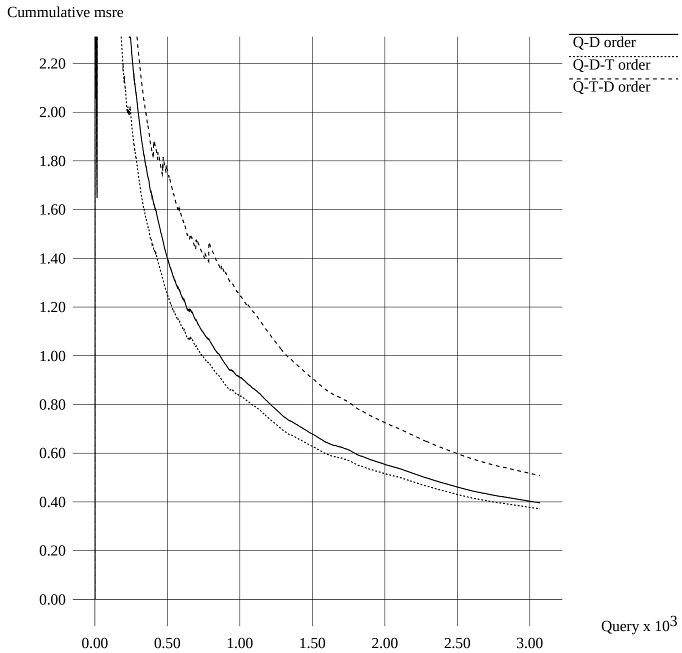

Figure 8: Comparison of Ordering Q-D, Q-D-T and Q-T-D on MDT for OZ

4.3 Evaluation of the Trained MDT

Finally, we tested the MDT performance on test data, after training it on some training data. Both test and training data were obtained in a similar manner during our data collection. For this experiment, we trained the MDT by sending it some initial qfb from the WebSource. Once the MDT is trained, we no longer allowed the MDT to learn by sending it qfb. Figure 9a shows the relative error after the MDT was trained on 3100+ qfb. Figure 9b shows the cumulative $msre$ for different sizes (approximately 300, 1400 and 2900 qfb) of training data.

As indicated in Figure 9b, when the training data was fairly small, e.g., 300 qfb, the MDT is not stable and the error is greater. Our previous experiment indicated that the MDT started stabilizing after approximately 1000 qfb. Thus, the MDT prediction improves after training it on larger sets of qfb.

4.4 Comparison with other Learning Techniques

We now consider other learning techniques. Compared to the MDT learning, neural networks [10] typically exhibit sophisticated learning behavior. However, a neural network is very sensitive to the training data, since it does not allow direct manipulation of the training process. In contrast, the MDT learning, while simple,

