Vol. 38 No. 12 Transactions of Information Processing Society of Japan Dec. 1997

Regular Paper

A Neural Networks Approach for Query Cost Evaluation

JIHAD BOULOS,† YANN VIEMONT†† and KINJI ONO†

This paper presents a new approach for query cost evaluation that may help or replace the known analytical approach. Our proposed approach is based on neural networks and the connectionist concept. A neural network is trained to learn the execution cost of the implementation algorithm(s) for a logical algebra operation (or query) with some predicates; after that, this network is used to estimate this operation (query) cost with other entries. The approach is based on a curve fitting like since neural networks have been proven to be "universal approximators." An additional advantage of this approach is its applicability to user defined methods where the user does not need to estimate the cost of his method since the system may apply this method several times, collects measurements, and captures its behavior with its curve fitting capacity.

1. Introduction

Cost models are still a problematic issue in database research because of their imprecision and the continuous introduction of new data types and processing in extended database system. Most cost models used by new complex optimizers remain primitive regarding the real execution complexity. This simplification may mislead the optimizer to choose a Query Evaluation Plan (QEP) that was estimated cheaper than others in a certain environment (platform, DBMS, data), but is more expensive in the real one. Moreover, in distributed multi- database systems, several proprietary systems must cooperate with a global optimizer. Often, these systems do not provide their internal cost models and hence a general one must be assumed for all of them.

In this paper we study the feasibility and effectiveness of replacing an analytical cost model for query cost evaluation by a set of neural networks. The approach builds upon replacing each analytical cost formula for estimating the execution cost of a query or an operation by a neural network. This network is trained to capture the operation execution cost in a certain environment by a set of inputs (operation cost influencing parameters) and an output (the execution time). This implies that a phase of input-output measurement followed by a training phase must proceed the exploitation phase.

Cost models are in general simple, do not evolve with the environment, and put several assumptions on influencing factors (e.g., cache hit ratio, size of sorting area, ...) on query execution costs at execution time. For example, most known query optimizers uses the $(n \times \log_2 n)$ number of I/O for a sort operation as its cost. The base 2 is usually used for the logarithm while in reality the correct base must be the number of available clusters (merging units.) This last parameter may radically change the sort behavior from an I/O-bound operation to a CPU-bound one, when there is a sufficient number of buffers. Reference 3) showed that, whenever sufficient disks are available, a sequential scan is first CPU-bound when the tuple sizes are small and only becomes I/O-bound when these sizes get large. Most analytical models take a constant size for evaluated tuples. Moreover, Ref. 3) also identified two situations in the execution of single operations that are sensible to the available buffer size: 1) choosing between a sequential scan and an index-scan with an unclustered index, and 2) choosing between a nested-loops with an index-scan over the inner relation and a hash-join.

Several research papers[1),6),7)] address the issue of query cost estimation for heterogeneous DBMSs. In all these papers, the authors use a calibration process for some predefined analytical models.

Reference 1) establishes a calibration method to extract for a predefined analytical cost model its parameter values that depend on the platform and the DBMS. A benchmark is executed on a system to measure its performance, and an equation system is then constructed and resolved. This process produces a calibrated cost

† National Center for Science Information Systems (NACSIS)
†† Laboratoire PRiSM, Univ. de Versailles-St-Quentin, France

model for a specific platform. References 6) and 7) use the same process to approximate the execution cost of a query in a heterogeneous environment by executing some predefined queries on the target DBMS with different result sizes. Contrary to Ref. 1) where basic operations are modeled, in Refs. 6) and 7) global queries are modeled.

The paper is organized as follows: Section 2 establishes a formal equivalence between query cost executions and analytical functions and Section 3 presents a background on neural networks for function approximation. In Section 4, we discuss ways of constructing and training neural networks for queries and operations cost prediction and Section 5 presents issues in using neural networks in evaluating some of the different execution algorithms of database operations. Section 6 presents the results of several experiments to test the efficiency of neural networks in query cost evaluation and in Section 7 experiment on cost learning and prediction of user defined methods is reported. A conclusion and some future directions are stated in Section 8.

2. Query Cost Functions

Equivalence between query costs and cost estimation functions must be established in a first step to formalize the applicability of neural networks in query cost estimation. References 1), 6) and 7) also use similar equivalence models.

