The VLDB Journal (2004) 13: 162–176 / Digital Object Identifier (DOI) 10.1007/s00778-003-0110-4

# Evolutionary techniques for updating query cost models in a dynamic multidatabase environment

**Amira Rahal**[1], **Qiang Zhu**[1], **Per-Åke Larson**[2]

[1] Department of Computer and Information Science, The University of Michigan – Dearborn, Dearborn, MI 48128, USA
(e-mail: {arabi,qzhu}@umich.edu)
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
(e-mail: palarson@microsoft.com)

Edited by L. Liu. Received: November 25, 2002 / Accepted: May 20, 2003
Published online: September 30, 2003 – 

**Abstract.** Deriving local cost models for query optimization in a dynamic multidatabase system (MDBS) is a challenging issue. In this paper, we study how to evolve a query cost model to capture a slowly-changing dynamic MDBS environment so that the cost model is kept up-to-date all the time. Two novel evolutionary techniques, i.e., the shifting method and the block-moving method, are proposed. The former updates a cost model by taking up-to-date information from a new sample query into consideration at each step, while the latter considers a block (batch) of new sample queries at each step. The relevant issues, including derivation of recurrence updating formulas, development of efficient algorithms, analysis and comparison of complexities, and design of an integrated scheme to apply the two methods adaptively, are studied. Our theoretical and experimental results demonstrate that the proposed techniques are quite promising in maintaining accurate cost models efficiently for a slowly changing dynamic MDBS environment. Besides the application to MDBSs, the proposed techniques can also be applied to the automatic maintenance of cost models in self-managing database systems.

**Keywords:** Multidatabase – Query optimization – Cost model – Evolutionary technique – Self-managing database

## 1 Introduction

A multidatabase system (MDBS) integrates data from multiple component (local) databases. A major challenge, among others [6,8,10,11,16], for performing global query optimization in an MDBS is that some local information required by global query optimization, such as local cost models, may not be available at the global level. However, the global query optimizer needs such local cost information to decide how to decompose a global query into local queries and where to execute the local queries.

Research supported by the US National Science Foundation under Grant # IIS-9811980 and The University of Michigan under OVPR and UMD grants.

Several techniques to derive cost models for an autonomous local database system (DBS) at the global level in an MDBS have been proposed in the literature recently. Du et al. proposed a calibration method that makes use of the observed costs of some special queries run against a special synthetic calibrating database to deduce necessary local cost parameters [5]. Gardarin et al. extended Du et al.'s method so as to calibrate cost models for object-oriented local DBSs in an MDBS [7]. Zhu and Larson proposed a query sampling method that develops regression cost models for local query classes based on observed costs of sample queries run against actual user databases [21,24,25]. Zhu and Larson also introduced a fuzzy method to derive fuzzy cost models in an MDBS based on fuzzy set theory [23]. Naacke et al. suggested an approach to combining a generic cost model with specific cost information exported by wrappers for local DBSs [12]. Adali et al. suggested maintaining a cost vector database to record cost information for every query issued to a local DBS [1]. Roth et al. introduced a framework for costing in their *Garlic* federated system [15].

All the above techniques considered only a static system environment. However, in reality, an MDBS environment may change dramatically over time. There are many dynamic factors in an MDBS environment. They can be classified into three types based on their changing frequencies: (I) *frequently changing factors*, such as CPU load, number of I/Os per second, and size of memory space being used, etc., which can change significantly within a short period of time (e.g., every few seconds or minutes); (II) *slowly changing factors*, such as local database management system (DBMS) configuration parameters, physical data distribution/organization on a disk, local database conceptual/physical schemas, etc., which usually change little by little, and a significant change may be accumulated after a certain period of time (e.g., a couple of days, weeks, or months); and (III) *steady factors*, such as local DBMS type, local database location, local CPU speed, etc., which may stay unchanged for a long time (e.g., many months or even years). Note that, since we concern ourselves with local cost models in an MDBS, only dynamic factors at local sites are considered here. In general, there are also dynamic

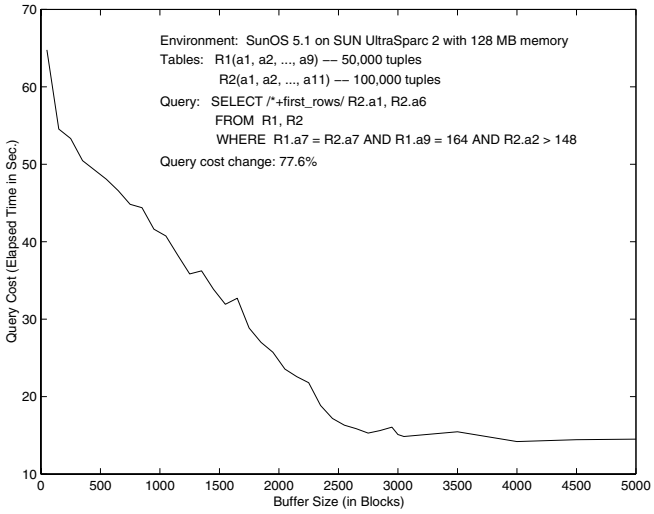


**Fig. 1.** Query cost affected by buffer size in Oracle 8.0

environmental factors for the network in an MDBS that were considered in [18].

Clearly, the steady factors (Type III) usually do not cause any problem for a query cost model since they rarely change. To take the frequently changing factors (Type I) into consideration for estimating query costs, Zhu et al. suggested three techniques in [19,20], i.e., the qualitative approach, the fractional analysis approach, and the probabilistic approach. The qualitative approach is suitable for estimating the cost of a small query using a cost model with a qualitative variable indicating system contention states. The fractional analysis approach is suitable for estimating the cost of a large query by analyzing cost fractions in a dynamic environment following a prior known load curve. The probabilistic approach is suitable for estimating the cost of a large query based on Markov chain theory in a randomly changing dynamic environment.

However, no research has been done on estimating local cost parameters in a slowly changing dynamic environment (caused by factors of Type II). Although such an environment may not change dramatically during the execution of one query, the costs of the same query executed at different times in the environment can be significantly different. Figure 1 shows that the cost of a query in Oracle 8.0 can change dramatically as the buffer size (a configuration parameter) is adjusted by a database administrator over time. Note that the performance change could be even more dramatic if the query were performed against a larger database on a faster machine with larger memory. Compared to the factors of Type I, the factors of Type II change gradually rather than rapidly. But a significant change may be observed after a certain period of time. An obsolete cost model may cause a query optimizer to choose an inefficient execution plan for a query, which would lead to a serious performance problem. The question now is how to obtain accurate query cost estimates at all times in such a slowly changing environment.

In this paper, we tackle this challenge by evolving a cost model to capture the slowly changing environment so that the cost model is kept up-to-date all the time. One direct method of keeping a cost model updated is to periodically rebuild the cost model by the query sampling method [21]. However, the overhead of such a rebuilding approach is high. To reduce the overhead, we propose two new evolutionary techniques, i.e., a shifting method and a block-moving method. The key idea is to develop recurrence updating formulas to adjust a cost model at each updating step rather than rebuild the cost model from scratch every time so that some common work done previously can be reused. The shifting method evolves a cost model more smoothly but takes more overhead as compared with the block-moving method. Evolving a cost model to capture a dynamic database environment is our novel approach; it has not been found in literature.

The rest of the paper is organized as follows. Section 2 discusses the idea of cost model evolution and the direct rebuilding approach. Section 3 introduces the shifting method. Section 4 presents the block-moving method. Section 5 considers some implementation issues. Section 6 shows some experimental results. Section 7 summarizes the conclusions.

## 2 The rebuilding approach

In the query sampling method [21,25], queries that can be performed on a local DBS are first grouped into homogeneous query classes, and a cost model is then developed for each query class based on the observed costs of sample queries drawn from the class via multiple regression analysis in statistics. Such a cost model captures the performance behavior of queries from the relevant class for a specific environment in which the sample queries were executed. If the environment has changed dramatically since the cost model was developed, the cost model may become out of date. The question is how to keep the cost model up-to-date so that it always reflects the current environment.

Suppose we want to keep the cost model $M$ for a query class $G$ up-to-date in a dynamic multidatabase environment. Let the set of significant explanatory variables determined by the query sampling method for the cost model be $\{x_1, x_2, \ldots, x_n\}$, e.g., the operand table size, the result table size,[1] the operand table tuple length, etc. Cost model $M$ is then of the following form [24]:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \quad (1)$$

where $\beta_i$'s ($0 \leq i \leq n$) are the regression coefficients determined by sample queries. Using the relevant value of each explanatory variable $x_j$ ($1 \leq j \leq n$) for query $Q$ in $G$, we can estimate cost $y$ of $Q$ from Eq. 1. Note that the aim of this paper is not to develop a new cost modeling technique. Instead, it assumes that a cost model and its parameters are determined by the techniques presented in [24]. This paper focuses on developing new techniques to efficiently adjust the coefficients in the cost model so that the environmental changes can be effectively captured.

In the rest of this paper, we adopt the following notation. A column vector is denoted by $\vec{z}$ using a lowercase letter. A row vector is denoted by the transposition of its corresponding column vector $\vec{z}^T$. A matrix is denoted by a bold-faced capital letter (e.g., $\mathbf{A}$), and the inverse of $\mathbf{A}$ is denoted by $\mathbf{A}^{-1}$. A

[1] The result table size is usually estimated by using an estimated selectivity of the query.

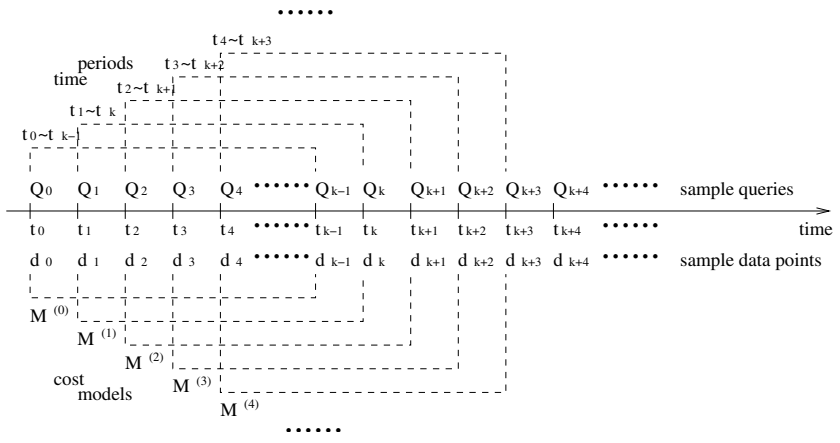


**Fig. 2.** Rebuilding up-to-date cost models

scalar value or variable is denoted by a normal lowercase letter (e.g., $b$).

Let $Q_i$ ($i = 0, 1, 2, \ldots$) be a sample query from $G$ executed at time $t_i$. Let $y_i$ be the observed cost (elapsed time) of $Q_i$ and $x_{ij}$ be the observed value of $j$-th variable $x_j$ ($1 \leq j \leq n$) for $Q_i$. Let row vector $\vec{x}_i^T = (1, x_{i1}, x_{i2}, \ldots, x_{in})$. Note that the first component "1" in vector $\vec{x}_i^T$, which corresponds to the constant term in Eq. 1, is used to simplify the cost formula derivation later on. We call pair $(y_i, \vec{x}_i^T)$ as the sample data point (for sample query $Q_i$) observed at time $t_i$. Hence we have a sequence of sample data points $(y_0, \vec{x}_0^T)$, $(y_1, \vec{x}_1^T), \ldots$ (corresponding to the sequence of sample queries $Q_0, Q_1, \ldots$) at $t_0, t_1, \ldots$.