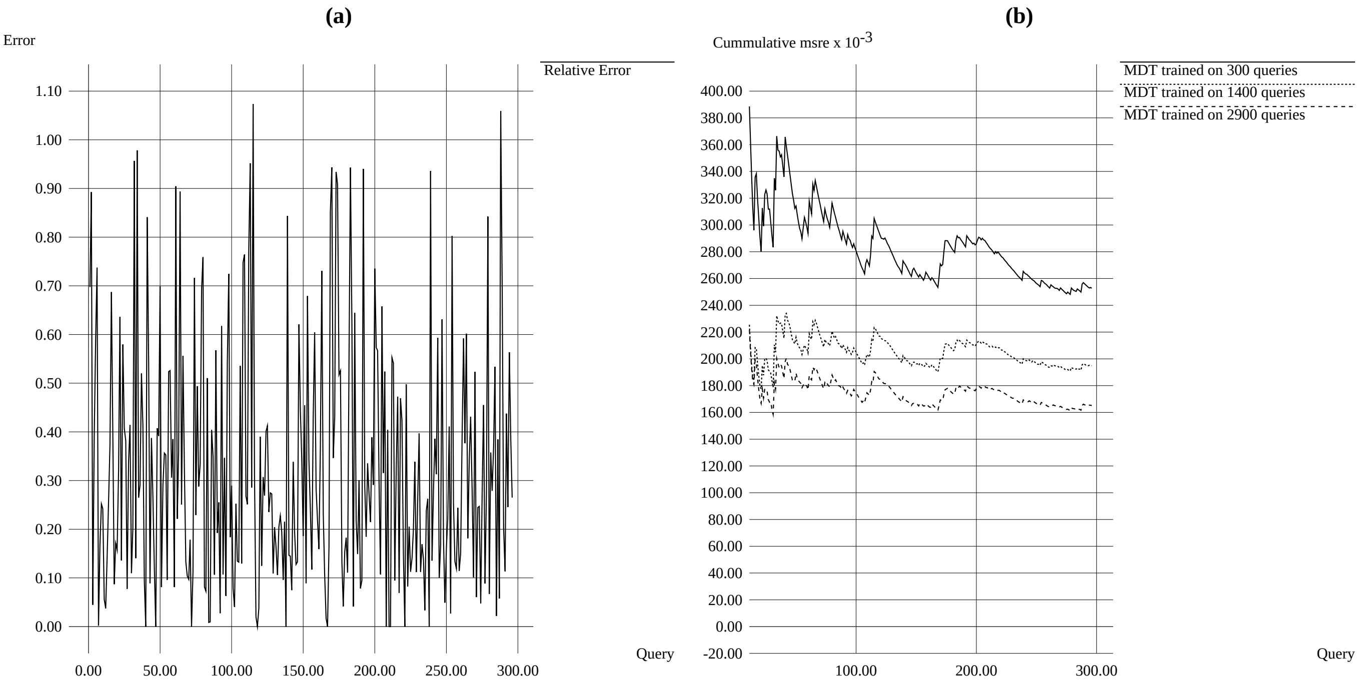

Figure 9: Testing the MDT Prediction After Learning - Relative Error and Cumulative $msre$

allowed us to directly manipulate features that controlled the learning, enabling us to better understand the behavior of the experimental data itself. In future work, we plan to train a standard back propagation neural network with our experimental data, to make an empirical comparison with MDT learning.

CART [3] is a classification and regression algorithm. Using a set of classification variables, it adopts a binary recursive splitting, and successively partitions the data into discrete subgroups, based on each possibly relevant variable, until further splitting is infeasible. CART is also sensitive to the ordering of variables. In comparison, the MDT learning is less complex, since it identifies a split directly, using a fixed allowed deviation value for each dimension. This is less costly than the regression that is performed by CART to determine the split. We further note that the MDT can split a range into multiple subranges (cells), and it provides other tuning features such as precision of each dimension. In addition, the MDT uses the confidence window, and the buffer of qfb in each cell, to overcome the effects of noise.

The MDT learning resembles the classification task to some extent, in that the splitting tends to cluster the data on the different dimensions. However, we note that the MDT does not really perform clustering. For example, suppose there were several qfb in a cell, whose range for Time was 0 to 3am, and suppose all the qfb were actually collected between 2 am and 3 am, and there was no qfb at other times. Then, the MDT, unlike a clustering algorithm, would not form a cell with range 2 am to 3 am for Time, since it would have no qfb to split this cell. MDT learning resembles the learning in HERMES [1]. HERMES performs off-line summarization of query response times, using information on the domain of attributes, query bindings and selectivity. HERMES does not model the unpredictable nature of WebSources, where user loads and network loads impact the response time. Finally, while the *NWS* facility collects data in an intrusive manner, i.e., each server is monitored internally, such statistics could be used when available to augment the MDT prediction.

5 Query Scrambling using the MDT Prediction

In this section, we first introduce the query scrambling (QS) technique for optimization. We then describe how the MDT prediction of delay (response time) can be used in the QS algorithm. Next, we present a number of experiments demonstrating how well the MDT prediction benefits QS.

5.1 Query Scrambling

Query Scrambling [2, 24] is a query optimization technique to combat the unexpected delay problem in wide area networks; such delay results in the unavailability of data residing at a remote site. Reasons for the delay include network congestion, overload at the server, a physical disconnection, etc. Modern query optimizers produce plans statically, and cannot account for unexpected delays. The delay may linearly increase the query response time (RT), compared to the statically determined RT.

QS tries to modify a plan dynamically, to perform other work which does not directly depend on the delayed relation, so as to hide the effect of delay. The QS algorithm first looks for query plan subtrees which are unaffected by the delay, and tries to reschedule the current plan to execute the unaffected subtree first; this is the *Rescheduling Phase*. In the case that there are no unaffected subtrees identified in the current plan, the QS algorithm may create new operators, for example joins of relations which were not joined in the original plan; this is the *Operator Synthesis Phase*. The QS algorithm proceeds in several iterations, or *scrambling steps*. Each step may produce a new, and commonly more expensive plan, but whose response time may be less, in the presence of delay. In [24], they only consider initial delays on one relation. However the approach can be generalized to more complex cases.

