1988

IMPROVEMENTS ON THE BEST CASE PERFORMANCE OF A DYNAMIC PROGRAMMING ALGORITHM

Goker Gursel
San Diego State University

IMPROVEMENTS ON THE BEST CASE PERFORMANCE OF A DYNAMIC PROGRAMMING ALGORITHM

Goker Gursel
Department of Mathematical Sciences
San Diego State University

ABSTRACT

A dynamic programming algorithm that was initially designed to solve simple queries in chain networks (Scheuermann and Gursel 1984) and later extended to solve similar queries in ring networks (Gursel 1986) is introduced. The algorithm, with the objective of minimizing the total volume of data transmission, can be incorporated into heuristic approaches to solve general queries in networks not necessarily completely connected. Given the sizes of the referenced relations under different reduction states as input, the solution is expressed as a sequence of semi-join operations. The time and space complexity of the algorithm is known to be $O(n^3)$ and $O(n^2)$ respectively. In this paper the time complexity of the algorithm is improved to $O(n)$ under favorable input conditions.

1. INTRODUCTION

An important problem in distributed database systems is that of processing queries that reference geographically dispersed data. The network transmission delays are in general several orders of magnitude slower than the local processing or disk transfer speeds. Minimizing the total data transmission volume among network nodes while ignoring local processing costs has often been the optimization objective in processing such queries.

The problem of finding the optimal data transmission schedule for processing a general query in an arbitrary network topology is known to be NP-hard (Hevner 1979). Hence, the algorithms developed for distributed query processing either find optimal solutions for special restricted problems or are based on heuristic models. With the exception of a few (Kerschberg, Ting and Yao 1982; Sugihara et al. 1984), most algorithms assume the existence of a completely connected network. A special subclass of queries, simple queries, was introduced by Apers, Hevner and Yao (1983). Optimal algorithms for processing simple queries in completely connected networks were developed under both the total time and the response time minimization objectives. These algorithms were then used as tools in deriving heuristics solutions for general queries.

Scheuermann and Gursel (1984) developed a dynamic programming algorithm was developed for the optimal processing of simple queries in chain networks. The algorithm was later extended for the optimal solution of the same problem in ring networks (Gursel 1986). In both cases, the relational database model was considered and minimization of the total data transmission volume has been the optimization objective. The input to the algorithm consisted of the estimates on the sizes of the referenced and intermediate relations. In the section 2, the algorithm as it is defined for simple queries in chain networks is introduced. It will then be modified it in two steps to improve its best case performance. The improvements will be applicable to its extension over ring networks.

2. BACKGROUND AND DEFINITIONS

The topology of a chain network can be represented by a communication graph G(N,E) where,

N = the set of network nodes on which we impose the order $N_1,N_2,...,N_n$ (n is the number of nodes),

E = the set of communication links among nodes
= $\{(N_i,N_{i+1}) \mid N_i,N_{i+1} \in N, 1 \leq i < n\}$

Note that the edges represent bi-directional links and each node, except N_1 and N_n, is connected to two other nodes. The communication graph of a ring network is similar to that of a chain network except that there is an additional edge between N_1 and N_n.

We assume a *nonredundant materialization* of the database. That is, a distinct relation, R_i, is assumed to reside at each node, N_i.

Simple queries are defined as queries for which, after initial local processing, each referenced relation contains only the common joining attribute denoted by A which is also the only output of the query. A simple query is in general described by an equi-join qualification clause $q_s = \bigwedge_{i=2,n}(R_1 . A = R_i . A)$ and a result node. In the following discussions without loss of generality N_1 is considered to be the result node.

Considering only the data transmission costs, the optimal strategy for processing a simple query consists of a sequence of semi-join operations known as a semi-join reduction program (SJRP). The semi-join of relations R_i and R_j over the common attribute A is defined as:

$$R_i \,.\, A \propto R_j \,.\, A = \{t_i | t_i \in R_i,\ t_i \in R_j \text{ and } t_i \,.\, A = t_j \,.\, A\}$$

As a result, tuples in R_i that do not join with those reduced reduced in R_j are eliminated. Hence R_i is said to be *reduced*. Note that in the case of simple queries a semi-join is equivalent to the intersection of the operant relations. In order to keep track of the reduction state of each relation we associate as in with each relation R_i a time varying *join set* J_i which is defined as:

Initialization: $J_i \leftarrow \{i\}$
Update: $J_i \leftarrow J_i \cup J_k$ if R_i is reduced by R_k.