For any query a certain number n of variables govern its execution cost (this is also true for basic physical operations). We denote these variable by $x_1, \ldots, x_n$. Any variable, however, need not to be independent from the others or strictly linear. It is allowed, for examples, that $x_i = x_j \times x_k$ or $x_i = x_j \times f(x_k)$. This flexibility allows a variable to capture the cost of nonlinear operations such as a sort. The query execution cost C is dependent on each of these variables and tends to vary in a continuous manner with their variations. We can hence say

$$C = a_0 + a_1 x_1 + \ldots + a_n x_n = a_0 + \sum_{i=1}^{n} a_i x_i$$

where $a_0, \ldots, a_n$ are *cost coefficients* representing the influence of each variable x_i on C. To get the equivalence between the query cost and this linear combination of cost factors x_i we must find $a_0, \ldots, a_n$. One way to get these is to execute the query m times while varying $x_1, \ldots, x_n$ to get an equation system of the form

$$C_j = a_0 + a_1 x_{1,j} + \ldots + a_n x_{n,j} = a_0 + \sum_{i=1}^{n} a_i x_{i,j} \qquad (j = 1, \ldots, m).$$

Solving this system by the least squares method yields the coefficients $\hat{a}_0, \ldots, \hat{a}_n$ that give an estimated cost E_i for each measured cost C_i. These coefficients are an approximation of $a_0, \ldots, a_n$ and minimize

$$LSE = \sum_{j=1}^{m} [C_j - (a_0 + a_1 x_{1,j} + a_2 x_{2,j} + \cdots + a_n x_{n,j})]^2.$$

Equivalence between any query cost function and a continuous cost function is hence established.

3. Neural Network Approximation Capabilities

A neural network is characterized by its architecture. The elements of a network are computing units and their interconnecting edges. Formally, Ref. 4) defined "*a neural network architecture is a tuple* (I, N, O, E) *consisting of a set* I *of input sites, a set* N *of computing units, a set* O *of output sites and a set* E *of weighted directed edges. A directed edge is a tuple* (u, v, w) *whereby* $u \in I \cup N$, $v \in N \cup O$ *and* $w \in R$."

For our problem of query cost evaluation, we focus our attention on feed-forward neural networks (**Fig. 1**) because of their proven function approximation capabilities.

3.1 Feed-Forward Networks

A feed-forward neural network is a computational graph whose nodes are computing units and whose directed edges transmit numerical information from node to node. Each computing unit is capable of evaluating a single primitive function—the activation function—of its inputs and compares it with the unit-computed

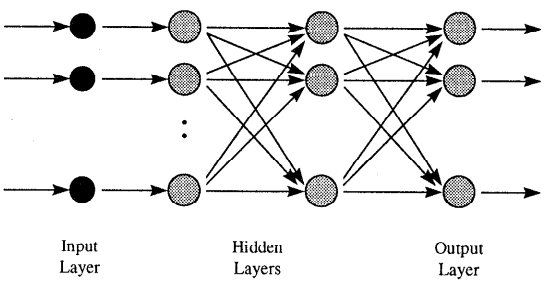


Fig. 1 A feed-forward neural network.

threshold. Hence, the network represents a chain of function compositions that transform an input to an output (called a pattern). The network can be understood as a particular implementation of a composite function from input to output space, which is called *network function*. The most known and used learning algorithm in feed-forward neural networks for function approximation is the back-propagation algorithm. With this learning algorithm, the most popular activation function is the *sigmoid*. Given random weights $w_1, \ldots, w_n$ and a bias $-\theta$, a sigmoidal unit computes for the inputs $x_1, \ldots, x_n$ the output

$$\frac{1}{1 + \exp\left(\sum_{i=1}^{n} w_i x_i - \theta\right)}.$$

The learning problem of the back-propagation algorithm consists of finding the optimal combination of weights $w_1, \ldots, w_n$ so that the network function φ approximates a given function f as closely as possible. However, and unlike statistical regression methods, we do not need to know the shape of the f function *explicitly* but implicitly through some examples.

Consider a feed-forward network with n input and k output units. It is trained with a set $\{(x_1, t_1), \ldots, (x_m, t_m)\}$ consisting of m orders pairs of n- and k- dimensional vectors (the input-output patterns). Let the primitive functions at each node of the network be continuous and differentiable (e.g., the sigmoid). The weights of the edges are real numbers selected at random. When the input x_i from the training set is presented to the network, it produces an output o_i different in general from the target t_i. What we want is to make o_i and t_i identical for $i = 1, \ldots, m$, by using a learning algorithm. Explicitly, we want to minimize the error function of the network, defined as

$$Er = \frac{1}{2} \sum_{i=1}^{m} \|o_i - t_i\|^2.$$

After minimizing this function for the training set, new unknown input patterns are presented to the network and we expect it to *interpolate*.

Hornik, Stinchcombe, and White[2] were the first to prove in the following theorem the universal approximation property of feed-forward neural networks.

HSW Theorem: *standard multi-layer feed-forward neural networks with as few as a single hidden layer can approximate any continuous function uniformly on any compact set and any measurable function arbitrarily well in the corresponding metric, regardless of the squashing function (continuous or not), regardless of the dimension of the input space, and regardless of the (finite) measure used.*

Hence, feed-forward networks are theoretically capable of approximating any multidimensional function to any desired degree of accuracy-provided sufficiently hidden units are available. As a consequence, HSW theorem establishes that "*failures in applications can be attributed to inadequate learning, inadequate numbers of hidden units, or the presence of a stochastic rather than a deterministic relation between input and target.*"

The formal equivalence between query costs and analytical cost estimation functions and the formal proof in the HSW theorem of the existence of at least one neural network for approximating any analytical continuous function provide a proof on the capabilities of feed-forward neural networks in approximating query cost functions.