The general behavior of RT, when QS is utilized, is shown in Figure 10, which plots response time (RT) versus delay[6]. RT_i is the response time of the initial plan with no delay. With delay and no QS, RT_i will increase linearly with delay. Each vertical step in the graph labeled QS corresponds to a scrambling step and switches to a new plan. All lines parallel to the line labeled $RT = Delay$, e.g., the line labeled $RT = RT_i + Delay$ indicates that no scrambling occurs, and the QS algorithm preserves the current plan.

5.2 Impact of MDT Delay Prediction on Query Scrambling

At each scrambling step, the QS algorithm must base its decision on knowledge of the expected delay. The delay when a scrambling decision is made is a critical point with a *critical delay CrD*. The value of the MDT prediction of the expected delay, ED, the real delay RD, and the critical delay CrD, all play a significant

[6] We consider the ED optimization strategy for QS which we felt was best suited for our study[24].

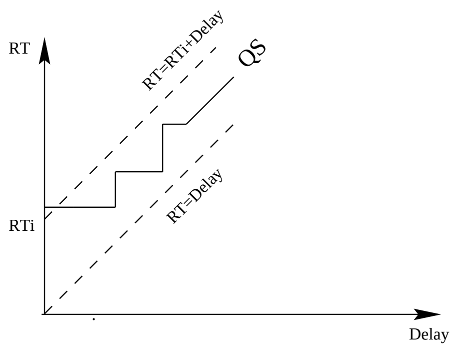

Figure 10: Behavior of Response Time (RT) with Query Scrambling (QS)

role in determining how the MDT prediction can affect the choice of the QS algorithm. To simplify our presentation, we consider a simple query with two alternative plans and one point of critical delay, as seen in Figure 11(a). We can generalize to more complex queries in a straightforward manner as is seen later. Consider an initial (optimal) query plan P_i, and a new plan P_s, which is suboptimal but can hide some delay.

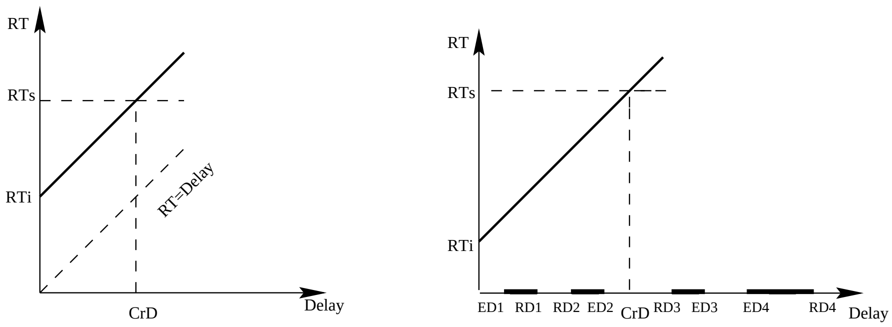

Figure 11: Critical Delay (a) and Safe Prediction Error for the MDT (b)

RT_i is the response time of P_i assuming no delay, and RT_s is the initial response time for P_s. In general, CrD is a function of RT_s and RT_i, and can be determined to be $RT_s - RT_i$, where we ignore the relative costs of the plans.

A perfect prediction scenario is the case when $ED = RD$. When prediction is imperfect, i.e., $ED \neq RD$, we may expect to make a poor scrambling decision. However, what is significant, is that even though the prediction is imperfect, the prediction error could be safe, with no penalty associated with an imperfect prediction. Conversely, the prediction error could be unsafe and could lead to underestimation or overestimation penalties. This distinction is crucial in evaluating the quality of MDT prediction in the context of Query Scrambling.

We now describe the relationship between ED, RD and CrD, and their influence on the choice of the QS algorithm. The first case is when the prediction error $\mid RD - ED \mid$ is safe, and there is no penalty. The situation is portrayed in Figure 11(b), and is the case when ED and RD occur on the same side of CrD.

- $ED < RD$. If $ED < CrD$ & $RD < CrD$, or $RD \geq CrD$ & $ED \geq CrD$, the scrambler is insensitive to the prediction error. In the first case, QS chooses the initial plan P_i, and in the second case, QS chooses P_s.

- $ED > RD$. If $ED > CrD$ & $RD > CrD$, or $RD \leq CrD$ & $ED \leq CrD$, QS is insensitive to prediction error. In the first case, QS chooses the new plan P_s, and in the second case QS chooses initial plan P_i.

Next, we discuss the case where the prediction error is `unsafe` and there is a penalty.