Assume the sample size for multiple regression analysis in the query sampling method is $k$.[2] At time $t_{k-1}$, we have $k$ sample data points $(y_0, \vec{x}_0^T)$, $(y_1, \vec{x}_1^T)$, ..., $(y_{k-1}, \vec{x}_{k-1}^T)$. Applying the multiple regression analysis on these sample data points, we can obtain a cost model $M^{(0)}$ that reflects the system performance behavior for query class $G$ during the time period $t_0 \sim t_{k-1}$.

As mentioned before, the system environment may change significantly over time, although for our problem we assume that it changes slowly. If we keep using $M^{(0)}$ to estimate the costs of queries executed at time $t_k$, $t_{k+1}$, $t_{k+2}$, ..., the estimates may become progressively worse. To get better estimates, we need to update the cost model according to sample data points observed at $t_k$, $t_{k+1}$, $t_{k+2}$, .... More specifically, at time $t_k$, when sample data point $(y_k, \vec{x}_k^T)$ is obtained, we should derive a new cost model $M^{(1)}$ based on the most recent $k$ sample data points $(y_1, \vec{x}_1^T)$, $(y_2, \vec{x}_2^T)$, ..., $(y_k, \vec{x}_k^T)$. In other words, we should incorporate the performance information contained in the newest sample data point $(y_k, \vec{x}_k^T)$ into the cost model since it reflects the current system environment. On the other hand, we also need to remove the oldest sample data point $(y_0, \vec{x}_0^T)$ from consideration for the cost model since (i) it contains the least information about the performance behavior of the current system environment and (ii) keeping all old sample data points would make the sample set size grow bigger and bigger, which increases the complexity of cost model derivation.

In general, at time $t_{s+k-1}$ ($s = 1, 2, \ldots$), when sample data point $(y_{s+k-1}, \vec{x}_{s+k-1}^T)$ is obtained, we need to derive a new cost model $M^{(s)}$ based on $k$ sample data points $(y_s, \vec{x}_s^T)$, $(y_{s+1}, \vec{x}_{s+1}^T), \ldots, (y_{s+k-1}, \vec{x}_{s+k-1}^T)$ (Fig. 2).

Note that although the difference between two consecutive cost models $M^{(i)}$ and $M^{(i+1)}$ ($i$=0, 1, ...) may be small, assuming the environment changes slowly, the difference between two far-away cost models $M^{(i)}$ and $M^{(i+q)}$, where $q$ is large, can be very significant. Using the up-to-date cost models $M^{(0)}$, $M^{(1)}$, $M^{(2)}$, ... to estimate query costs in the current system environment, we can get better cost estimates as compared with using the static cost model $M^{(0)}$ all the time.

The question is how to derive the up-to-date cost models $M^{(1)}$, $M^{(2)}$, $M^{(3)}$, .... One simple way is to rebuild cost model $M^{(s)}$ from scratch at each time $t_{s+k-1}$ ($s = 1, 2, \ldots$) via multiple regression analysis. In other words, we solve the following normal equations:

$$(\sum_{i=s}^{s+k-1} \vec{x}_i \vec{x}_i^T)\vec{\beta}^{(s)} - \sum_{i=s}^{s+k-1} \vec{x}_i y_i = 0 \tag{2}$$

for the vector $[\vec{\beta}^{(s)}]^T = (\beta_0^{(s)}, \beta_1^{(s)}, \ldots, \beta_n^{(s)})$ of $\beta_i^{(s)}$'s coefficients in cost model $M^{(s)}$ of the form of Eq. 1, which minimizes the following sum of squared error terms: $f = \sum_{i=s}^{s+k-1}(\vec{x}_i^T \vec{\beta}^{(s)} - y_i)^2$. In fact, the solution of Eq. 2 can be expressed as:

$$\vec{\beta}^{(s)} = [\mathbf{P}^{(s)}]^{-1} \sum_{i=s}^{s+k-1} \vec{x}_i y_i \tag{3}$$

where $\mathbf{P}^{(s)} = \sum_{i=s}^{s+k-1} \vec{x}_i \vec{x}_i^T$ is termed the covariance matrix of normal equations (Eq. 2).

From [17], solving Eq. 2 requires

$$(k+1)(n+1)^2 + (k+2)(n+1) + (n+1)^3 \tag{4}$$

number of scalar multiplications and divisions. The number of scalar additions and/or subtractions involved is not considered here since they require less overhead. Hence the complexity of applying multiple regression $p$ times is $p$ times of Eq. 4. When $p$ is large, the overhead to keep the cost model updated is very significant.

[2] A commonly used rule for sampling is to sample at least ten observations for every parameter to be estimated [13], i.e., at least $10 * (n + 2)$ sample queries in our case, where $n$ is the number of explanatory variables in the cost model (note that the variance of error terms is also one parameter to be estimated here).

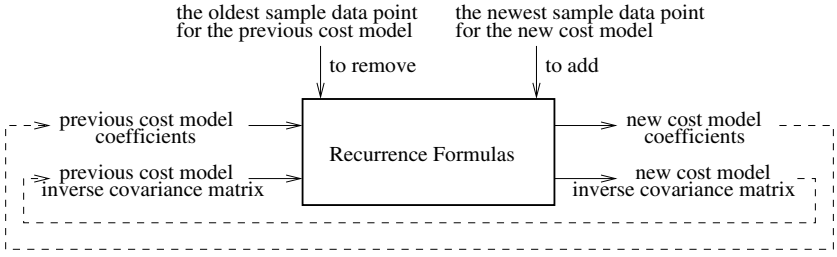


**Fig. 3.** The shifting method to evolve a cost model

The question now is if we can update the cost model more efficiently. We will present two efficient techniques for solving the problem in the next two sections.

## 3 The shifting method

Assume that we have obtained the initial cost model $M^{(0)}$ (at time $t_{k-1}$) via multiple regression based on initial $k$ sample data points $(y_0, \vec{x}_0^T)$, $(y_1, \vec{x}_1^T)$, ..., $(y_{k-1}, \vec{x}_{k-1}^T)$. At time $t_k$, when new sample data point $(y_k, \vec{x}_k^T)$ is observed, we need to get cost model $M^{(1)}$ based on $k$ sample data points $(y_1, \vec{x}_1^T)$, $(y_2, \vec{x}_2^T)$, ..., $(y_k, \vec{x}_k^T)$. Can we avoid rebuilding $M^{(1)}$ from scratch? Yes we can.

We notice that there are $k-1$ common sample data points, i.e., $(y_1, \vec{x}_1^T)$, ..., $(y_{k-1}, \vec{x}_{k-1}^T)$, on which both cost models $M^{(1)}$ and $M^{(0)}$ are based. This fact implies that $M^{(1)}$ and $M^{(0)}$ have a certain coherent relationship. Based on this observation, in this section we develop a shifting method to obtain new cost model $M^{(1)}$ by adjusting old cost model $M^{(0)}$. The basic idea is to add the effect of the newest sample data point $(y_k, \vec{x}_k^T)$ to $M^{(0)}$ and in the meantime remove the effect of the oldest sample data point $(y_0, \vec{x}_0^T)$ from $M^{(0)}$, resulting in new cost model $M^{(1)}$. That is, $M^{(1)}$ is obtained by shifting $M^{(0)}$ one sample data point toward the new time.

Note that during the initial regression analysis for $M^{(0)}$ we can get both the coefficients vector $\vec{\beta}^{(0)}$ and the inverse covariance matrix $[\mathbf{P}^{(0)}]^{-1}$ (see Eq. 3). In the following discussion, we assume that a cost model includes both the coefficients vector and the inverse covariance matrix. Using $\vec{\beta}^{(0)}$ and $[\mathbf{P}^{(0)}]^{-1}$ for initial cost model $M^{(0)}$ (together with $(y_0, \vec{x}_0^T)$ and $(y_k, \vec{x}_k^T)$), we can derive recurrence formulas to calculate $\vec{\beta}^{(1)}$ and $[\mathbf{P}^{(1)}]^{-1}$ for new cost model $M^{(1)}$. $\vec{\beta}^{(1)}$ and $[\mathbf{P}^{(1)}]^{-1}$ can then be used to calculate $\vec{\beta}^{(2)}$ and $[\mathbf{P}^{(2)}]^{-1}$ for even newer cost model $M^{(2)}$. In general, a new cost model $M^{(s)}$ ($s = 1, 2, \ldots$) can be obtained by adjusting the previous cost model $M^{(s-1)}$ (Fig. 3). Hence the cost model can evolve smoothly over time with the dynamic environment in this way.

A recurrence formula to update the cost model coefficients is given in the following theorem.

**Theorem 1** *The coefficients $\vec{\beta}^{(s)}$ of new cost model $M^{(s)}$ can be recursively calculated by adjusting the coefficients $\vec{\beta}^{(s-1)}$ of previous cost model $M^{(s-1)}$ ($s = 1, 2, \ldots$) as follows:*

$$\vec{\beta}^{(s)} = \vec{\beta}^{(s-1)} + \vec{w}(e/(1+a)) + \vec{w}'\{e' - \vec{x}_{s-1}^T\vec{w}(e/(1+a))\}/(1+a') \quad (5)$$

*where*

$$e = y_{s+k-1} - \vec{x}_{s+k-1}^T\vec{\beta}^{(s-1)},\ \vec{w} = [\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1}$$
$$a = \vec{x}_{s+k-1}^T\vec{w},\ e' = y_{s-1} - \vec{x}_{s-1}^T\vec{\beta}^{(s-1)}$$
$$\vec{w}' = -[\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s-1} + (\vec{w}\vec{x}_{s+k-1}^T[\mathbf{P}^{(s-1)}]^{-1})\vec{x}_{s-1}/(1+a)$$
$$a' = \vec{x}_{s-1}^T\vec{w}'$$

*Proof.* See Appendix. □

Theorem 1 states that the coefficients $\vec{\beta}^{(s)}$ of new model $M^{(s)}$ can be obtained by adjusting the coefficients $\vec{\beta}^{(s-1)}$ of previous model $M^{(s-1)}$. The adjustments for the previous coefficients depend on the newest sample data point $((y_{s+k-1}, \vec{x}_{s+k-1}^T))$, the oldest sample data point $((y_{s-1}, \vec{x}_{s-1}^T))$, the previous inverse covariance matrix $([\mathbf{P}^{(s-1)}]^{-1})$, and the errors ($e'$ and $e$) of using the previous model to estimate query costs at the newest and oldest sample data points, which in turn depend on the previous coefficients $(\vec{\beta}^{(s-1)})$. Hence, the new coefficients can be obtained by using $(y_{s+k-1}, \vec{x}_{s+k-1}^T)$, $(y_{s-1}, \vec{x}_{s-1}^T)$, $[\mathbf{P}^{(s-1)}]^{-1}$, and $\vec{\beta}^{(s-1)}$. Note that $\vec{\beta}^{(s)}$ obtained from Eq. 5 is the same as the one obtained from the rebuilding approach in terms of accuracy.

To use Eq. 5 to update the cost model iteratively for each new sample data point, we need not only the previous coefficients but also the previous inverse covariance matrix. Thus we also need to keep the inverse covariance matrix updated at each step. The following theorem gives the recurrence formula for updating the inverse covariance matrix.