The set of indices in J_i represent the set of relations (including R_i) that have transitively or non-transitively reduced R_i. The notation $R_i\langle J_i\rangle$ indicates the reduction state of R_i at a particular time. Due to the special arrangement of the nodes in a chain network, it is possible to express the join set J_i as a range of indices. Let g and h be two indices such that $1 \le g \le h \le n$We then define the *join range* notation $J_i = \langle g,h\rangle$ to denote the set $\{g,g+1,...,h\}$. Note that $J_i = \langle 1,n\rangle$ represents the universal set of indices, namely $\{1,2,...,n\}$.

A relation R_i is said to be *fully reduced* when its join set $J_i = \langle 1,n\rangle$. A simple query in a chain network with N_1 as the result node is said to be *solved* when R_1 is fully reduced. As is the case for chain queries (Chiu 1980) and simple queries in ring networks, two types of semi-join operations are defined for simple queries in chain networks:

1) for $i<n$, $y_i = R_i \,.\, A \propto R_{i+1} \,.\, A$
(y_i reduces R_i from its right, hence $J_i \leftarrow J_i \cup J_{i+1}$)

2) for $i>1$, $x_i = R_i \,.\, A \propto R_{i-1} \,.\, A$
(x_i reduces R_i from its left, hence $J_i \leftarrow J_i \cup J_{i-1}$)

Under the total time minimization objective, the cost of a SJRP is defined to be the cost sum of the constituent semi-join operations. The cost of a semi-join is in general expressed as a linear function of the amount of data transmitted. Let a_{ij} and b_{ij} denote the cost of transmitting a unit of data and the transmission start-up cost respectively when the link between nodes N_i and N_j is used. Hence the cost of the semi-join $R_i \,.\, A \propto R_j \,.\, A$ is given as:

$$a_{ij} * s(R_j\langle J_j\rangle \,.\, A) + b_{ij}$$

where $s(R_j\langle J_j\rangle \,.\, A)$ is the size of the transmitted joining attribute $R_j \,.\, A$ which depends on the current join-set J_j.

These results hold true for arbitrary values of a_{ij} and b_{ij}. However, for notational simplicity, we restrict $a_{ij} = 1$ and $b_{ij} = 0$ for all i and j. We also note that, with respect to simple queries, the join of a set of relations is the same regardless of the join site. That is, $s(R_i\langle J_i\rangle) = s(R_k\langle J_i\rangle) = s(J_i)$. In other words, given the join set J_i the resulting relation need not be identified. It always corresponds to the intersection $\Omega_{k \in J_i} R_k$.

The sizes of all relations under all possible reduction states, namely $s(J_i)$ for all J_i, are assumed to be given as input to our algorithm. Note that there are a total of n^2-n+1 possible distinct join-sets J_i. Hence, $cost(x_i) = s(J_{i-1})$ and $cost(y_i) = s(J_{i+1})$.

Let X_c^d and Y_c^d denote sequences of x-type and y-type semi-join operations respectively that transmit data from some source node N_c to some destination node N_d. That is,

X_c^d: $x_{c+1}x_{c+2}...x_d$ (right bound transmission)

Y_c^d: $y_{c+1}y_{c+2}...y_d$ (left bound transmission)

The cost of X_c^d or Y_c^d can be expressed as the sum of the costs of the constituent semi-joins in sequence.

Let $S\langle j,i\rangle$ denote the optimal strategy (SJRP) under and only total time minimization objective that reduces R_i *and only* R_i in the join range $J_i = \langle j,i\rangle$ for some $1 \le j \le i$.Let $C\langle j,i\rangle$ denote the cost of $S\langle j,i\rangle$. We note the following properties of $S\langle j,i\rangle$:

P1 - For $j = i$, $S\langle j,i\rangle$ reduces to null strategy. Hence $C\langle j,i\rangle = 0$.

P2 - $S\langle j,i\rangle$ does not contain y_{i-1}. Suppose it did: then R_{i-1} would be reduced in the join range $J_{i-1} = \langle j,i\rangle$ before R_i, which conflicts with the definition of $S\langle j,i\rangle$.

The next property follows the theorem proven by Gursel and Scheuermann (1984).

> An optimal strategy for processing a simple query in a chain network, given that the resulting node coincides with one end of the chain network, can always be obtained as a strictly serial SJRP under the total time minimization objective.

P3 - $S\langle j,i\rangle$ is a strictly serial SJRP.