- $ED < RD$ (Figure 12a). $ED < CrD$ & $RD \geq CrD$. This is an underestimation error of the MDT prediction which causes an *underestimation penalty of prediction*. The value of the penalty E_{under} is equal to $(RT_i + RD) - RT_s = RD - (RT_s - RT_i) = RD - CrD$. For a delay RD, the better plan is P_s, but due to MDT prediction of ED, the QS algorithm chooses P_i and incurs a penalty.

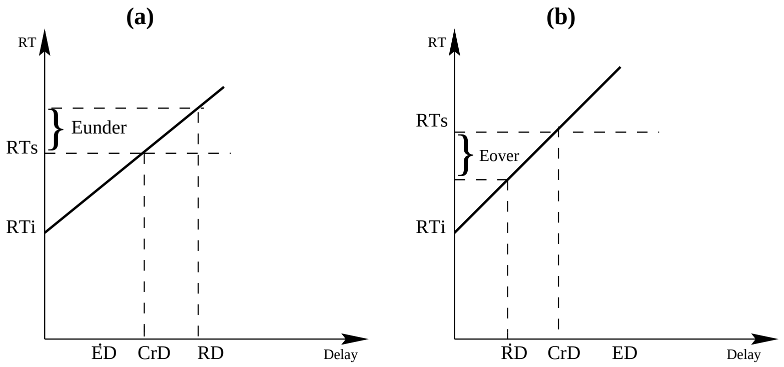

Figure 12: Underestimation and Overestimation Penalties due to Prediction Error

- $ED > RD$ (Figure 12b). $RD \leq CrD$ & $ED > CrD$. This is an overestimation error of the MDT prediction, which causes an *overestimation penalty of prediction*. The value of the penalty $E_{over} = RT_s - (RT_i + RD) = (RT_s - RT_i) - RD = CrD - RD$. For a delay of RD, the better plan is P_i, but due to MDT prediction of ED, the QS algorithm chooses P_s and incurs the penalty.

Note that the values of both penalties E_{under} and E_{over} depend on CrD and RD, and is independent of RT_i and RT_s. We will use this property later, in section 5.3, to estimate the quality of MDT prediction.

We can characterize the quality of MDT delay prediction in the context of query scrambling as follows: *a good MDT prediction has to minimize both underestimation and overestimation penalties*. In the next section, we consider how well the MDT performs on experiment data. What is critical is that while the MDT prediction may be imperfect, the prediction error could be `safe` and there could be no penalty.

5.3 Experimental Evaluation of the MDT Prediction

The experimental evaluation was performed using a simulator of a distributed query processing environment, with a two-phase randomized query optimizer [11]. The simulator is described in [24]. The QS algorithm is implemented on top of the simulator. The query processing environment has a *query site*, which execute queries, and *data sites*, that store relations used in queries. It assumes each relation is located in a different data site. All joins are executed using the *hybrid hash join* method [16]. The query site has 300 pages of memory and the page size is 4096 bytes. All simulation parameters are defined as in [24].

5.3.1 Result of MDT Prediction on Query Qx2 (Small Sample)

Consider a simple join query Qx2 on two relations SMALL (50000 tuples with tuple size of 180K) and LARGE (2000000 tuples of 180K each), over a single join attribute and a projectivity of 0.2 for each relation, i.e., 20% of the tuple is projected in the result.

We assume the SMALL relation is delayed. The initial plan with cost of 673.61 seconds, is a hash join, where the left (inner) relation is SMALL. The alternative and suboptimal plan with cost of 690.73 seconds, has the left (inner) relation as LARGE. The cost difference for the two plans, for the simulation environment described above is around 17 seconds, and this is the only scrambling opportunity, i.e., $CrD = 17sec$.