**Theorem 2** *The inverse covariance matrix $[\mathbf{P}^{(s)}]^{-1}$ of new cost model $M^{(s)}$ can be recursively calculated based on the inverse covariance matrix $[\mathbf{P}^{(s-1)}]^{-1}$ of previous cost model $M^{(s-1)}$ ($s = 1, 2, \ldots$) as follows:*

$$[\mathbf{P}^{(s)}]^{-1} = [\mathbf{P}^{(s-1)}]^{-1} - (\vec{w}\vec{x}_{s+k-1}^T[\mathbf{P}^{(s-1)}]^{-1})/(1+a) - \vec{w}'\vec{x}_{s-1}^T\{[\mathbf{P}^{(s-1)}]^{-1} - (\vec{w}\vec{x}_{s+k-1}^T[\mathbf{P}^{(s-1)}]^{-1})/(1+a)\}/(1+a') \quad (6)$$

*where*

$$\vec{w} = [\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1},\ a = \vec{x}_{s+k-1}^T\vec{w}$$
$$\vec{w}' = -[\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s-1} + (\vec{w}\vec{x}_{s+k-1}^T[\mathbf{P}^{(s-1)}]^{-1})\vec{x}_{s-1}/(1+a)$$
$$a' = \vec{x}_{s-1}^T\vec{w}'$$

*Proof.* See Appendix. □

Theorem 2 indicates that the new inverse covariance matrix $[\mathbf{P}^{(s)}]^{-1}$ can be obtained by using $(y_{s+k-1}, \vec{x}_{s+k-1}^T)$, $(y_{s-1}, \vec{x}_{s-1}^T)$, and $[\mathbf{P}^{(s-1)}]^{-1}$.

Equations 5 and 6 share many common factors. They can be evaluated efficiently by one algorithm. To further improve efficiency, the following formula

$$\vec{\beta}^{(s)} = \vec{\beta}^{(s-1)} + \vec{w}(e/(1+a)) + \vec{w}'[y_{s-1} - \vec{x}_{s-1}^T(\vec{\beta}^{(s-1)} + \vec{w}(e/(1+a)))]/(1+a') \quad (7)$$

which is equivalent to Eq. 5, is adopted. The following algorithm updates the coefficients and the inverse covariance matrix of a cost model using the shifting method:

**Algorithm 1** Cost model evolution based on the shifting method.
**Input**: (1) Coefficients $\vec{\beta}^{(s-1)}$ of previous model $M^{(s-1)}$; (2) inverse covariance matrix $[\mathbf{P}^{(s-1)}]^{-1}$ of previous model $M^{(s-1)}$; (3) newest sample data point $(y_{s+k-1}, \vec{x}_{s+k-1}^T)$; (4) oldest sample data point $(y_{s-1}, \vec{x}_{s-1}^T)$.
**Output**: (1) Coefficients $\vec{\beta}^{(s)}$ of new model $M^{(s)}$; (2) inverse covariance matrix $[\mathbf{P}^{(s)}]^{-1}$ of new model $M^{(s)}$.
**Method**:
1. Compute $\vec{w} := [\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1}$
2. Compute $a := \vec{x}_{s+k-1}^T\vec{w}$;
3. Compute $\vec{u}^T := \vec{x}_{s+k-1}^T[\mathbf{P}^{(s-1)}]^{-1}$
4. Compute $\mathbf{B} := \vec{w}\vec{u}^T$
5. Compute $\mathbf{A} := \mathbf{B}/(1+a)$
6. Compute $\mathbf{C} := [\mathbf{P}^{(s-1)}]^{-1} - \mathbf{A}$
/*first two terms in Eq. 6*/
7. Compute $\vec{w}' := \mathbf{C}(-\vec{x}_{s-1})$
8. Compute $a' := \vec{x}_{s-1}^T\vec{w}'$
9. Compute $\vec{v}^T := \vec{x}_{s-1}^T\mathbf{C}$
10. Compute $\mathbf{D} := \vec{w}'\vec{v}^T$
11. Compute $\mathbf{E} := \mathbf{D}/(1+a')$ /*third term in Eq. 6*/
12. Compute $[\mathbf{P}^{(s)}]^{-1} := \mathbf{C} - \mathbf{E}$
/*new inverse covariance matrix*/
13. Compute $\hat{y}_{s+k-1} := \vec{x}_{s+k-1}^T\vec{\beta}^{(s-1)}$
14. Compute $e := y_{s+k-1} - \hat{y}_{s+k-1}$
15. Compute $h := e/(1+a)$
16. Compute $\vec{r} := h\vec{w}$
17. Compute $\vec{q} := \vec{\beta}^{(s-1)} + \vec{r}$ /*first two terms in Eq. 7*/
18. Compute $\hat{y}'_{s-1} := \vec{x}_{s-1}^T\vec{q}$
19. Compute $b := y_{s-1} - \hat{y}'_{s-1}$
20. Compute $d := b/(1+a')$
21. Compute $\vec{h} := d\vec{w}'$ /* third term in Eq. 7 */
22. Compute $\vec{\beta}^{(s)} := \vec{q} + \vec{h}$ /* new coefficients*/
23. Return $\vec{\beta}^{(s)}$ and $[\mathbf{P}^{(s)}]^{-1}$

Starting with the initial cost model $M^{(0)}$, Algorithm 1 can be repeatedly applied for each new sample data point to evolve the cost model for capturing the dynamic environment. The complexity of Algorithm 1 is given in the following corollary.

**Corollary 1** *Algorithm 1 requires* $8(n+1)^2 + 7(n+1) + 2$ *number of scalar multiplications and divisions to calculate* $\vec{\beta}^{(s)}$ *and* $[\mathbf{P}^{(s)}]^{-1}$ *for new cost model* $M^{(s)}$ *based on* $\vec{\beta}^{(s-1)}$ *and* $[\mathbf{P}^{(s-1)}]^{-1}$ *of previous cost model* $M^{(s-1)}$*, where* $n$ *is the number of explanatory variables in the cost model.*

*Proof.* Correctness can be easily checked by counting the number of scalar multiplications and divisions required by each step in Algorithm 1. □

From Corollary 1, we can see that the asymptotic performance behavior of the shifting method for one step is $\theta(n^2)$ rather than $\theta(n^3)$ as required by the rebuilding approach via multiple regression for one step (Eq.4).

Another very attractive advantage of the shifting method is that its complexity is independent of sample size $k$. To achieve a better cost model, the rebuilding approach has to employ a larger sample size, which implies a larger overhead, while the shifting method can obtain the same cost model without increasing any overhead. The reason for this phenomenon is that, no matter how large the sample set is, the difference between a new cost model and its previous cost model is only two sample data points, and the shifting method fully exploits the shared work for building two cost models based on the common sample data points rather than building the new cost model from scratch.

Furthermore, based on the fact that $k \geq 10(n+2)$ (see footnote 2), we can show that the shifting method for one step is always more efficient than the rebuilding approach for one step for any $n \geq 0$. Clearly, the more steps the shifting method and the rebuilding approach are applied for, the more performance gain the shifting method will achieve. Therefore, we have the following important conclusion:

**Corollary 2** *The shifting method is more efficient than the direct rebuilding approach for evolving a cost model in a dynamic environment.*

## 4 The block-moving method

The shifting method adjusts the cost model every time a new sample data point is observed. The cost model is kept up-to-date in this way. However, if the environment changes very slowly, people may want to accumulate several sample data points and use them to update the cost model all at once to reduce the updating overhead. On the other hand, people sometimes may want to run several sample queries in the current environment and use their observed data to update the cost model to capture the environment. In either case, we need to update the cost model based on a batch/block of sample data points rather than one individual sample data point.

Clearly, the block size should never be larger than the given sample size. If the block size equals the sample size, we can simply employ the rebuilding approach to update the cost model. If the block size is less than the sample size, as with the shifting method, we can derive recurrence formulas to calculate a new cost model based on its previous cost model for each block of sample data points observed.

Assume that we have cost model $M^{(s-1)}$ (including its coefficients $\vec{\beta}^{(s-1)}$ and inverse covariance matrix $[\mathbf{P}^{(s-1)}]^{-1}$) at time $t_{s+k-2}$ ($s = 1, 2, \ldots$) based on $k$ sample data points $(y_{s-1}, \vec{x}_{s-1}^T)$, $(y_s, \vec{x}_s^T)$, …, $(y_{s+k-2}, \vec{x}_{s+k-2}^T)$. At time $t_{s+k+m-2}$, we want to obtain new cost model $M^{(s+m-1)}$ based on $k$ sample data points $(y_{s+m-1}, \vec{x}_{s+m-1}^T)$, $(y_{s+m}, \vec{x}_{s+m}^T)$,…, $(y_{s+k+m-2}, \vec{x}_{s+k+m-2}^T)$. Unless block size $m = k$, there are some common sample data points $(y_{s+m-1}, \vec{x}_{s+m-1}^T), \ldots, (y_{s+k-2}, \vec{x}_{s+k-2}^T)$ $(1 \leq m < k)$ shared by cost models $M^{(s-1)}$ and $M^{(s+m-1)}$. To save some work done previously for $M^{(s-1)}$, we may want to use recurrence formulas to calculate coefficients $\vec{\beta}^{(s+m-1)}$ and

inverse covariance matrix $[\mathbf{P}^{(s+m-1)}]^{-1}$ for new cost model $M^{(s+m-1)}$ based on (1) the previous coefficients $\vec{\beta}^{(s-1)}$; (2) the previous inverse covariance matrix $[\mathbf{P}^{(s-1)}]^{-1}$; (3) the newest block of sample data points $(y_{s+k-1}, \vec{x}^T_{s+k-1})$, ..., $(y_{s+k+m-2}, \vec{x}^T_{s+k+m-2})$; and (4) the oldest block of sample data points $(y_{s-1}, \vec{x}^T_{s-1}), \ldots, (y_{s+m-2}, \vec{x}^T_{s+m-2})$.

Figure 3 can still illustrate the idea of this method, except that (1) the newest sample data point should be replaced by the newest block of sample data points and (2) the oldest sample data point should be replaced by the oldest block of sample data points. Since this method updates the cost model for each new block of sample data points instead of each individual new sample data point, we call this method the block-moving method.

The following theorem specifies the recurrence formula for updating the coefficients of the cost model.

**Theorem 3** *The coefficients $\vec{\beta}^{(s+m-1)}$ of new cost model $M^{(s+m-1)}$ can be recursively calculated by adjusting the coefficients $\vec{\beta}^{(s-1)}$ of previous cost model $M^{(s-1)}$ ($s = 1, 2, \ldots$) as follows:*

$$\vec{\beta}^{(s+m-1)} = (\mathbf{I} - \mathbf{H}\,\mathbf{P}_{rem})^{-1}(\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}\vec{\beta}^{(s-1)} + (\mathbf{I} - \mathbf{H}\,\mathbf{P}_{rem})^{-1}\mathbf{H}\,(\vec{b}_{add} - \vec{b}_{rem}) \quad (8)$$

*where*

$$\mathbf{P}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i\vec{x}_i^T,\ \mathbf{P}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i\vec{x}_i^T$$

$$\vec{b}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i y_i,\ \vec{b}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i y_i$$

$$\mathbf{H} = (\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}[\mathbf{P}^{(s-1)}]^{-1}$$

*and* $\mathbf{I}$ *is the* $(n+1) \times (n+1)$ *identity matrix.*

*Proof.* See Appendix. □

As with the shifting method, to apply the block-moving method we also need to keep the inverse covariance matrix updated for each block. The following theorem specifies the recurrence formula for updating the inverse covariance matrix of the cost model.