Hence for $j \neq i$,$S\langle j,i\rangle$ fits into the least costly of the following two forms:

Form a) $S<j,i-1>\ x_i$ (see Figure 1a)

where $S<j,i-1>$ reduces R_{j-1} in the join range $<j,i-1>$ and x_i transmits the result to N_i, thereby reducing R_i in the join range $<j,i>$.

Form b) $S<t,i-1>\ Y^j_{i-1} X^i_j$ (see Figure 1b)

where $S<t,i-1>$ reduces R_{i-1} in the join range $<t,i-1>$ for some $j<t<i$,

Y^j_{i-1} transmits the reduced R_{i-1} to N_j, thereby reducing R_j in the join range $<j,i-1>$,

X^i_j transmits the result to N_i, thereby reducing R_i in the join range $<j,i>$.

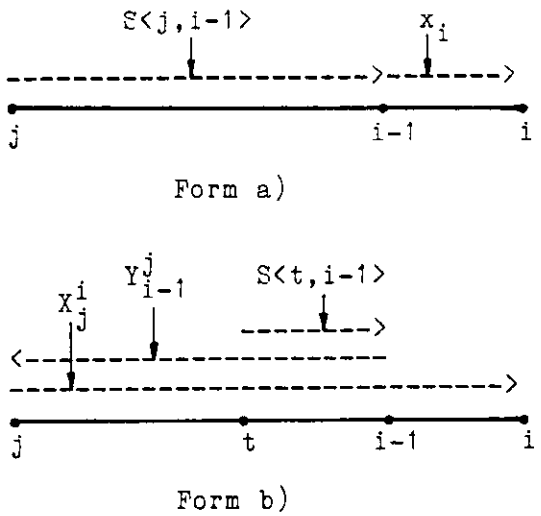

Figure 1. Two Possible Forms of S<j,i>

The recurrence equations for C<j,i>, referred to as Version 1, follows the form of S<j,i> described above.

$$C<i,i> = 0 \tag{1}$$

$$C<j,i> \underset{1<j<i}{=} \min \begin{cases} C<j,i-1> + cost(x_i) & (2.a) \\ \min_{j<t<i} \{*\} & (2.b) \end{cases}$$

where * is $C<t,i-1> + cost(Y^j_{i-1}) + cost(X^i_j)$. Equations (2.a) and (2.b) express the cost of S<j,i> in form a and form b respectively. Clearly $cost(x_i)$ in equation (2.a) is $s(J_{i-1} = <j,i-1>)$ and it is given as input. The costs of (Y^j_{i-1}) and (X^i_j) in equation (2.b) can be expressed as

$$cost(Y^j_{i-1}) = \sum_{k=i-2,...,j} cost(y_k) = \sum_{k=i-1,...,j+1} s(J_k) \tag{3}$$

where $J_k = <q,i-1>$ and $q = \min(k,t)$,

$$cost(X^i_j) = \sum_{k=j+1,...,j} cost(x_k) = \sum_{k=j,...,i-1} s(J_k) \tag{4}$$

where $J_k = <j,i-1>$.

The set of all terms C<j,i> for i=1,.,n and j=1,.,i can be computed using the recurrence equations (1) through (4) in n passes as shown in Figure 2. In computing a term, say C<j,i> for $j \neq i$, a set of terms that have already been computed in the previous pass, namely C<j,i-1> for $j<t<i$, are used. The terms used in computing each new term are shown in Figure 2 by means of directed arrows. The notation A→B indicates that the term A is used in computing B. Moreover the number of comparisons done in computing each term is also shown. The notation ${}_mA$ indicates that m comparisons are made in computing term A. It follows from Figure 2 that a total of $(n^3-3n^2+2n)/6$ comparisons are made in the process. The terms $cost(Y^j_{i-1})$ and $cost(X^i_j)$ can be computed by gradual increments over the corresponding terms used in computing a prior cost term and using $O(n^2)$ space. Hence, the time complexity of the process is $O(n^3)$.

Once the cost of all optimal substrategies that reduce R_n in some join range are found, the cost of the optimal solution strategy S_{op} that will fully reduce R_1 can be determined as

$$cost(S_{op}) = \min_{t=1,...,n}(C<t,n> + cost(Y^1_n)) \tag{5}$$

where S<t,n> reduces R_n in some join range $J_n = <t,n>$ and Y^1_n transmits the result to N_1, thereby fully reducing R_1. Note that

$$cost(Y^1_n) = \sum_{k=n,...,2} s(J_k)$$

where $J_k = <q,n>$ and $q = \min(k,t)$.