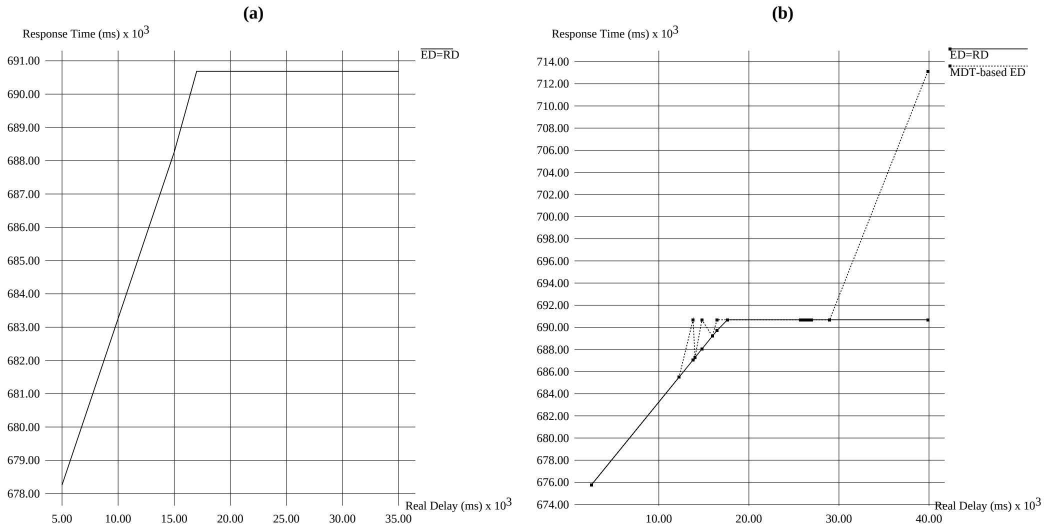

Figure 13: Critical Delay (a) and small MDT sample quality (b) for Qx2 query

Table 1 has 16 MDT predictions, using the data collection `BR`. We report on the result of the QS algorithms using the MDT prediction of real delay, i.e., if the prediction error was `safe`, or lead to a penalty of either *overestimation* or *underestimation*. We purposefully obtained this sample of MDT prediction in an early stage of learning, when the prediction error was significant, to give us an opportunity to study the penalties of MDT prediction error. Figure 13a represents the critical delay for this query. Figure 13b plots two response time curves. For the first curve, labeled $ED = RD$, the simulator uses the perfect prediction. For the second curve, labeled $MDT - based\ ED$, the simulator uses the MDT prediction of ED. Referring to Table 1, test #5 leads to an underestimation penalty, and tests #6, #7 and #12 lead to overestimation penalties. In all other tests, the prediction errors were `safe` and there were no penalties. In Figure 13b, the 4 predictions that incurred penalties are shown as deviations from the curve labeled $ED = RD$.

Table 1: Results of Testing the Prediction of an Initial MDT Sample

Test#	1	2	3	4	5	6	7	8
RD	2499.0	12236.0	14020.0	15942.0	39869.0	14781.0	16460.0	17621.0
ED	12565.0	12565.0	12565.0	14020.0	14020.0	39869.0	39869.0	39869.0
Error	safe	safe	safe	safe	**over**	**under**	**under**	safe

Test#	9	10	11	12	13	14	15	16
RD	28968.0	26926.0	26517.0	13804.0	25734.0	25852.0	26825.0	26181.0
ED	39869.0	29959.0	26926.0	26517.0	20649.0	22729.0	23705.0	24448.0
Error	safe	safe	safe	**under**	safe	safe	safe	safe

5.3.2 Large Scale Testing of the MDT Prediction Penalty

To facilitate the ease of large scale testing, with thousands of MDT predictions, we exploit the fact that once the critical delay is obtained for a plan, then the `safe` MDT prediction error, and the values for E_{under} and E_{over}, for an `unsafe` prediction error, can be directly calculated, based on the value of CrD and RD. This was discussed in section 5.2. Thus, we used the simulator to generate plans, performs scrambling, and evaluate the critical delay points. We could then determine the quality of MDT prediction directly, using the RD and MDT prediction. In the following tests, we used the MDT prediction for the data collection `OZ`.

5.3.3 Result of the MDT Prediction Penalty on Query Qx4

We consider a 4-way join query, Qx4, whose statistics, initial plan P_i, and first scrambled plan P_s are in Figure 14. We also assume a projectivity of 0.2 for each relation. The cost of the initial plan for Qx4 is 1527.03 seconds. The cost of the scrambled plan is 1559.07 seconds. Thus the first critical delay CrD occurs at approximately 32 seconds.