**Theorem 4** *The inverse covariance matrix $[\mathbf{P}^{(s+m-1)}]^{-1}$ of new cost model $M^{(s+m-1)}$ can be recursively calculated based on the inverse covariance matrix $[\mathbf{P}^{(s-1)}]^{-1}$ of previous cost model $M^{(s-1)}$ ($s = 1, 2, \ldots$) as follows:*

$$[\mathbf{P}^{(s+m-1)}]^{-1} = \{\mathbf{I} - ((\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}[\mathbf{P}^{(s-1)}]^{-1})\mathbf{P}_{rem}\}^{-1} (\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}[\mathbf{P}^{(s-1)}]^{-1} \quad (9)$$

*where*

$$\mathbf{P}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i\vec{x}_i^T,\ \mathbf{P}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i\vec{x}_i^T$$

*and* $\mathbf{I}$ *is the* $(n+1) \times (n+1)$ *identity matrix.*

*Proof.* See Appendix. □

Since Eqs. 8 and 9 share some common terms, the following algorithm efficiently updates both the coefficients and the inverse covariance matrix of a cost model using the block-moving method:

**Algorithm 2** Cost model evolution based on the block-moving method.

**Input**: (1) Coefficients $\vec{\beta}^{(s-1)}$ of previous model $M^{(s-1)}$; (2) inverse covariance matrix $[\mathbf{P}^{(s-1)}]^{-1}$ of previous model $M^{(s-1)}$; (3) newest block of sample data points $(y_{s+k-1}, \vec{x}^T_{s+k-1})$, ...,$(y_{s+k+m-2}, \vec{x}^T_{s+k+m-2})$; (4) oldest block of sample data points $(y_{s-1}, \vec{x}^T_{s-1}), \ldots, (y_{s+m-2}, \vec{x}^T_{s+m-2})$.

**Output**: (1) Coefficients $\vec{\beta}^{(s+m-1)}$ of new model $M^{(s+m-1)}$; (2) inverse covariance matrix $[\mathbf{P}^{(s+m-1)}]^{-1}$ of new model $M^{(s+m-1)}$.

**Method**:

1. Compute $\mathbf{P}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i\vec{x}_i^T$
2. Compute $\mathbf{F} = [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add}$
3. Compute $\mathbf{G} = \mathbf{I} + \mathbf{F}$
4. Compute $\mathbf{G}^{-1}$
5. Compute $\mathbf{H} = \mathbf{G}^{-1}[P^{(s-1)}]^{-1}$
6. Compute $\mathbf{P}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i\vec{x}_i^T$
7. Compute $\mathbf{J} = \mathbf{H}\mathbf{P}_{rem}$
8. Compute $\mathbf{K} = \mathbf{I} - \mathbf{J}$
9. Compute $\mathbf{K}^{-1}$
10. Compute $[\mathbf{P}^{(s+m-1)}]^{-1} = \mathbf{K}^{-1}\mathbf{H}$

/*new covariance from Eq. 9*/

11. Compute $\vec{o} = \mathbf{G}^{-1}\vec{\beta}^{(s-1)}$
12. Compute $\vec{m} = \mathbf{K}^{-1}\vec{o}$; /*first term in Eq. 8*/
13. Compute $\vec{b}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i y_i$
14. Compute $\vec{b}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i y_i$
15. Compute $\vec{d}_{one} = \vec{b}_{add} - \vec{b}_{rem}$
16. Compute $\vec{n} = [\mathbf{P}^{(s+m-1)}]^{-1}\vec{d}_{one}$

/*second term in Eq. 8*/

17. Compute $\vec{\beta}^{(s+m-1)} = \vec{m} + \vec{n}$; /*from Eq. 8*/
18. Return $\vec{\beta}^{(s+m-1)}$ and $[\mathbf{P}^{(s+m-1)}]^{-1}$

Starting with the initial cost model $M^{(0)}$, Algorithm 2 can be repeatedly applied for each new block of sample data points to evolve the cost model for capturing the dynamic environment. The complexity of Algorithm 2 is given in the following corollary.

**Corollary 3** *Algorithm 2 requires $6(n+1)^3 + (2m+3)(n+1)^2 + 2m(n+1)$ number of scalar multiplications and divisions to calculate $\vec{\beta}^{(s+m-1)}$ and $[\mathbf{P}^{(s+m-1)}]^{-1}$ for new cost model $M^{(s+m-1)}$ based on $\vec{\beta}^{(s-1)}$ and $[\mathbf{P}^{(s-1)}]^{-1}$ of previous cost model $M^{(s-1)}$, where $n$ is the number of explanatory variables in the cost model and $m$ is the block size.*

*Proof.* Correctness can be easily checked by counting the number of scalar multiplications and divisions required by each step in Algorithm 2. □

Note that the complexity of Algorithm 2 depends not only on the number $n$ of explanatory variables but also the block size $m$ ($\leq k$). If block size $m$ is very close to the sample size $k$, the block-moving method may not outperform the direct rebuilding approach. The former is superior only if block size

$m$ is relatively small compared with sample size $k$. The following corollary specifies a condition for the block-moving method to be superior.

**Corollary 4** *The block-moving method is more efficient than the rebuilding approach if and only if block size* $m < 1/2[-5(n+1) + (k+3) - 1/(n+2)]$. *For the minimal sample size* $k = 10 * (n+2)$, *the condition becomes* $m < 1/2[5(n+1) + 13 - 1/(n+2)]$.

*Proof.* The claims follow from Eq. 4 and Corollary 3. Lengthy algebraic derivations are omitted. □

Note that the block size determines how many steps we want to combine in the procedure for evolving the cost model. The larger the block size, the more evolutionary overhead is reduced. To obtain a quality cost model, one usually uses a sample with a large size (much larger than the minimal one). This allows a large block size to be used for the block-moving method. However, a large block size may cause the evolution of the cost model to be not smooth. Thus a fixed number (e.g., 10) that is much smaller than the sample size is usually chosen as the block size for the block-moving method. As with the shifting method, the performance of the block-moving method is independent of the sample size, but it guarantees the same quality of the cost model produced by the rebuilding approach with a large sample.

Comparing the block-moving method with the shifting method, it is not difficult to see from Corollaries 1 and 3 that applying the shifting method once is more efficient than applying the block-moving method once for block size $m > 1$. This is because the latter needs to take more new sample data points into consideration for updating the cost model. However, to obtain the same model $M^{(s+m-1)}$ from old model $M^{(s-1)}$, the shifting method has to be applied $m$ times, while the block-moving method only needs to be applied once. Hence the complexity for obtaining $M^{(s+m-1)}$ from $M^{(s-1)}$ via the shifting method is:

$$8m(n+1)^2 + 7m(n+1) + 2m \tag{10}$$

Comparing Eq. 10 with Corollary 3, we notice that the block-moving method is more efficient when $m$ is sufficiently large, while the (repeated) shifting method is more efficient when $m$ is very small. The following corollary gives a condition for each method to be superior.

**Corollary 5** *When* $m \geq (n+1)$, *the block-moving method is more efficient than the shifting method. When* $m < 3(n+1)^2/(5n+7)$, *the shifting method is more efficient than the block-moving method.*

*Proof.* The claims follow from Eq. 10 and Corollary 3. Lengthy algebraic derivations are omitted. □

Note that Corollary 4 gives a sufficient and necessary condition, while Corollary 5 gives two sufficient conditions only.

From Corollaries 4 and 5, we know that block size $m$ cannot be too small (e.g., $m \geq (n+1)$) in order for the block-moving method to outperform the shifting method, and block size $m$ cannot be too large (i.e., $m < 1/2[5(n+1) + 13 - 1/(n+2)]$) in order for the block-moving method to outperform the rebuilding approach. Clearly we have the following conclusion:

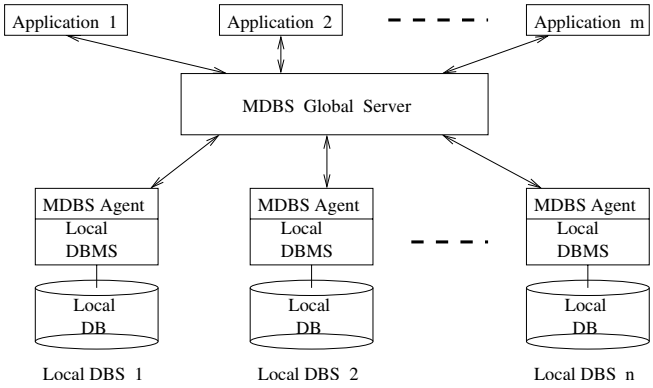


**Fig. 4.** A multidatabase system architecture

**Corollary 6** *For an appropriately chosen block size, the block-moving method can be superior to both the rebuilding approach and the shifting method in terms of time efficiency.*

Be aware that some intermediate cost models are skipped if the block-moving method is applied compared with the shifting method. Therefore, the latter is still a better choice if smooth evolution and accuracy are desired.

## 5 Implementation considerations

The cost model evolution techniques discussed in the previous sections were developed for a multidatabase environment running the multidatabase prototype CORDS-MDBS [2].

Figure 4 shows the system architecture for CORDS-MDBS. In CORDS-MDBS, the global data model is assumed to be relational and each local DBS is associated with an MDBS agent that provides a relational interface if the local DBMS is nonrelational. Thus the global query optimizer in the MDBS may view participating local DBMSs as relational ones. Note that the cost model evolution techniques suggested in this paper do not rely on the relational data model. As long as a query class and the explanatory variables of its cost model are identified, the techniques can be applied directly to evolve the cost model even if the data model is nonrelational.

Initially, the query sampling method in [21,25] can be employed to classify local queries and develop a cost model for each query class at a local site. An evolutionary technique such as the shifting or block-moving method can then be used to evolve the cost model for each query class to capture the slowly changing dynamic environment. The cost models are kept in the MDBS catalog and used during global query optimization.

A question is how to obtain sample queries to update the cost model. One way is to generate artificial sample queries and run them from time to time. A disadvantage of this way is that these sample queries compete for system resources with user queries. To fully utilize the work done in the system, a better way is to make use of user queries as sample queries. More specifically, when a local DBS is requested to run a local user query by either the global query optimizer (due to the decomposition of a global query) or a local user, a query classifier in the MDBS agent will identify the class to which the query belongs. This query is then used as a sample query for the corresponding query class, and its observed data are used

to evolve the relevant cost model via the chosen evolutionary technique.

One potential problem that needs to be considered here is the starvation problem. It is possible that user queries for a particular query class are not executed as frequently as required to capture the dynamic environment. If user queries for a query class are in fact never used in an application environment, the solution is simple – we can simply ignore the update of the corresponding cost model since we only need to maintain those cost models that are useful for practical user queries. However, if the query class is indeed needed in applications but its queries are not run as often as the environment changes, one solution in this case is to generate and run some additional artificial sample queries when necessary to supplement the user sample queries. The system makes full use of user queries and also keeps the cost model up-to-date, although some extra overhead cannot be avoided in this case.

The opposite of the starvation problem is the overflow problem. In other words, user queries are executed too frequently for a particular query class. In this case, the employed evolutionary technique should not take every user query as a sample query to update the cost model because there might be very little change in the underlying environment since the last update. One solution to this problem is to periodically pick up a user query as a sample query to evolve the cost model. Another solution is to activate the cost model updating procedure whenever an observable change in the environment is detected (e.g., the number of queries with a large error of cost estimation is beyond a threshold) and to deactivate the cost model updating procedure when the cost model is up-to-date. In a practical system, the evolution procedure can also be controlled manually via some system configuration parameters or system commands.

As pointed out previously, the evolutionary techniques developed in this paper are suitable for capturing the effect of the slowly changing factors on a cost model in a dynamic environment. These factors usually change little by little (i.e., do not cause an abrupt dramatic change to the underlying system environment at any time). However, a significant environmental change may be accumulated after a certain period of time (e.g., a couple of days, weeks, or months). We need to keep the cost model updated as the environment changes. If the environmental change exceeds a certain level when the next sample data point is obtained, the shifting method should be employed to update the cost model using the new sample data point. Otherwise, the newly observed sample data point is kept in a block (set) to be used by the block-moving method for updating the cost model at a later time. Note that if the environmental change is too small (negligible) when the next sample data point is obtained, this sample data point can be skipped since it has little effect on the cost model (notice that if the environment changes very slowly, we do not need to update the cost model often). Hence we assume that the next sample data point is obtained when the underlying environment experiences an observable (nonnegligible) change since the last sample data point.