3.2 Accuracy Measure

Several methods have been proposed in the neural network field of research for measuring the goodness of a neural network approximating a function. All of these methods require a network to be trained on a subset of the measured points and validated on the rest or the whole set. We have applied this constraint in all the experiments reported in this paper. However, we use here general statistical analysis formulas adopted from Ref. 7) because of their applicability in our problem of continuous function approximation.

The standard error of estimation is defined as

$$SE = \sqrt{\frac{\sum_{i=1}^{m} (C_i - E_i)^2}{[m - (n+1)]}}.$$

This standard error is an indication of the accuracy of the estimation. The smaller SE is, the better the estimation is.

A second descriptive measure used to judge the goodness of a trained network is the coefficient of multiple determination R^2, defined as

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (C_i - E_i)^2}{\sum_{i=1}^{m} \left[C_i - \left(\sum_{j=1}^{m} C_j\right) \Big/ m\right]^2}.$$

$R^2 (\in [0, 1])$ is the proportion of variability in the response variable C explained by the base variables x's. The larger R^2 is, the better the estimation is. We also use the average absolute error and the average relative error in each

evaluation of the accuracy of the estimation. This is because these two evaluation measures are well known and used. The absolute error is computed as

$$\sum_{i=1}^{m} |C_i - E_i|$$

and the relative error is computed as

$$\frac{\sum_{i=1}^{m} |C_i - E_i|}{\sum_{i=1}^{m} C_i}.$$

4. Evaluation Approaches

The query cost evaluation process may be applied on complete queries or on basic operations within each query. The advantage of having a single large network structure for each type of queries is its bounded global error rate while its inconvenience is the large structure of the networks and the null inputs that these networks may have when a simple query is injected into one of them. On the other hand, we can construct a small network for each atomic operation in a QEP and connect these small networks in a large one according to each QEP. The advantage of this choice is its adaptability to the QEP structure of any query while its inconvenience is the propagation of the error rate of the different networks.

4.1 Modeling Queries

In the first approach of query processing cost evaluation by neural networks, whole queries may be measured, learned, and evaluated. In this context a classification of query types must be processed to identify the fittest network to a query, and then the network is used to estimate the cost of the query. A finite set of query types must be identified and modeled.

The types of queries may be classified according to several criteria, e.g., the number of joins in the query, the number of predicates in the query, the complexity of the functions that must be applied in the query. Since neural networks have a powerful capability of classification[4], they may classify in a first phase an evaluated query in a certain type (**Fig. 2**), and then evaluate the cost of the query according to the neural network that captures the execution model of the type (**Fig. 3**).

This approach—adopted by Refs. 6) and 7)—is relatively simple but the identification of query types and the classification of queries complicate the evaluation process.

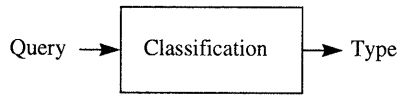


Fig. 2 A first phase of query classification.

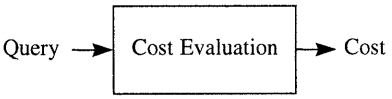


Fig. 3 A second phase of query cost evaluation.

4.2 Modeling Operators

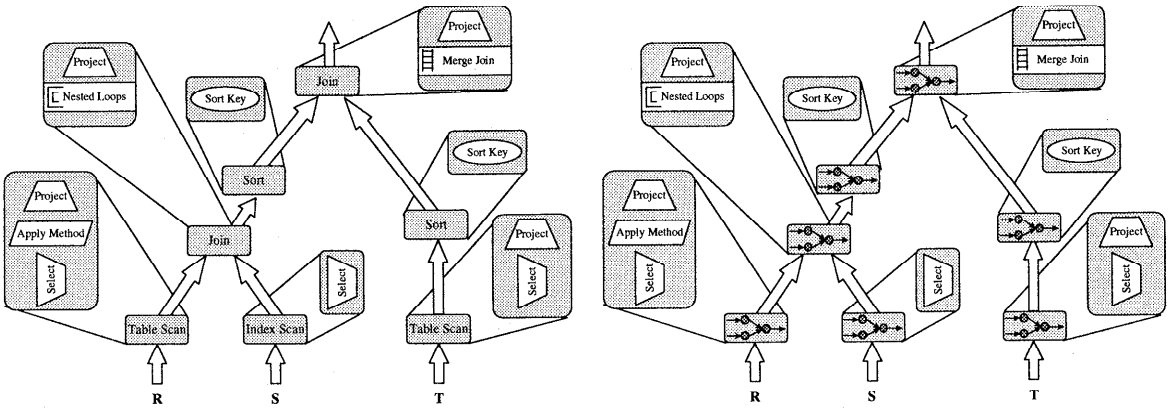
In a different approach of query processing cost evaluation by neural networks, basic operations in QEPs may be measured, modeled, and evaluated. However, for performance evaluation of queries, the constraint in applying this approach is that the evaluator must have access to the QEPs generated by the DBMS. In this case, a query execution tree is generated by the DBMS without a real execution of the query; this query tree is then passed to the evaluator and a separated neural network according to the node type evaluates the cost of each node in the tree. The whole cost of the query is computed from the query tree node costs.