Table	#Tuples	Tuple Size
SMALL	10000	180
MEDIUM	500000	180
LARGE	2000000	180
HUGE	3000000	180

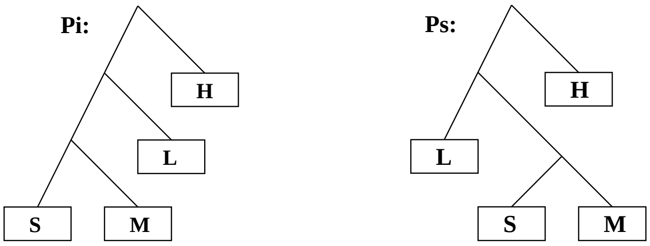

Figure 14: Statistics, P_i and P_s for *Qx4* query

Table 2 represents the results of testing the MDT prediction penalty. The first row represents the entire sample of 3100+ predictions. There were 657 cases that resulted in penalty, 111 were unsafe underestimation errors, and 546 were unsafe overestimation errors. We note that the total underestimation penalty is comparable to the overestimation penalty. When we consider the first 1000 predictions, we note that the total penalty incurred is a significant proportion of the total penalty for the total sample. Thus, when we consider either the last 500 MDT predictions, or the last 250, we see that the number of errors is small, and the total penalty is a small fraction of the total penalty for the total sample. This indicates that as the MDT learns, the total penalty due to unsafe estimation error decreases significantly.

Table 2: Results of MDT Prediction for Query Qx4

# of unsafe errors	**Total penalty (ms)**	**#underest.**	**Underest. penalty (ms)**	**#overest.**	**Overerest. penalty (ms)**	**Sample size**
657	5935589.0	111	2733440.0	546	3202149.0	3100+
90	2200697.0	47	1570113.0	43	630584.0	first 1000
25	59078.0	0	0.0	25	59078.0	last 500
2	875.0	0	0.0	2	875.0	last 250

5.3.4 Result of the MDT Prediction on Query Q8.mod

Finally, we considered a modified version of the TPC-D benchmark query Q8, the National Market Share Query. The SQL query statement, the statistics for the relations, the initial plan P_i, and the first scrambled plan P_s, is in in Appendix B. As before, we consider that there is one delayed relation PART. The critical delay at the first scrambling step that is considered by the simulator occurs at approximately 21 seconds.

Table 6 presents the results of the MDT predictions. As in the previous example, the penalty due to overestimation was dominant. While 222 of the 511 `unsafe` predictions occurred in the first 1000 predictions, the total penalty for the first 1000 predictions is significant, compared to the total penalty for the total sample. Further, there were no `unsafe` predictions in the last 500 MDT predictions.

We conclude that the MDT learning can be characterized as very good, from the perspective of minimizing the total penalty incurred by the QS algorithm, when QS uses the MDT prediction for the real delay.

Table 3: Result of MDT Prediction for Query Q8.mod

# of unsafe errors	**Total penalty (ms)**	**#underest.**	**Underest. penalty (ms)**	**#overest.**	**Overerest. penalty (ms)**	**Sample size**
511	3631717.0	88	1717749.0	423	1913968.0	3100+
222	2705800.0	71	1633942.0	151	1071858.0	first 1000
0	0.0	0	0.0	0	0.0	last 500

6 Conclusion

In this paper, we report on the MDT, a tool which uses query feedback from WebSources to predict response time (delay) and confidence in the prediction. We report on the features of MDT training, which improves with the correct ordering of significant dimensions such as Quantity, Day and Time, and also improves with inclusion of significant dimensions. We test the MDT on experiment data collected from several WebSources. We then use the MDT prediction of delay in a scrambling enabled optimizer. We identify when MDT overestimation or underestimation of the real delay is unsafe and results in a penalty, and when the prediction error is safe, and there is no penalty. We test how good the MDT prediction is in minimizing these penalties, for the experiment data.

In future work, we will refine the idea of initial delay versus response time, and we will investigate tuning the scrambling algorithm using the MDT confidence in its prediction. We will augment the MDT qfb with performance data collected from other monitoring techniques, and we plan to study MDT performance in an experimental environment. We also plan to compare the MDT learning with neural network learning.

Acknowledgements

We thank María Esther Vidal, Michael Franklin and Tolga Urhan for their many comments and insights into this study; we thank Tolga for the use of his scrambling enabled optimizer, and we thank Tao Zhan and Pyuonguk Cho for their assistance in data analysis.

References

[1] S. Adali et al. Query caching and optimization in distributed mediator systems. *Proc. of the ACM Sigmod Conference*, 1996.

[2] L. Amsaleg, M. Franklin, A. Tomasic, and T. Urhan. Scrambling query plans to cope with unexpected delays. *Proc. of PDIS Conference*, 1996.

[3] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth and Brooks/Cole, 1984.

[4] Microsoft Corporation. *OLE2 Programmer's Reference.* Microsoft Press, Redmond WA, 1996.

[5] W. Du et al. Query optimization in a heterogeneous dbms. *Proc. of the Very Large Data Bases Conference (VLDB)*, 1992.