In fact, the shifting method and the block-moving method can be applied together in the following integrated scheme to evolve a cost model for a slowly changing environment. Let $M$ be the current cost model and $Q_i, Q_{i+1}, \ldots$ be the incoming sequence of sample queries. Let $\varepsilon(Q_j)$ be the relative error of the estimated cost given by current cost model $M$ for sample query $Q_j (j = i, i+1, \ldots)$. A large estimation error usually indicates that the cost model needs to be updated to reflect the environmental change. Hence if $\varepsilon(Q_i)$ is greater than or equal to a threshold $d_1$ (e.g., 70%, which can be calibrated through experiments), we apply the shifting method to update cost model $M$ using the sample data point for $Q_i$. If $\varepsilon(Q_i)$ is less than threshold $d_1$, the sample data point for $Q_i$ is kept as one of the sample data points in the block to be used by the block-moving method for the next update of the cost model. The subsequent errors $\varepsilon(Q_{i+1}), \varepsilon(Q_{i+2}), \ldots \varepsilon(Q_{i+(m-1)})$ are examined until either (1) $\varepsilon(Q_{i+(m-1)})$ is larger than threshold $d_1$ or (2) the number $m$ of kept sample data points reaches a block size chosen based on Corollaries 4 and 5. The block-moving method is then applied to update current cost model $M$ using the block of $m$ sample data points for sample queries $Q_i$, $Q_{i+1}$, $\ldots Q_{i+(m-1)}$. Note that if the number $m$ of kept sample data points is too small in case (1), to improve the performance (see Corollary 5), the shifting method (rather than the block-moving method) could be repeatedly applied to each sample data point in the block to achieve the same final updated cost model. This integrated scheme aims to achieve both good efficiency and smoothness of a cost model evolution, taking advantage of both the shifting and block-moving methods.

Note that the rebuilding approach can always be applied to update a cost model. However, its efficiency is usually not as good as the evolutionary techniques. To reduce overhead, we cannot apply the rebuilding approach frequently. As a result, the cost model may not change smoothly. Every time a cost model changes significantly, many compiled (optimized) queries may need to be reoptimized for the new environment, which causes the system to be jammed with reoptimization jobs. A smooth change of the cost model helps the job scheduler to properly schedule reoptimization jobs so that a system jam is avoided. Hence the evolutionary techniques are the efficient and seamless approaches to evolving cost models in a slowly changing environment.

However, there are cases for which the evolutionary techniques are not Suitable, for example, if the environment stays unchanged for a long time (e.g., many months/years) and then suddenly experiences a dramatic change (e.g., caused by hardware or software upgrade). The evolutionary techniques are no longer applicable since all previous sample data points become useless and a new set of sample data points are needed to establish the new cost model. The monolithic rebuilding approach can be applied in such a case. On the other hand, if the environment changes very rapidly (e.g., within a few seconds, minutes, or hours), the evolutionary techniques are not applicable either since an evolutionary cost model may not be stable. Special techniques such as the qualitative approach [19], the fractional analysis approach [20], and the probabilistic approach [20] are needed to solve query cost estimation issues in such an environment. Besides, the conventional dynamic query optimization techniques could also be adopted in such a case.

It is worth pointing out that, as another application, our evolutionary techniques can be adopted in self-managing DBSs, which have attracted many researchers recently [3,4, 9,22,26]. A self-managing DBS can automatically update its configuration parameters and optimization statistics to reflect a changing environment while the system is in use. Since all

updates are done on the fly, the overhead of such a technique is required to be as small as possible. So far no technique has been proposed to automatically adjust the parameters of a cost model in such a system. Our techniques in this paper can be utilized to solve this issue in such systems.

## 6 Experimental results

To examine the effectiveness and efficiency of the evolutionary techniques discussed in the previous sections, we conducted extensive experiments in our multidatabase environment, where Oracle 8.0 and DB2 5.0 were used as local DBMSs running under SunOS 5.1 on SUN UltraSparc 2 workstations.

Note that conducting extensive experiments in a real slowly changing environment is infeasible since it would take too long to complete the experiments. To effectively simulate a slowly changing dynamic environment, we artificially generated different numbers of concurrent processes with various work/sleep ratios to change the system contention level in a given environment. The cost of a small probing query is used to gauge the system contention level. The higher the probing query cost, the higher the system contention level. Forty-nine system contention levels (with the probing query cost ranging from 3 seconds to 98 seconds) were considered in both the Oracle and DB2 environments. A slowly changing environment was achieved by assuming that the system contention level gradually changes from the lowest level 1 to the highest level 49. Note that we basically adjusted the frequently changing factors (i.e., CPU load, memory usage, etc.), which are easy to manipulate, to change the environment and kept each change stable for some time to simulate a slowly changing environment. The essentials of our evolutionary techniques are their capability to evolve a cost model to capture the environmental change (reflected in the updated coefficients), no matter what causes the environment to change. The main purpose of our experiments is to verify the evolution capabilities of the techniques. Hence the above simulated environments are reasonable for the experiments.

The experimental databases used in the experiments were the same as those in [19–21]. More specifically, each local database consists of 12 tables $R_i(a_1, a_2, \ldots, a_j)$ ($i = 1, 2, \ldots, 12; j \in \{3, 5, 7, 9, 11, 13\}$) with data randomly generated and cardinalities ranging from 3,000 $\sim$ 250,000. Each table has a number of indexed columns and various selectivities for different columns.

Note that our evolutionary techniques do not rely on any particular query class. The experimental results for all query classes are similar. In this section, we report typical experimental results for a representative query class $G_{15}$ (defined in [21]) consisting of unary queries that have no usable indexes in their qualification conditions. To demonstrate that the techniques have a similar behavior for other query classes, we also report some experimental results for another query class $G_{14}$ consisting of unary queries that have usable (nonclustered) indexes in their qualification conditions.

The sample queries used to evolve cost models were randomly chosen from the relevant query class. One hundred sample queries were executed at each system contention level on both Oracle 8.0 and DB2 5.0. Hence a total of 4900 sample queries were executed in a sequence for each environment. The first 100 sample queries (i.e., sample size $k = 100$) were used to derive the initial cost model (with $n = 6$ variables, i.e., the cardinality of the operand table, the cardinality of the result table, the tuple length of the operand table, the tuple length of the result table, the physical size of the intermediate table, and the physical size of the operand table [21]) via the query sampling method. The effect of environmental factors, such as physical data distribution and system buffer setup, on query performance is reflected in the coefficients of the variables in the cost model. The evolutionary techniques were then used to evolve the cost model (i.e., updating the coefficients) to capture the dynamic environment. The shifting method was used to adjust the cost model every time a new sample query point was observed from the execution sequence, while the block-moving method was used to adjust the cost model every time a block of sample query points was observed from the execution sequence.

To examine the accuracy of the evolutionary cost models obtained from the evolutionary techniques, we also ran some test queries randomly chosen from the query class in both the Oracle and DB2 environments. The cost of each test query was estimated by using the corresponding updated cost model in the environment, and the observed and estimated costs were compared. To see accuracy gains from the evolution techniques, the cost estimates using the initial (static) cost model were also compared.

Note that, unlike scientific computation in engineering, the accuracy of cost estimation in query optimization is not required to be very high. In analysis on the experiments, the cost estimates with relative errors within 30% are considered to be very good, and the cost estimates that are within the range of one-time larger or smaller than the corresponding observed costs (e.g., 2 min vs. 4 min) are considered to be good. Only those cost estimates that are not of the same order of magnitude with the observed costs (e.g., 2 min vs. 3 h) are not acceptable.

Table 1 shows the percentages of good and very good cost estimates for test queries in $G_{15}$ at four representative contention levels (each level has 100 test queries) on Oracle 8.0 and DB2 5.0. In the table, cost estimates from the initial (static) cost model, the evolutionary cost model by the shifting method, and the evolutionary cost model by the block-moving method ($m = 10$) were listed. To show that the experimental results are similar for different query classes, Table 2 lists two representative sets of experimental results for another query class $G_{14}$.

From the experiments, we have the following observations on the effectiveness of the evolutionary techniques:

- The shifting method can derive a good evolutionary cost model to capture a slowly changing dynamic environment. From Table 1 we can see that the evolutionary cost model obtained from the shifting method can give good cost estimates for most test queries (87.5% on average, including 76% very good ones). If we do not evolve the cost model and keep using the initial (static) cost model, we can only obtain on average 1.25% good cost estimates for the test queries in a dynamic environment. The estimation accuracy gains are dramatic. Figures 5 and 6 show a typical comparison for cost estimates from the two cost models in Oracle 8.0 and DB2 5.0, respectively. Note that the actual

**Table 1.** Percentages of good cost estimates for test queries in $G_{15}$ from experiments on Oracle 8.0 and DB2 5.0

| Local DBMS | Contention level | Static: very good% | Static: good% | Shifting: very good% | Shifting: good% | Block-moving: very good% | Block-moving: good% |
|---|---|---|---|---|---|---|---|
| Oracle | 12 | 1% | 5% | 83% | 88% | 78% | 87% |
| | 25 | 0% | 0% | 70% | 82% | 68% | 79% |
| | 37 | 1% | 3% | 63% | 74% | 59% | 74% |
| | 49 | 0% | 1% | 70% | 87% | 68% | 78% |
| DB2 | 12 | 0% | 1% | 85% | 93% | 85% | 90% |
| | 25 | 0% | 0% | 82% | 91% | 78% | 87% |
| | 37 | 0% | 0% | 65% | 88% | 53% | 73% |
| | 49 | 0% | 0% | 90% | 97% | 93% | 97% |
| Average | | 0.25% | 1.25% | 76% | 87.50% | 72.75% | 83.13% |

**Table 2.** Similar experimental results obtained for test queries in $G_{14}$ on Oracle 8.0

| Local DBMS | Contention level | Static: very good% | Static: good% | Shifting: very good% | Shifting: good% | Block-moving: very good% | Block-moving: good% |
|---|---|---|---|---|---|---|---|
| Oracle | 12 | 0% | 1% | 82% | 93% | 81% | 91% |
| | 25 | 1% | 1% | 86% | 92% | 78% | 88% |

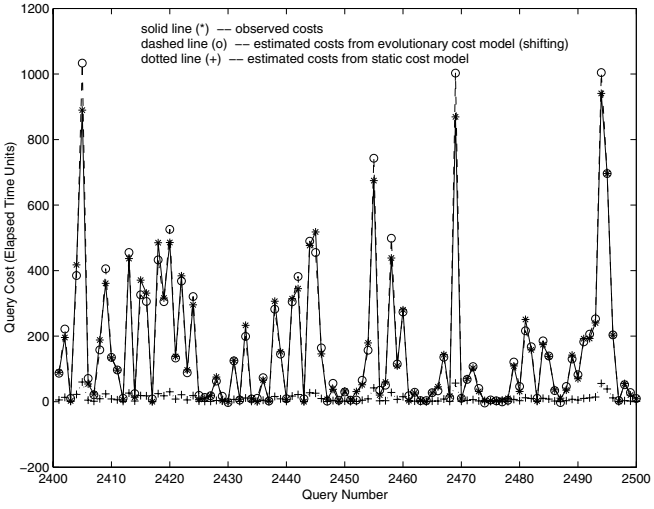


**Fig. 5.** Cost estimates for test queries in $G_{15}$ from static and shifting models at contention level 25 on Oracle 8.0

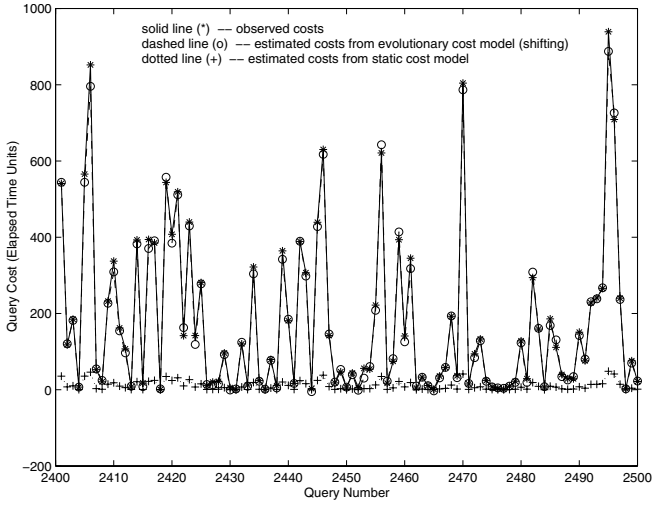


**Fig. 6.** Cost estimates for test queries in $G_{15}$ from static and shifting models at contention level 25 on DB2 5.0

time measuring unit for the relevant commercial systems is not revealed here to avoid a potential license violation.