The number of networks in this set is known in advance since a network captures the cost of a query node type, which is an element in a finite set of physical operations. Statistics on a node in the query tree, and eventually the result size of a preceding node, are the main inputs to the neural network modeling its execution cost. An illustration of the mapping of a query tree to a tree of neural networks is given in **Fig. 4**.

This approach of query processing cost evaluation by neural networks is more complicated than the first one, but its applicability and integration in an optimizer or a performance evaluation tool seem to be more realistic. It has been adopted by Ref. 1).

5. Neural Network Issues

Several problems arise when applying the approximation capabilities of neural networks to query cost evaluation. The kind and structure of the networks that mostly adapt to the problem must be identified and constructed. Influencing and irrelevant environment parameters for a platform, a DBMS, and an application also must be identified and separated, and measurements must be done to obtain a significant amount of data for training and validating the



(a) A QEP with its statistics. (b) The equivalent network of neural networks.

Fig. 4 The mapping between a QEP and a network of neural networks.

networks. As a general rule of thumb, small values for the training and validating patterns are more convenient for neural networks. Large values of some input parameters may be "cooked" in order to bring them to more convenient ones. An example of such parameters is the size of a relation. In this case, it is better to inject in the neural network the logarithmic value of the size (e.g., $\log_{10}(\text{size})$).

Also, an applied rule for testing a neural network generalization is to train the network on a part of the I/O data and to test its performance on a second part. We applied this rule in all the experiments reported in this paper: 20% of the measurements were used as training data, and the whole measurements were used as validation data.

5.1 Network Structure

The theorem of Ref. 2) (on which all this work is founded) guarantees the existence of (at least) a neural network that approximates a specific function. However, it did not give a method to generate the network; a trail-and-error process for the network structure must be undertaken. During the learning phase and in the cases where a network reaches a stable state that does not satisfy the user (i.e., stuck in a local minimum), the network is then reinitialized with other random values and the learning phase is repeated. If after several trials, the network did not converge to an acceptable rate, a different structure of the network must be regenerated and the process of learning is again reinitialized.

When should the training phase stop? The learning phase makes as many cycles on the learning examples as the user is unsatisfied with the error rate showed by the network. When the user decides that the error rate is sufficiently small, he stops the learning phase and tests the network with other input data.

5.2 Input Parameters

The choice of the input statistical parameters to a neural network must characterize a database application and at the same time be relevant to the network. In addition, there is a part of these parameters that depends on whether the general approach (modeling queries) or the building block one (modeling operators) is adopted and on the data model and its optimization strategy. However, some parameters are the same for both choices.

Some of the common inputs that are the most representative for most queries are stated here. These may be: the value of a predicate constant, the size of the input, the size of the output (selectivity), indexes for each relation, clustered indexes or not, average size of a tuple, number of attributes in a tuple, minimum and maximum values for each attribute, kind of the predicate(s) $(=, <, \leq, >, \geq)$, symbolic or numerical predicate, CPU power, disk access time, network bandwidth.

SQL queries must be executed with different shapes, predicates, predicate constants, and database sizes. For example, the SQL query that must be executed to extract the cost of a table scan is:

Select Count(*) **From** R;

The "Count" is used to avoid the cost of result transfer between the DBMS working space and the client working space and it was used for the same reason in all the other queries.

6. Experimental Results

Several experiments were repeated with different versions of a commercial DBMS and on several platforms. All the results were consistent and reproducible. We report here experiments on a Sun-Sparc1000 machine. Selection queries from the AS^{3}AP[5] and the DKS[1] benchmarks were executed several times on several sizes of the databases and measurements were collected. The tests were made in two ways: measurements and estimations were made on global relational queries, and then measurements and estimations were made on basic algebraic operations. For all the experiments, 20 measurement points were taken for each experiment where 20% were used to train the neural network and the whole 20 points were used to test it. We report the performance of several neural networks in query processing cost estimation for one global joining query and several other basic algebraic operations.

We have also calibrated general analytical cost formulas for some basic operations to compare their output performance against that of neural networks. The accuracy of the results of each method is given for each experiment with the standard error of estimation SE, the coefficient of multiple determination R^2, the average absolute error, and the average relative error.