As a by-product of the above process, the substrategies S<j,i> for i=1,...,n and j=1,...,i and thereafter S_{op} can be expressed as sequences of semi-joins, by keeping a record of the term that yields to a minimum in the computation of each new term and later by a back-tracing process over the records which are kept. The details of this bookkeeping and back-tracing process are given in Gursel (1983) and will not be addressed here. The process, however, does not affect the time and space complexity of the algorithm.

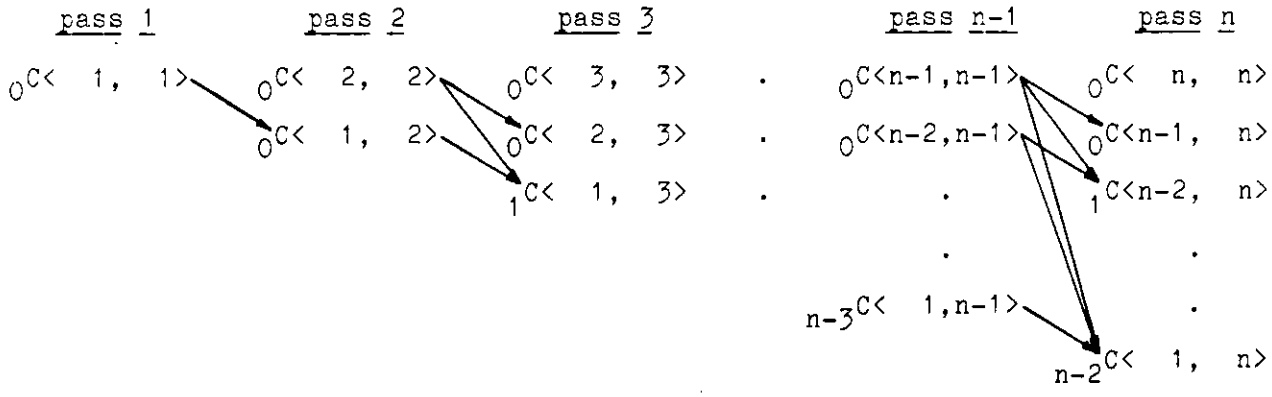

Figure 2. Computation of C<j,i> for i=1,...,n and j=1,...,i

3. IMPROVING THE PERFORMANCE

We have introduced the recurrence equations referred to as Version 1 of the dynamic programming algorithm. In this section, we will modify equations (1) and (2) in two steps into Version 2 and Version 3 and improve the response time of the algorithm significantly under favorable input conditions.

The number of terms computed and the number of comparisons made in computing each term in the recursion can be minimized with the following observations.

Observation 1: Consider the ith pass of the recursion. Let $j=m_i$ be the smallest index that minimizes the following expression:

$$\min_{j=1,\ldots,i} \; (C<j,i> + s(J_i<j,i>)) \qquad (6)$$

Note that C<j,i> in equation (6) is the cost of reducing R_i in the join range J_i = <j,i> and the second term is the cost of transmitting the result to an adjacent node. We then observe that substrategies S<j,i> for $j>m_i$ are suboptimal with respect to $S<m_i,i>$ since R_i can be reduced in a wider join range by $S<m_i,i>$ and later be transmitted to an adjacent node for less cost. Hence, these substrategies can be eliminated from further consideration in the subsequent pass. That is, in the i+1 pass:

1. The terms C<j,i+1> for $j=m_i+1,.,i$ need not be computed.

2. In computing C<j,i+1> for $j=1,.,m_i$, terms of the i^{th} pass, namely C<t,i> for $t>m_i$ can be excluded from comparisons since they will not yield to minimal values.

The implications of Observation 1 are graphically illustrated in Figure 3. Note the reduction in the number of terms computed and comparisons performed in the i+1 pass.

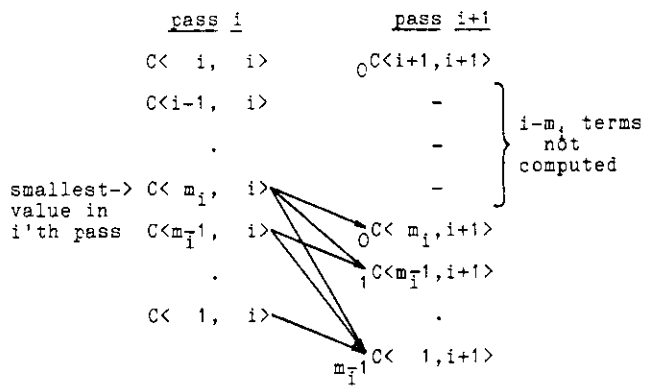

Figure 3. Implications of Observation 1

We can then rewrite our recurrence equations (1) and (2) as **Version 2**:

$$C<i,i> = 0 \qquad (7)$$

$$C<j,i> = \min_{j=m_{i-1},\ldots,1} \begin{cases} C<j,i-1> + s(J_{i-1} = <j,i-1>) & (8.a) \\ \min_{t=m_{i-1},\ldots,j+1} \{ * \} & (8.b) \end{cases}$$

where * is as given in equation (2). Note that the index m_i can be found in linear time using equation (6).