- The block-moving method can also derive a good evolutionary cost model to capture a slowly changing dynamic environment. From Table 1 we can see that the evolutionary cost model obtained from the block-moving method can give good cost estimates for most test queries (83.13% on average, including 72.75% very good ones), which is much better than the initial static cost model. Figures 7 and 8 show a typical comparison for cost estimates from the two cost models in Oracle 8.0 and DB2 5.0, respectively. Comparing the shifting and block-moving methods, a cost model obtained from the former is more accurate than the one obtained from the latter, as pointed out in Sect. 4. In general, the more rapidly the environment changes and the larger the block size, the better accuracy can be obtained by the shifting method over the block-moving method.
- The initial (static) cost model cannot be used in a dynamic environment. The cost models in most existing DBSs currently do not cope with a dynamic environment. Our experiments demonstrate that using a static cost model can hardly give good cost estimates for queries run in a dynamic environment (see Tables 1 and 2). The situation becomes progressively worse when the environment moves farther and farther away from the original environment. Figure 9 shows a comparison of relative errors of cost estimation for a query run from contention level 1 to 49 in our dynamic environment by the initial (static) cost model and the evolutionary cost model obtained from the shift-

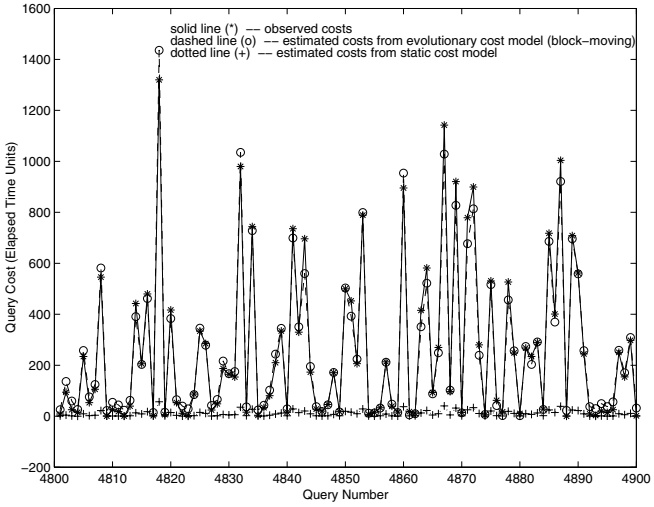


**Fig. 7.** Cost estimates for test queries in $G_{15}$ from static and block-moving models at contention level 49 on Oracle 8.0

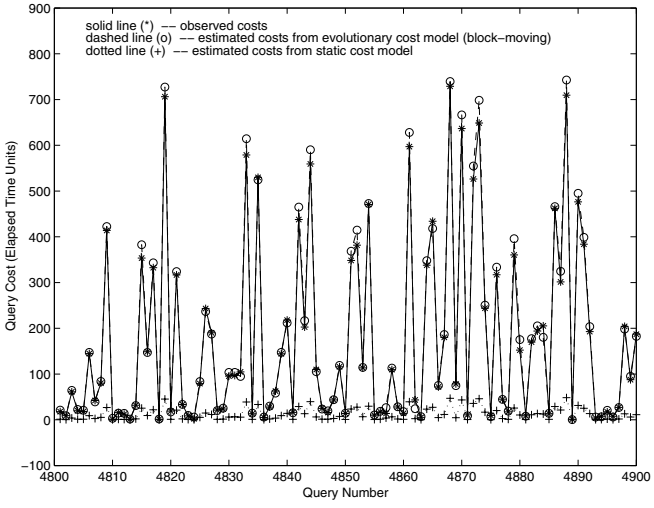


**Fig. 8.** Cost estimates for test queries in $G_{15}$ from static and block-moving models at contention level 49 on DB2 5.0

ing method. From the figure we can see that the relative errors become larger and larger for the static cost model as the environment moves farther and farther away from the initial one, while the relative errors are kept within 30% by the evolutionary cost model no matter how much the environment changes.

To examine the efficiency of the evolutionary techniques, we compared the execution costs of the shifting method, the block-moving method, and the rebuilding approach for various cases. Table 3 shows the execution time units for one invocation (step) of each technique. Table 4 shows the execution time units for repeated invocations (multiple steps) of the shifting method and the rebuilding approach, where $p$ is the number of times the relevant method is invoked to update the cost model for $p$ consecutive new sample data points. Note that the block-moving method is invoked only once for each block of sample data points rather than for each individual point.

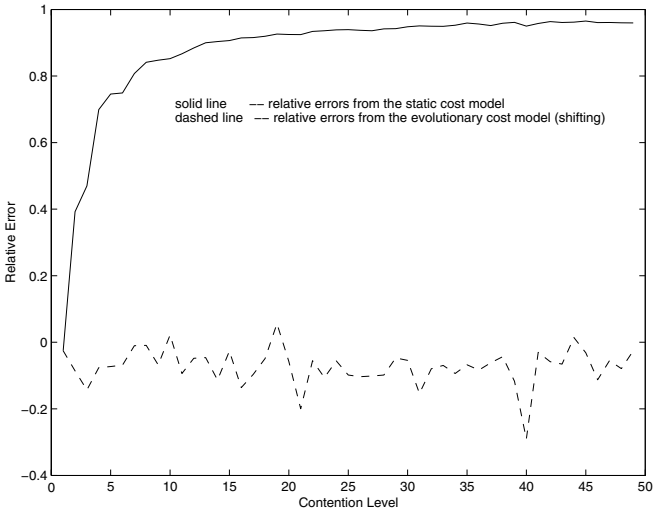


**Fig. 9.** Errors for cost estimates of a query executed at various contention levels from static and evolutionary (shifting) cost models on Oracle 8.0

Hence it is not compared in Table 4. From the experiments, we have the following observations on the efficiency of the evolutionary techniques:

- The shifting method is more efficient than the rebuilding approach, as predicated by Corollary 2 from theoretical analysis. Table 4 indicates that the former can improve efficiency by about 89% for any $p$, i.e., the cost of the rebuilding approach is about eight times larger than that of the shifting method. The more times (steps) the shifting method is invoked, the more (absolute) overhead can be saved (Fig. 10). Note that the sample size (i.e., 100) used in the experiments was close to the minimum (i.e., 80) for the given query class. The above performance improvement is mainly caused by the complexity reduction from $\theta(n^3)$ to $\theta(n^2)$. On the other hand, to improve the quality of a regression cost model, one ought to use a sample with a larger size $k$. Given that the query class for which the cost model was developed contains a large number of queries [21], a sample of 1000 or more queries could be used. However, as pointed out earlier, the cost of the shifting method is independent of sample size $k$, while the cost of the rebuilding approach is proportional to $k$ (Fig. 11). In fact, for a sample of size 1000, the cost of the latter is about 77 times larger than that of the former (to obtain the same result). Furthermore, the query class considered in the experiments was relatively simple. For a more complex query class (e.g., a join query class [21]), both the number $n$ of explanatory variables and the sample size $k$ (increasing with $n$, as indicated in footnote 2) can be much larger. The overhead saving by the shifting method can be even more significant.
- The efficiency of the block-moving method depends on the block size. The larger the block size, the higher the execution cost.
  - When the block size is relatively small (e.g., $m = 3, 5, 10$), invoking the block-moving method once is more efficient than invoking the rebuilding approach

**Table 3.** Execution time units for one-step rebuilding, shifting, and block-moving methods

| Method | RB | S | BM (3) | BM (5) | BM (10) | BM (20) | BM (30) | BM (40) | ... |
|---|---|---|---|---|---|---|---|---|---|
| One-step execution cost | 0.904e−3 | 0.110e−3 | 0.631e−3 | 0.680e−3 | 0.775e−3 | 0.973e−3 | 0.117e−2 | 0.151e−2 | ... |

RB – rebuilding method; S – shifting method; BM ($m$) – block-moving method with block size $m$

**Table 4.** Execution time units for repeated invocations of rebuilding (RB) and shifting (S) methods

| Repeat# $p$ | $p = 1$ | $p = 3$ | $p = 5$ | $p = 10$ | $p = 20$ | $p = 30$ | $p = 40$ | ... | $p = 700$ | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| RB | 0.904e−3 | 0.268e−2 | 0.445e−2 | 0.889e−2 | 0.178e−1 | 0.269e−1 | 0.359e−1 | ... | 0.628e+0 | ... |
| S | 0.110e−3 | 0.305e−3 | 0.498e−3 | 0.981e−3 | 0.195e−2 | 0.282e−2 | 0.397e−2 | ... | 0.682e−1 | ... |

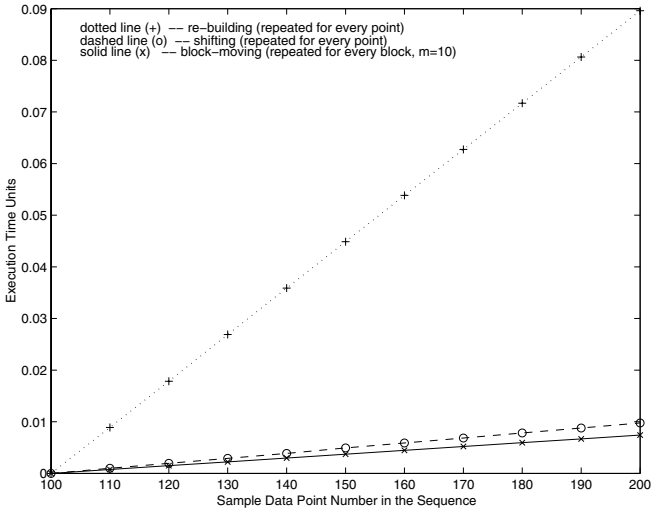


**Fig. 10.** Efficiency comparison for shifting, block-moving ($m = 10$), and rebuilding methods

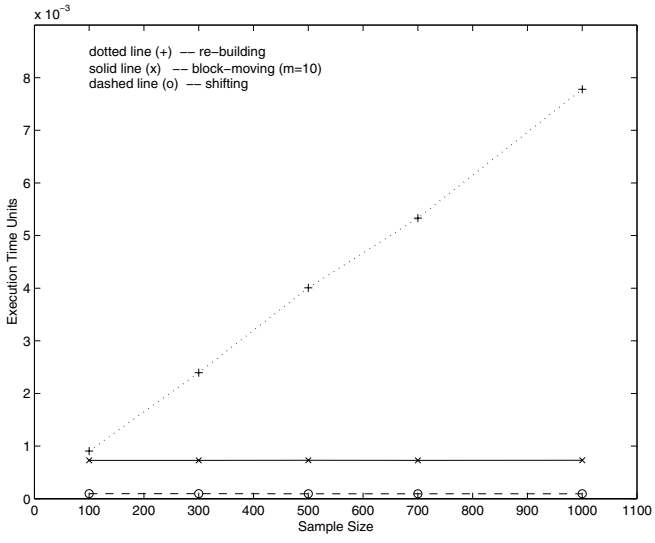


**Fig. 11.** Performance effect of sample size on (one-step) rebuilding, shifting, and block-moving techniques

once (Table 3). The former can improve efficiency as much as 52% in such a case. If the rebuilding approach is invoked multiple times to keep the cost model up-to-date for each new sample data point, the performance gain from the block-moving method can be even dramatically larger. The block-moving method becomes inefficient, compared to the (one-step) rebuilding approach, if the block size is relatively large (e.g., $m = 30$ in Table 3). In such a case, the latter can be used efficiently to keep the cost model up-to-date for each block of sample data points rather than the former. However, the cost of the rebuilding approach increases with sample size $k$, while the cost of the block-moving method is independent of $k$ (Fig. 11). Increasing the sample size (either to improve the cost model quality or to handle a more complex query class) would make the rebuilding approach less efficient than the block-moving method with a larger block size.