[6] D. Florescu et al. A methodology for query reformulation in cis using semantic knowledge. *Intl. Journal of Intelligent and Cooperative Information Systems, special issue on Formal Methods in Cooperative Information Systems*, 1996.

[7] G. Gardarin et al. *IRO-DB: A Distributed System Federating Object and Relational Databases, In Object-Oriented Multidatabase Systems : A solution for Advanced Applications, Bukhres, O. and Elmagarmid,A.* Prentice Hall, 1996.

[8] Open System Group. An explanation of the specweb96 benchmark. *http://www.specbench.org/osg/web96/webpaper.html*, 1996.

[9] J.R. Gruser, L. Raschid, and V. Zadorozhny. Learning from query feedback to predict response time of web sources. *Technical Report, UMIACS, University of Maryland*, 1998 (in preparation).

[10] J. A. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, CA, 1991.

[11] Y. Ioanidis and Y. Kang. Randomized algorithms for optimizing large join queries. *Proc. of the ACM Sigmod Conference*, 1990.

[12] A. Layman et al. The xml- data home page. *http://www.microsoft.com/standards/xml/xmldata-f.htm.*

[13] A.Y. Levy et al. Querying heterogeneous information sources using source descriptions. *Proc. of VLDB*, 1996.

[14] Sun Microsystems. Java (tm): Programming for the internet. *http://java.sun.com.*

[15] Sun Microsystems. Java (tm): Programming for the internet. *http://java.sun.com.*

[16] P. Mishra and M. Eich. Join processing in relational databases. ACM Computing Surveys*, Vol. 24, N. 1*, 1992.

[17] H. Naacke, G. Gardarin, and A. Tomasic. Leveraging mediator cost models with heterogeneous data sources. *Proc. of ICDE*, 1998.

[18] Y. Papakonstantinou et al. Capabilities-based query rewriting in mediator systems. *Proc. of the Conference on Parallel and Distributed Information Systems*, 1996.

[19] M.T. Roth and P. Schwarz. Don,t scrap it, wrap it! a wrapper architecture for legacy data sources. *Proc. of VLDB*, 1997.

[20] A. Sayal, P. Scheuermann, and P. Vingralek. Selection algorithms for replicated web servers. *Proc. of the Internet Server Performance Workshop (in conjunction with SIGMETRICS'98)*, 1998.

[21] K. Thompson, G. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network, November/December*, 1997.

[22] A. Tomasic et al. Scaling heterogeneous databases and the design of disco. *Proceedings of the Intl. Conf. on Distributed Computing Systems*, 1996.

[23] G. Trent and M. Sake. Webstone: The first generation in http server benchmarking. *http://www.mindcraft.com/webstone/paper.html*, 1995.

[24] T. Urhan, M. Franklin, and L. Amsaleg. Cost-based query scrambling for initial delays. Proc. of the ACM Sigmod Conference, 1998.

[25] V. Vassalos and Y. Papakonstantinou. Using knowledge of redundancy for query optimization in mediators. *Proc. of the AAAI Symposium on AI and Data Integration*, 1998.

[26] A. Ward, P. Glynn, and K. Richardson. Internet service performance failure detection. *Proc. of the Internet Server Performance Workshop (in conjunction with SIGMETRICS'98)*, 1998.

[27] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.

[28] R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. *Proc. of the 6th High-Performance Distributed Computing Conference*, 1997.

A Contingency Tables for χ^2 Analysis for Observation 02 (α of 0.01)

Source	URL	Sample Size
BR	http://www.lbd.dcc.ufmg.br	3163
INR	http://www-rodin.inria.fr	5154
LeM	http://www.lemonde.fr	3051
OZ	http://broncho.ct.monash.edu.au	5314
UM	http://www.umiacs.umd.edu	9661
UT	http://www.cs.toronto.edu	9397
WTH	http://www.weather.com	2411

Dim - Dimension
EV - Expected Value
OV - Observed Value

BR

Dim	EV	Range	OV
Date	26.217	30-30-40	16.195
		25-25-50	20.457
		50-25-25	12.768
		25-50-25	22.014
Time	29.141	30-30-40	128.38
		25-25-50	108.135
		50-25-25	122.446
		25-50-25	128.893
Quantity	29.141	30-30-40	1519.686
		25-25-50	1638.261
		50-25-25	1271.302
		25-50-25	1761.313