6.1 Global Approach

In this test the global execution cost of a query joining two relations is estimated. Two relations of 100,000 tuples each are joined with the sort-merge-join algorithm. A predicate is applied on one of the two relations in order to change its result size and hence the merge size. The query has the following shape:

Select Count(*)
From $R1, R2$
Where $R1.A1 = R2.A2$
and $R2.A2 = cst$

The optimizer choose the following QEP for this query:

Select (Count)	1
Merge-join	Result-size
Sort(join)	Result-size
Table-Access-Pred($R2$)	100,000
Sort(join)	100,000
Table-Access-Full($R1$)	100,000

The first relation is full scanned and the 100,000 tuples are sorted, while the second relation is full scanned and the predicate is applied and then the remaining tuples are sorted. Since a uniform distribution was chosen for both attributes in the two relations, the result size from merging the two relations is the same as the size of the second relation after applying the predicate. The analytical cost formula that was calibrated for this query was computed as follows:

$$\begin{aligned}&\text{Cost(Sort-Merge-Join)}\\&= (2 \times 100000 \times \mathit{Access\_Tuple})\\&\quad + (100000 \times \log_2(100000)\\&\quad\quad /\mathit{machine\_dependent\_constant})\\&\quad + (\text{Result\_Size} \times \log_2(\text{Result\_Size})\\&\quad\quad /\mathit{machine\_dependent\_constant})\\&\quad + \text{Result\_Size} \times \mathit{Merge\_Join}.\end{aligned}$$

The only input parameter to the network was the predicate constant value—since all other parameters were unchanged—and the output was the elapsed cost time. The most appropriate structure of the neural network was 1-3-2-1; it converged after 2,000 cycles in about 3 seconds.

Table 1 A comparison of error rates on a sort-merge join query.

	SE	R^2	*Avg. Abs. Err.*	*Avg. Rel. Err.*
NN Estimates	0.34	0.99	0.21	0.99%
Analytical Estimates	0.43	0.98	0.34	1.5%

Table 1 compares the four mentioned error rates for both the neural network and the analytical formula. The reader must remember that the higher R^2 is, the better the estimates are. **Figure 5** illustrates the curves for the query real execution time, the neural network estimates, and the analytical formula estimates. From both the comparison table and the figures, it is shown that the neural network estimates are better than those of the analytical formula are.

6.2 Building Blocks

With this approach of cost estimation where each physical operation is isolated and estimated alone, the efficiency of neural networks with different input parameters for several operations is tested. The reported tests here are for the scan, the sort, and the merge-join operations.

6.2.1 Scan

The first test consisted of scanning a relation with several sizes. The size of the relation was varied between 5,000 and 100,000 tuples, and the same query was repeated on all sizes. A neural network was generated (with a structure of 1-3-2-1) and trained on a subset (20%) of the measurement data, and converged after few

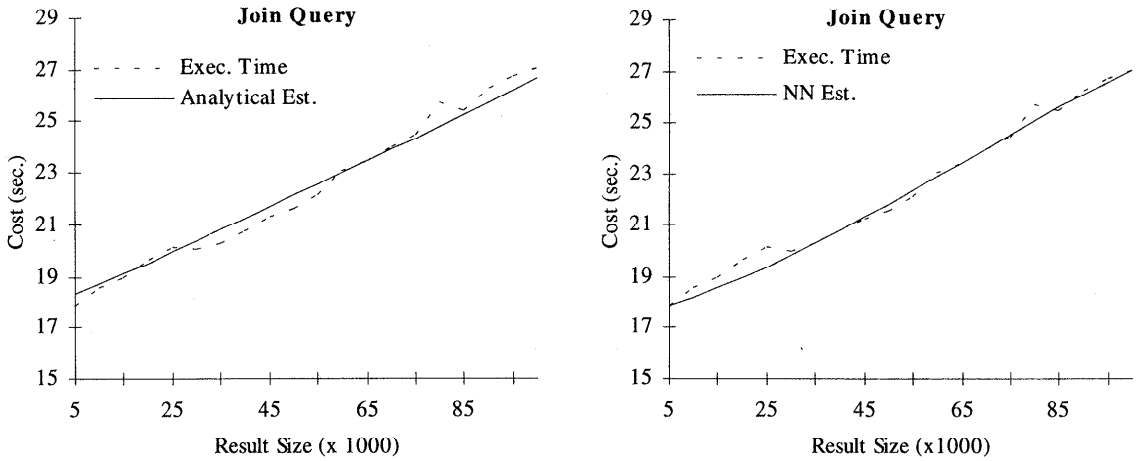


Fig. 5 Performance of a neural network against an analytical formula for the sort-merge query.