– When block size $m$ is not too small (e.g., $m = 10$), invoking the block-moving method once is more efficient than the (repeated) shifting method, which has to be invoked multiple times to keep the cost model up-to-date for every sample data point (including the sample data point at the end of each block). The performance gain can be as much as 62%. Otherwise, the latter may be more efficient (e.g., $m = 3, 5$).
– There are cases (e.g., $m = 10$) in which invoking the block-moving method once is more efficient than using either the (repeated) shifting method or the rebuilding approach (once). If all the methods are invoked repeatedly to keep the cost model up-to-date in such a case, the block-moving method can save a significant amount of overhead over time (Fig. 10).

The observations on the block-moving method are consistent with Corollaries 4 to 6 from our theoretical analysis.

## 7 Conclusions

A major challenge for performing global query optimization in an MDBS is that the cost models for local DBSs may not be available at the global level. Dynamic environmental factors add more difficulties to this problem. In this paper, we have suggested evolving a cost model to capture a slowly changing dynamic multidatabase environment so that the cost model is kept as accurate as possible all the time.

A direct approach to keeping a cost model up-to-date in a dynamic environment is to periodically rebuild the cost model

via the query sampling method [21]. However, this rebuilding approach incurs a high overhead. To improve the efficiency, we propose two new evolutionary techniques, i.e., the shifting method and the block-moving method, in this paper.

The shifting method employs recurrence updating formulas to adjust a cost model by adding the effect of a new sample data point and in the meantime removing the effect of the oldest sample data point from the model every time a new sample data point is observed. This method is more efficient than the rebuilding approach since it simply adjusts the previous cost model rather than rebuilding the cost model from scratch using the entire set of sample data points, most of which have actually already been taken into account in the previous model. Especially when the rebuilding approach has to be repeatedly applied to keep the cost model updated for every new sample data point, the shifting method can save costs dramatically. Furthermore, the overhead of the rebuilding approach is proportional to the sample size used to develop the cost model, while the overhead of the shifting method is independent of the sample size.

Instead of adjusting a cost model for every individual new sample data point, the block-moving method uses recurrence updating formulas to adjust a cost model by adding the effect of a block (batch) of new sample data points and in the meantime removing the effect of the block of the oldest sample data points from the model every time a block of new sample data points are observed. Our analysis shows that when the block size is not too small, the block-moving method is more efficient than applying the shifting method for every individual new sample data point in the block to get the same updated cost model in the end. On the other hand, the block size chosen for the block-moving method cannot be too large. Otherwise, one can simply apply the rebuilding approach since the new block of sample data points is almost the entire set of sample data points on which the updated cost model is based. The block-moving method with an appropriate block size can be more efficient than both the shifting method and the rebuilding approach. However, the accuracy/smoothness of an evolutionary cost model obtained from the block-moving method is worse than that obtained from the shifting method. A trade-off between accuracy and efficiency is required when making a choice between the shifting method and the block-moving method. An integrated scheme to automatically select one of the two methods based on the underlying environment is suggested.

To reduce the overhead of executing sample queries, we suggest using user queries as sample queries so that their observed information can be used to evolve a cost model. The relevant starvation problem can be solved by generating some supplemental artificial sample queries, while the relevant overflow problem can be solved by updating a cost model periodically or only when a significant change to the model has been detected.

Our experimental results are consistent with our theoretical ones, which demonstrate that both the shifting method and the block-moving method are quite promising in maintaining good evolutionary cost models for a slowly changing dynamic multidatabase environment in terms of both effectiveness and efficiency.

Besides the application in MDBSs, the proposed evolutionary techniques can also be used to automatically maintain cost models in self-managing DBSs, which have attracted researchers and practitioners in the database area recently.

*Acknowledgements.* The authors would like to thank Yu Sun for his help in developing some experimental programs for the work reported in this paper. Our grateful thanks are due to anonymous reviewers for their valuable comments and constructive suggestions for improving the paper. The preliminary version of this paper appeared in [14], which did not cover some discussions including the derivation of the block-moving method and its performance analysis and comparison with others.

## References

1. Adali S, Candan KS, Papakonstantinou Y, Subrahmamian VS (1996) Query caching and optimization in distributed mediator systems. In: Proceedings of the 1996 ACM SIGMOD international conference on management of data, Montreal, Canada, 4–6 June 1996, pp 137–148
2. Attaluri GK, Bradshaw DP, Coburn N, Larson PÅ, Martin P, Silberschatz A, Slonim J, Zhu Q (1995) The CORDS multidatabase project. IBM Sys J 34(1):39–62
3. Chaudhuri S, Narasayya V (2000) Automating statistics management for query optimizers. In: Proceedings of the IEEE international conference on data engineering, San Diego, 28 February–3 March 2000, pp 339–348
4. Chaudhuri S, Christensen E, Graefe G, Narasayya V, Zwilling M (1999) Self tuning technology in Microsoft SQL Server. Data Eng Bull 22(2):20–26
5. Du W, Krishnamurthy R, Shan MC (1992) Query optimization in heterogeneous DBMS. In: Proceedings of the 18th international conference on very large data bases, Vancouver, BC, Canada, 23–27 August 1992, pp 277–291
6. Du W, Shan MC, Dayal U (1995) Reducing multidatabase query response time by tree balancing. In: Proceedings of the 1995 ACM SIGMOD international conference on management of data, San Jose, 22–25 May 1995, pp 293–303
7. Gardarin G, Sha F, Tang ZH (1996) Calibrating the query optimizer cost model of IRO-DB, an object-oriented federated database system. In: Proceedings of the 22th international conference on very large databases, Mumbai, India, 3–6 September 1996, pp 378–389
8. Lee C, Chen CJ (1997) Query optimization in multidatabase systems considering schema conflicts. IEEE Trans Knowl Data Eng 9(6):941–955
9. Lightstone S, Lohman GM, Zilio DC (2002) Toward autonomic computing with DB2 universal database. SIGMOD Rec 31(3):55–61
10. Litwin W, Mark L, Roussopoulos N (1990) Interoperability of multiple autonomous databases. ACM Comp Surv 22(3):267–293
11. Lu H, Shan MC (1992) On global query optimization in multidatabase systems. In: Proceedings of the 2nd international workshop on research issues on data engineering, Tempe, AZ, 2–3 February 1992, p 217
12. Naacke H, Gardarin G, Tomasic A (1998) Leveraging mediator cost models with heterogeneous data sources. In: Proceedings of the IEEE international conference on data engineering, Orlando, 23–27 February 1998, pp 351–360
13. Pfaffenberger R, Patterson JH (1987) Statistical methods for business and economics. Richard D Irwin, New York

14. Rahal A, Zhu Q, Larson PÅ(2002) Developing evolutionary cost models for query optimization in a dynamic multidatabase environment. In: Proceedings of the international conference on cooperative information systems, Irvine, CA, 30 October–1 November 2002, pp 1–18
15. Roth MT, Ozcan F, Haas LM (1999) Cost models DO matter: providing cost information for diverse data sources in a federated system. In: Proceedings of the 25th international conference on very large databases, Edinburgh, UK, 7–10 September 1999, pp 599–610
16. Sheth AP, Larson JA (1990) Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comp Surv 22(3):183–236
17. Treicher J, Richard J (1987) Theory and design of adaptive filters. Wiley, New York
18. Urhan T, Franklin MJ, Amsaleg L (1998) Cost-based query scrambling for initial delays. In: Proceedings of the 1998 ACM SIGMOD international conference on management of data, Seattle, 2–4 June 1998, pp 130–141
19. Zhu Q, Sun Y, Motheramgari S (2000) Developing cost models with qualitative variables for dynamic multidatabase environment. In: Proceedings of the IEEE international conference on data engineering, San Diego, 28 February–3 March 2000, pp 413–424
20. Zhu Q, Motheramgari S, Sun Y (2000) Cost estimation for large queries via fractional analysis and probabilistic approach in dynamic multidatabase environments. In: Proceedings of the international conference on database and expert systems, London, 4–8 September 2000, pp 509–525
21. Zhu Q, Larson PÅ(1998) Solving local cost estimation problem for global query optimization in multidatabase systems. Distr Paral Databases 6(4):373–420
22. Zhu Q, Dunkel B, Soparkar N, Chen S, Schiefer B, Lai T (1998) A piggyback method to collect statistics for query optimization in database management systems. In: Proceedings of the 1998 IBM CAS conference, Toronto, 30 November–3 December 1998, pp 67–82
23. Zhu Q, Larson PÅ(1997) A fuzzy query optimization approach for multidatabase systems. Int J Uncert Fuzz Knowl Based Sys 5(6):701–722
24. Zhu Q, Larson PÅ(1996) Building regression cost models for multidatabase systems. In: Proceedings of the 4th IEEE international conference on parallel and distributed information systems, Miami Beach, FL, 18–20 December 1996, pp 220–231
25. Zhu Q, Larson PÅ(1994) A query sampling method for estimating local cost parameters in a multidatabase system. In: Proceedings of the IEEE international conference on data engineering, Houston, 14–18 February 1994, pp 144–153
26. Zilio DC, Lightstone S, Lyons KA, Lohman GM (2001) Self-managing technology in IBM DB2 universal database. In: Proceedings of the international conference on information and knowledge management, Atlanta, 5–10 November 2001, pp 541–543


## Appendix

In this appendix, we sketch the proofs of the main theorems in this paper. The following matrix inversion lemma from [17] is applied in the proof for Theorem 1.