INR

Dim	EV	Range	OV
Date	26.217	30-30-40	85.983
		25-25-50	65.387
		50-25-25	108.053
		25-50-25	107.11
Time	29.141	30-30-40	89.872
		25-25-50	58.421
		50-25-25	123.59
		25-50-25	127.392
Quantity	29.141	30-30-40	3609.475
		25-25-50	4192.21
		50-25-25	2634.512
		25-50-25	3265.805

LeM

Dim	EV	Range	OV
Date	26.217	30-30-40	100.322
		25-25-50	83.02
		50-25-25	101.231
		25-50-25	113.875
Time	29.141	30-30-40	57.873
		25-25-50	41.403
		50-25-25	79.784
		25-50-25	74.307
Quantity	29.141	30-30-40	
		25-25-50	
		50-25-25	
		25-50-25	

OZ

Dim	EV	Range	OV
Date	26.217	30-30-40	36.224
		25-25-50	37.074
		50-25-25	53.797
		25-50-25	56.893
Time	29.141	30-30-40	10.151
		25-25-50	8.516
		50-25-25	13.985
		25-50-25	13.34
Quantity	29.141	30-30-40	4139.476
		25-25-50	4220.134
		50-25-25	3128.405
		25-50-25	3860.471

UM

Dim	EV	Range	OV
Date	26.217	30-30-40	35.6
		25-25-50	35.335
		50-25-25	37.063
		25-50-25	32.112
Time	29.141	30-30-40	14.312
		25-25-50	17.274
		50-25-25	17.67
		25-50-25	12.705
Quantity	29.141	30-30-40	4404.131
		25-25-50	5321.17
		50-25-25	2112.307
		25-50-25	4826.418

UT

Dim	EV	Range	OV
Date	26.217	30-30-40	18.529
		25-25-50	19.474
		50-25-25	25.896
		25-50-25	29.346
Time	29.141	30-30-40	38.927
		25-25-50	36.976
		50-25-25	38.108
		25-50-25	32.085
Quantity	29.141	30-30-40	5334.823
		25-25-50	6465.84
		50-25-25	3326.436
		25-50-25	5011.74

WTH

Dim	EV	Range	OV
Date	26.217	30-30-40	12.14
		25-25-50	16.526
		50-25-25	14.016
		25-50-25	15.81
Time	29.141	30-30-40	168.109
		25-25-50	175.286
		50-25-25	141.931
		25-50-25	184.193
Quantity	29.141	30-30-40	328.125
		25-25-50	294.25
		50-25-25	225.294
		25-50-25	298.533

B Query Q8.mod

```
SELECT O_ORDERDATE, L_EXTENDEDPRICE, N2.NAME
FROM PART, CUSTOMER, ORDER, LINEITEM, SUPPLIER, NATION N1, NATION N2, REGION
WHERE P.PARTKEY = L.PARTKEY
        AND L.SUPPKEY = S.SUPPKEY AND O.ORDERKEY = L.ORDERKEY
        AND C.CUSTKEY = O.CUSTKEY AND C.NATIONKEY = N1.NATIONKEY
        AND N1.REGIONKEY = R.REGIONKEY AND R.NAME = 'EUROPE'
        AND S.NATIONKEY = N2.NATIONKEY AND O.ORDERDATE BETWEEN '97-01-01' AND '98-12-31'
        AND P_TYPE = 'SMALL PLATED STEEL'
```

Table	#Tuples	Tuple Size
CUSTOMER	150000	180
ORDER	1500000	100
LINEITEM	1000000	120
SUPPLIER	10000	160
NATION1	25	40
NATION2	25	40
REGION	5	40
PART	200000	160

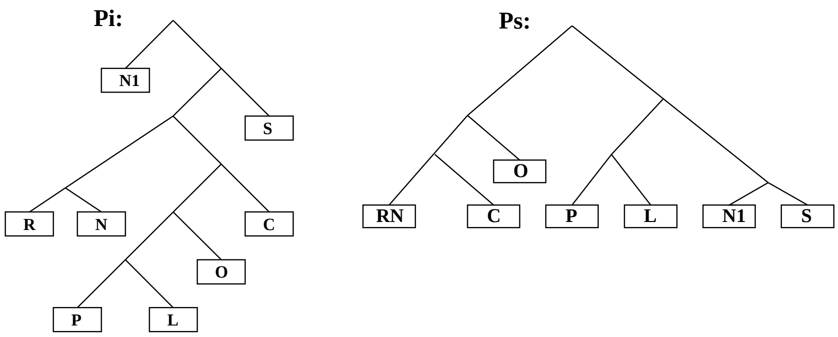

Figure 15: Statistics, P_i and P_s for Query *Q8.mod*