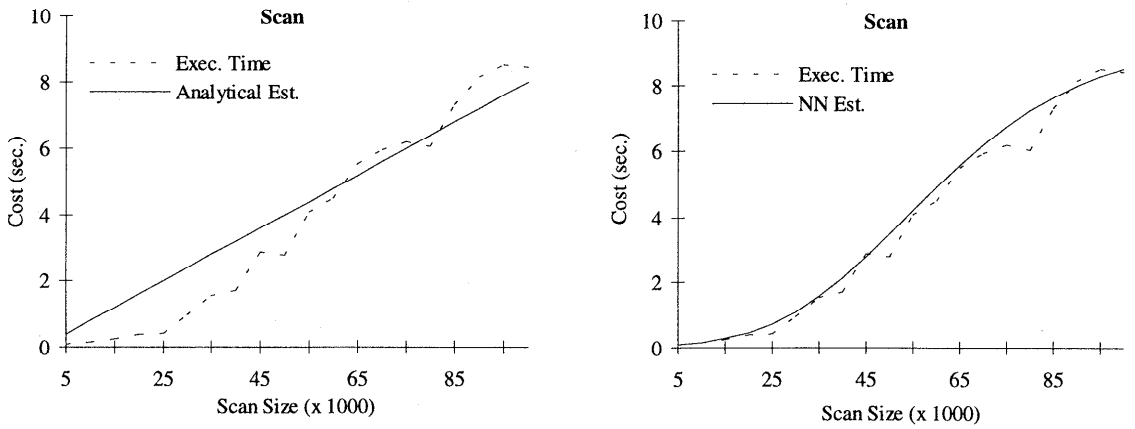


Fig. 6 Performance of a neural network against an analytical formula for the scan operation.

Table 2 A comparison of error rates on a elapsed cost of a scan.

	SE	R^2	*Avg. Abs. Err.*	*Avg. Rel. Err.*
NN Estimates	0.41	0.98	0.26	11.7%
Analytical Estimates	0.95	0.91	0.77	113.4%

hundred cycles. The measured cost of this test was the elapse time of the scan, i.e., the CPU and the I/O cost. **Table 2** compares the error rates of the neural network against those of the following calibrated analytical formula:

Cost(Scan)
= Scanned_tuples × *Access_tuple*.

Figure 6 illustrates the estimates of both the neural network and the analytical formula against the real execution time of the scan. The neural network outperformed the analytical formula by a considerable margin because of the cache effect on the small scan sizes and where the analytical formula could not capture it.

6.2.2 Sort

The second test consisted of sorting a relation with several sizes. The size of the sort was also varied between 5,000 and 100,000 tuples. A neural network was generated (with a structure of 1-3-2-1) and trained with a subset of the measurement data. The measured cost of this test was the elapse time of the sort, i.e., the CPU and the I/O cost. **Table 3** and **Fig. 7** compare the performance of the output of the neural network against that of the following calibrated cost formula:

Cost(sort)
= Sort_Size × $\log_2$(Sort_Size)
/(*machine_dependent_constant*).

Here, the neural network outperformed the analytical formula by a small margin.

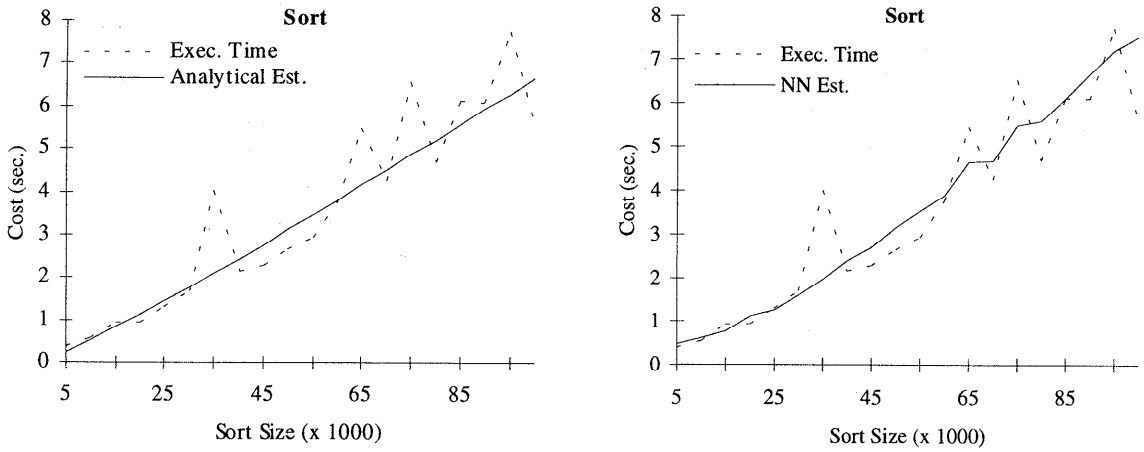


Fig. 7 Performance of a neural network against an analytical formula for the sort operation.