**Lemma 1** *Given four matrices* $\mathbf{W}$, $\mathbf{X}$, $\mathbf{Y}$ *and* $\mathbf{Z}$, *the following equation holds:*

$$(\mathbf{W}+\mathbf{XYZ})^{-1} = \mathbf{W}^{-1}-\mathbf{W}^{-1}\mathbf{X}(\mathbf{ZW}^{-1}\mathbf{X}+\mathbf{Y}^{-1})^{-1}\mathbf{ZW}^{-1}$$

*Proof of Theorem 1*:

Let $\vec{b}^{(l)} = \sum_{i=l}^{(k-1)+l} \vec{x}_i y_i$, where $l = s-1, s$. From Eq. 3, we have

$$\vec{\beta}^{(l)} = [\mathbf{P}^{(l)}]^{-1}\vec{b}^{(l)},\ l = s-1, s \tag{11}$$

Clearly,

$$\vec{b}^{(s)} = \vec{b}^{(s-1)} + \vec{x}_{s+k-1}y_{s+k-1} - \vec{x}_{s-1}y_{s-1} \tag{12}$$

$$\mathbf{P}^{(s)} = \mathbf{P}^{(s-1)} + \vec{x}_{s+k-1}\vec{x}^T_{s+k-1} - \vec{x}_{s-1}\vec{x}^T_{s-1} \tag{13}$$

Let

$$\mathbf{C}^{-1} = \mathbf{P}^{(s-1)} + \vec{x}_{s+k-1}\vec{x}^T_{s+k-1}$$

By Lemma 1, we get

$$\begin{aligned}\mathbf{C} &= [\mathbf{P}^{(s-1)}]^{-1} - [\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1} \\ &\quad (\vec{x}^T_{s+k-1}[\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1}+1)^{-1}\vec{x}^T_{s+k-1}[\mathbf{P}^{(s-1)}]^{-1} \\ &= [\mathbf{P}^{(s-1)}]^{-1} - (\vec{w}\vec{x}^T_{s+k-1}[\mathbf{P}^{(s-1)}]^{-1})/(1+a)\end{aligned} \tag{14}$$

where $\vec{w} = [\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1}$ and $a = \vec{x}^T_{s+k-1}\vec{w}$. Note that $a$ is a scalar.

From Eq. 13 and Lemma 1 we have

$$\begin{aligned}[\mathbf{P}^{(s)}]^{-1} &= (\mathbf{C}^{-1} - \vec{x}_{s-1}\vec{x}^T_{s-1})^{-1} \\ &= \mathbf{C} - \mathbf{C}(-\vec{x}_{s-1})(\vec{x}^T_{s-1}\mathbf{C}(-\vec{x}_{s-1})+1)^{-1}\vec{x}^T_{s-1}\mathbf{C} \\ &= \mathbf{C} - (\vec{w}'\vec{x}^T_{s-1}\mathbf{C})/(1+a')\end{aligned} \tag{15}$$

where $\vec{w}' = \mathbf{C}(-\vec{x}_{s-1})$ and $a' = \vec{x}^T_{s-1}\vec{w}'$. Note that $a'$ is a scalar.

Let

$$\vec{q} = \mathbf{C}(\vec{b}^{(s-1)} + \vec{x}_{s+k-1}y_{s+k-1})$$

By Eqs. 14 and 11, we get

$$\begin{aligned}\vec{q} &= ([\mathbf{P}^{(s-1)}]^{-1} - (\vec{w}\vec{x}^T_{s+k-1}[\mathbf{P}^{(s-1)}]^{-1})/(1+a))(\vec{b}^{(s-1)} \\ &\quad +\vec{x}_{s+k-1}y_{s+k-1}) \\ &= \vec{\beta}^{(s-1)} - (\vec{w}\vec{x}^T_{s+k-1}\vec{\beta}^{(s-1)})/(1+a) \\ &\quad +[\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1}y_{s+k-1} \\ &\quad -(\vec{w}\vec{x}^T_{s+k-1}[\mathbf{P}^{(s-1)}]^{-1}\vec{x}_{s+k-1}y_{s+k-1})/(1+a) \\ &= \vec{\beta}^{(s-1)} - (\vec{w}\vec{x}^T_{s+k-1}\vec{\beta}^{(s-1)})/(1+a) + \vec{w}y_{s+k-1} \\ &\quad -\vec{w}y_{s+k-1}(a/(1+a)) \\ &= \vec{\beta}^{(s-1)} + \vec{w}(e/(1+a))\end{aligned} \tag{16}$$

where $e = y_{s+k-1} - \vec{x}^T_{s+k-1}\vec{\beta}^{(s-1)}$, which is the error term for sample data point $(y_{s+k-1}, \vec{x}^T_{s+k-1})$ using cost model $M^{(s-1)}$.

Let $\vec{b}' = \vec{b}^{(s-1)} + \vec{x}_{s+k-1}y_{s+k-1}$. From Eqs. 11, 12 and 15, we have

$$\begin{aligned}\vec{\beta}^{(s)} &= [\mathbf{P}^{(s)}]^{-1}\vec{b}^{(s)} = [\mathbf{P}^{(s)}]^{-1}(\vec{b}' - \vec{x}_{s-1}y_{s-1}) \\ &= (\mathbf{C} - (\vec{w}'\vec{x}^T_{s-1}\mathbf{C})/(1+a'))(\vec{b}' - \vec{x}_{s-1}y_{s-1}) \\ &= \vec{q} - (\vec{w}'\vec{x}^T_{s-1}\vec{q})/(1+a') + \mathbf{C}(-\vec{x}_{s-1})y_{s-1} \\ &\quad -(\vec{w}'\vec{x}^T_{s-1}\mathbf{C})/(1+a')(-\vec{x}_{s-1})y_{s-1} \\ &= \vec{q} - (\vec{w}'\vec{x}^T_{s-1}\vec{q})/(1+a') + \vec{w}'y_{s-1} \\ &\quad -(\vec{w}'a'y_{s-1})/(1+a') \\ &= \vec{q} - (\vec{w}'\vec{x}^T_{s-1}\vec{q})/(1+a') + (\vec{w}'y_{s-1})/(1+a')\end{aligned}$$

From Eq. 16, we get

$$\begin{aligned}\vec{\beta}^{(s)} &= \vec{q} - \vec{w}'\vec{x}_{s-1}^T\{\vec{\beta}^{(s-1)} + (e/(1+a))\vec{w}\}/(1+a') \\ &\quad +(\vec{w}'y_{s-1})/(1+a') \\ &= \vec{q} - \vec{w}'\vec{x}_{s-1}^T\vec{\beta}^{(s-1)}/(1+a') - \{\vec{w}'\vec{x}_{s-1}^T \\ &\quad (e/(1+a))\vec{w}\}/(1+a') + (\vec{w}'y_{s-1})/(1+a') \\ &= \vec{q} - \{\vec{w}'\vec{x}_{s-1}^T(e/(1+a))\vec{w}\}/(1+a') \\ &\quad +\{\vec{w}'(y_{s-1} - \vec{x}_{s-1}^T\vec{\beta}^{(s-1)})\}/(1+a') \\ &= \vec{q} - \vec{w}'\vec{x}_{s-1}^T(e/(1+a))\vec{w}/(1+a') \\ &\quad +\vec{w}'e'/(1+a') \\ &= \vec{\beta}^{(s-1)} + \vec{w}(e/(1+a)) - \{\vec{w}'\vec{x}_{s-1}^T\vec{w}(e/(1+a)) \\ &\quad /(1+a')\} + \vec{w}'e'/(1+a') \\ &= \vec{\beta}^{(s-1)} + \vec{w}(e/(1+a)) + \vec{w}'\{e' \\ &\quad -\vec{x}_{s-1}^T\vec{w}(e/(1+a))\}/(1+a')\end{aligned}$$

where $e' = y_{s-1} - \vec{x}_{s-1}^T\vec{\beta}^{(s-1)}$, which is the error term for sample data point $(y_{s-1}, \vec{x}_{s-1}^T)$ using cost model $M^{(s-1)}$. □

*Proof of Theorem 2*:
Equation 6 can be obtained by substituting Eq. 14 in Eq. 15. □

*Proof of Theorem 3*:
Let $\vec{b}^{(l)} = \sum_{i=l}^{(k-1)+l} \vec{x}_i y_i$, where $l = s-1, s+m-1$. From Eq. 3, we have

$$\vec{\beta}^{(l)} = [\mathbf{P}^{(l)}]^{-1}\vec{b}^{(l)},\ l = s-1, s+m-1 \tag{17}$$

Clearly,

$$\vec{b}^{(s+m-1)} = \vec{b}^{(s-1)} + \vec{b}_{add} - \vec{b}_{rem} \tag{18}$$

$$\mathbf{P}^{(s+m-1)} = \mathbf{P}^{(s-1)} + \mathbf{P}_{add} - \mathbf{P}_{rem} \tag{19}$$

where

$$\vec{b}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i y_i,\ \vec{b}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i y_i$$

$$\mathbf{P}_{add} = \sum_{i=s+k-1}^{s+k+m-2} \vec{x}_i\vec{x}_i^T,\ \mathbf{P}_{rem} = \sum_{i=s-1}^{s+m-2} \vec{x}_i\vec{x}_i^T$$

Let

$$\mathbf{H}^{-1} = \mathbf{P}^{(s-1)} + \mathbf{P}_{add} \tag{20}$$

Multiplying Eq. 20 by $\mathbf{H}$ from the right, we get

$$\mathbf{I} = \mathbf{P}^{(s-1)}\mathbf{H} + \mathbf{P}_{add}\mathbf{H}, \tag{21}$$

where $\mathbf{I}$ is the $(n+1) \times (n+1)$ identity matrix. Multiplying Eq. 21 by $[\mathbf{P}^{(s-1)}]^{-1}$ on the left, we get

$$\begin{aligned}[\mathbf{P}^{(s-1)}]^{-1} &= \mathbf{H} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add}\mathbf{H} \\ &= (\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})\mathbf{H}\end{aligned} \tag{22}$$

Multiplying (22) by $(\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}$ from the left, we get

$$\mathbf{H} = (\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}[\mathbf{P}^{(s-1)}]^{-1} \tag{23}$$

According to Eq. 19 and 20, we have

$$\mathbf{P}^{(s+m-1)} = \mathbf{H}^{-1} - \mathbf{P}_{rem} \tag{24}$$

Multiplying Eq. 24 by $[\mathbf{P}^{(s+m-1)}]^{-1}$ from the right, we get

$$\mathbf{I} = \mathbf{H}^{-1}[\mathbf{P}^{(s+m-1)}]^{-1} - \mathbf{P}_{rem}[\mathbf{P}^{(s+m-1)}]^{-1} \tag{25}$$

Multiplying Eq. 25 by $\mathbf{H}$ from the left, we get

$$\begin{aligned}\mathbf{H} &= [\mathbf{P}^{(s+m-1)}]^{-1} - \mathbf{H}\mathbf{P}_{rem}[\mathbf{P}^{(s+m-1)}]^{-1} \\ &= (\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})[\mathbf{P}^{(s+m-1)}]^{-1}\end{aligned} \tag{26}$$

Multiplying Eq. 26 by $(\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}$ from the left, we get

$$[\mathbf{P}^{(s+m-1)}]^{-1} = (\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}\mathbf{H} \tag{27}$$

From Eqs. 23 and 27, we have

$$\begin{aligned}&[\mathbf{P}^{(s+m-1)}]^{-1} = \\ &(\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}(\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}[\mathbf{P}^{(s-1)}]^{-1}\end{aligned} \tag{28}$$

Hence,

$$\begin{aligned}&[\mathbf{P}^{(s+m-1)}]^{-1}\vec{b}^{(s-1)} = \\ &(\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}(\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}[\mathbf{P}^{(s-1)}]^{-1}\vec{b}^{(s-1)} \\ &= (\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}(\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}\vec{\beta}^{(s-1)}\end{aligned} \tag{29}$$

From Eqs. 17, 18, 27, and 29, we have

$$\begin{aligned}&\vec{\beta}^{(s+m-1)} = \\ &[\mathbf{P}^{(s+m-1)}]^{-1}\vec{b}^{(s-1)} + [\mathbf{P}^{(s+m-1)}]^{-1}(\vec{b}_{add} - \vec{b}_{rem}) \\ &= (\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}(\mathbf{I} + [\mathbf{P}^{(s-1)}]^{-1}\mathbf{P}_{add})^{-1}\vec{\beta}^{(s-1)} \\ &\quad + (\mathbf{I} - \mathbf{H}\mathbf{P}_{rem})^{-1}\mathbf{H}(\vec{b}_{add} - \vec{b}_{rem})\ \square\end{aligned}$$

*Proof of Theorem 4*:
The theorem holds based on Eqs. 28 and 23. □