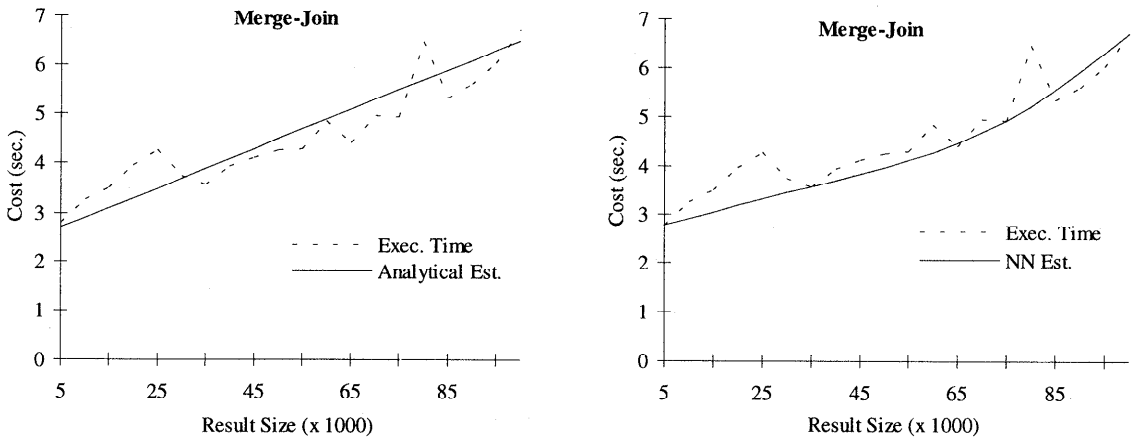


Fig. 8 Performance of a neural network against an analytical formula for the merge-join operation.

Table 3 A comparison of error rates on a sort.

	SE	R^2	*Avg. Abs. Err.*	*Avg. Rel. Err.*
NN Estimates	0.81	0.97	0.53	15.4%
Analytical Estimates	0.84	0.96	0.57	16.11%

Table 4 A comparison of error rates on the merge-join.

	SE	R^2	*Avg. Abs. Err.*	*Avg. Rel. Err.*
NN Estimates	0.49	0.8	0.34	7.44%
Analytical Estimates	0.47	0.82	0.38	8.44%

6.2.3 Merge-Join

The third test consisted of merging two relations with several result sizes. The result size of the merge was varied between 5,000 and 100,000 tuples. A neural network was generated (with a structure of 1-3-2-1) and trained with a subset of the measurement data. The measured cost of this test was the elapse time of the merge. **Table 4** and **Fig. 8** compare the performance of the neural network output against that of the following calibrated cost formula:

$$\text{Cost(merge-join)} = \text{Result\_Size} \times \mathit{Merge\_Join}.$$

Here, both the neural network and the analytical formula performed slightly the same.

6.3 Global against Building Blocks

The performance of the global approach of query cost estimation against that of the building block approach is an interesting issue. However, it would be unfair to directly compare the estimation accuracy of the two approaches. Summing the estimations of nodes in the building block approach overestimates a query cost

Table 5 A comparison of error rates on the merge-join.

	SE	R^2	*Avg. Abs. Err.*	*Avg. Rel. Err.*
Global	0.34	0.99	0.21	0.99%
Building Blocks	0.96	0.89	0.71	3.13%

because this approach does not take into account the pipelining and the overlapping of CPU and I/O of different operations. Measuring and calibrating these two factors, a more reliable comparison can be made. **Table 5** presents for the two approaches the four adopted error measurements for the scan-merge-join query of Section 6.1.

The global approach of query cost estimation outperforms that of the building blocks approach. However, it is up to the designer to choose one or the other of the two approaches depending on his needs and the applicability of the approach. The classification step in the global approach is still a limiting factor for its applicability in all situations.

7. Estimating Method Costs

An additional advantage of neural networks in query cost evaluation is their applicability to user defined methods. However, the condition for this applicability is that the method cost varies in a continuous manner with all its input parameters. Whenever this is the case, an equivalence model between the method input parameters and an analytical function is established and the applicability of neural networks for cost estimation of this method is proved in the same manner as in Section 2.

We defined and tested several methods with different execution costs and influencing input parameters. We report here on a method with two inputs. Increasing the number of input variables is straightforward.

The first input variable of the tested method had an exponential influencing cost on the method while the second input variable had a logarithmic influencing cost. The method was executed with 10 values for each variable, which makes 100 executions. We trained a network with 20% of the sample points and tested it with all points. The experimented network had a structure of 2-4-3-1. About 3000 cycles over the learning set was sufficient for the network to converge, which took about 10 seconds. The performance of the network went beyond our hopes and the average relative error between

Table 6 Performance of a neural network in estimating the cost of a complex user defined method.

	SE	R^2	*Avg. Abs. Err.*	*Avg. Rel. Err.*
NN Estimates	0.77	0.99	0.58	0.39%

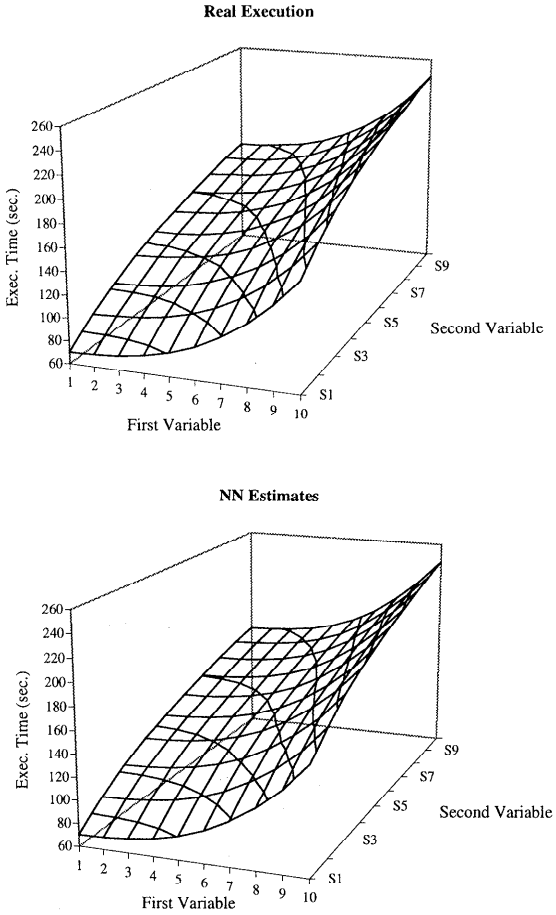


Fig. 9 Comparison between the real execution cost and the NN estimated cost of a complex method.

the estimated cost and the real execution cost for the 100 execution points was bellow 1% (**Table 6**). We can see from **Fig. 9** the shapes of both the real executions cost and the neural network estimated cost of this method.

8. Conclusion

We have proposed in this paper a new approach for query processing cost estimation. The approach is based on the capabilities of neural networks to approximate any measurable function in a curve fitting like manner. A set of neural networks is trained with measure-

ments from a working database environment in a first phase, and it produces cost estimates for evaluated queries in a second phase.

Limitations of analytical cost formulas and new requirements of emerging database technology motivated our work in searching for a new method to predict the performance of query processing algorithms. The neural network approach to overcome these limitations and meet the new requirements was decomposed, analyzed, and successfully experimented.

A discussion on global or building block ways of evaluating the cost of a query by neural networks was exposed and analyzed. We have the feeling that the building block approach where a tree of neural networks matches the tree of nodes in a QEP may outperform the global one in its effectiveness in a real working environment. We have, however, reported experimental results for both approaches and comparisons against calibrated analytical cost formulas proved the effectiveness of both of them. In addition, the effectiveness of neural network estimations of user defined method costs was also reported with an example.

More research work on feeding additional input parameters of a database environment into larger neural networks must be done. We did not until now integrate any platform dependent parameter in our tests; we intend investigating this influencing aspect on performance by varying the CPU power, the disk throughput, and the network bandwidth in a distributed environment.

References

1) Du, W., Krishnamurthy, R. and Shan, M.C.: Query Optimization in Heterogeneous DBMS, *18th VLDB Conference, Vancouver, British Columbia* (1992).
2) Hornik, K., Stinchcombe, M. and White, H.: Multilayer Feed-forward Networks are Universal Approximators, *Neural Networks*, Vol.2, pp.359–366 (1989).
3) Hellerstein, J. and Stonebraker, M.: Predicate Migration: Optimizing Queries with Expensive Predicates, *ACM SIGMOD, Washington DC* (May 1993).
4) Rojas, R.: *Neural Networks: A Systematic Introduction*, Springer-Verlag, Heidelberg (1996).
5) Turbyfill, C., Orji, C. and Bitton, D.: AS^{3}AP – An ANSI Sequel Standard Scalable and Portable Benchmark for Relational Database Systems, *Performance Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, San Mateo, CA (1991).
6) Yao, Z., Chen, C. and Roussopoulos, N.: Adaptive Cost Estimation for Client-Server based Heterogeneous Database Systems, University of Maryland Report, CS- TR-3648 (May 1996).
7) Zhu, Q. and Larson, P.: Building Regression Cost Models for Multidatabase Systems, *PDIS'96, Miami, Florida* (1996).

(Received April 3, 1997)
(Accepted September 10, 1997)

Jihad Boulos is a visisting researcher at NACSIS. He got a B.S. in Computer Science from the American-Lebanese University in Lebanon in 1991, an M.S. and a Ph.D. in Computer Science from the University of Paris-VI in 1993 and 1996 respectively. He is currently on a post-doctoral position at NACSIS. His main research interests concern database performance evaluation and assessment, continuous media servers and hierarchical storage systems.

Yann Viemont is an associate professor at the University of Versailles in Paris, France. He got an M.S. and Ph.D. in Computer Science from the University of Paris-VI. He worked at Digital Equipment in the United States and as an assistant professor at the University of Connecticut. His main research interests focuses on geographical database systems and database performance evaluation.

Kinji Ono is a professor and director of R&D at NACSIS. He received the B.S. in physics from the University of Tokyo in 1962, M.S. in Electrical Engineering from Stanford University in 1972, Dr. of Eng. from the University of Tokyo in 1983. He joined KDD in 1962 where he engaged in the research on satellite communications, computer communications. After serving as the Director of Research Laboratories ,he was assigned the Professor of NACSIS in 1993. His current research interests include high speed networking and distributed multimedia. He was awarded the Prize of the Minister of the STA, the Achievement Prize of IEICE. He is a Governor of ICCC, member of IPSJ, IEICE and Fellow of IEEE.