Consider the case where at each pass C<1,i> turns out to be the smallest term. Then the algorithm, as shown in Figure 4, computes at most two terms and performs a single comparison to determine the minimal term at each pass. Hence, its best case time complexity is O(n).

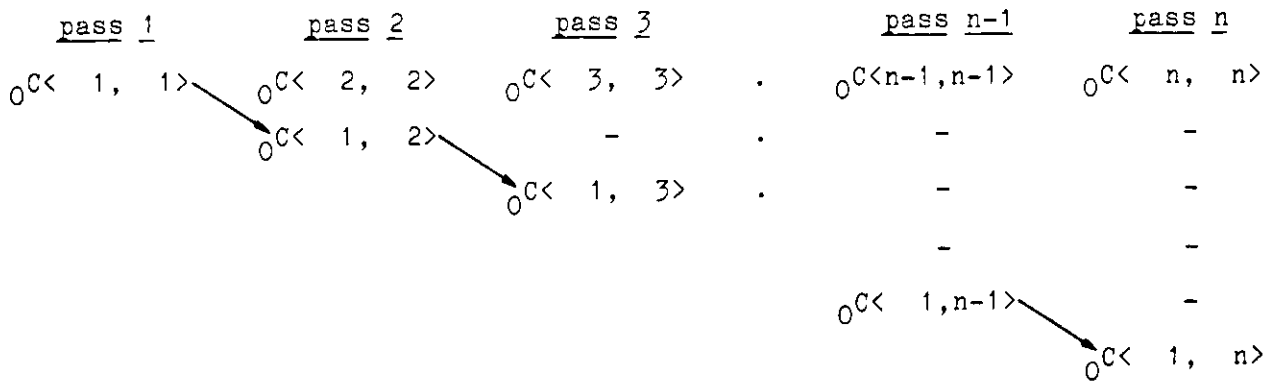

Figure 4. Best Case Performance

Consider the case where at some i^{th} pass, C<i,i> turns out to be the smallest term. Then in the subsequent i+1 pass all the i+1 cost terms need to be computed in order to determine the optimal substrategy. Even in such a case, the number of comparisons to be performed in computing each one of those cost terms can be minimized by considering ranked submininal terms of the previous pass as discussed below.

Observation 2: Consider the i^{th} pass of the recursion. Let $IS_i = \{m_i^1, m_i^2, ..., m_i^k\}$ be an ordered set of $1 \le k \le i$ indices such that $j = m_i^1$ is the smallest index that minimizes

$$\min_{j=1,...,i} \; (C<j,i> + s(J_i = <j,i>)) \qquad (9)$$

$j = m_i$ for some $1 < p \le k$ is the smallest index that minimizes

$$\min_{j=1,...,m_i^{p-1}-1} \; (C<j,i> + s(J_i = <j,i>)) \qquad (10)$$

and $m_i^k = 1$. Note that, by the above definition, $C<m_i^1,i> < C<m_i^2,i> < \cdots < C<1,i>$ and $i > m_i^1 > m_i^2 > \cdots > m_i^k = 1$.

We then observe that substrategies S<j,i> for $i > j > m_i^1$ are suboptimal with respect to $S<m_i^1,i>$ since R_i can be reduced in a wider join range by $S<m_i^1,i>$ and can later be transmitted to an adjacent node for less cost. Similarly, substrategies S<j,i> where $m_i^p > j > m_i^{p+1}$ for some $p<k$ are suboptimal with respect to $S<m_i^{p+1},i>$. Clearly the suboptimal substrategies can be eliminated from further consideration in the subsequent pass. That is, in the i+1 pass:

1. $C<j,i+1>$ for $j = m_i^1+1,...,i$ need not be computed.
2. In computing $C<j,i+1>$ for $j = 1,...,m_i^1$, terms of the i^{th} pass, namely $C<x,i>$ where $x \in IS_i$, can be excluded from comparisons.

The implications of Observation 2 are graphically illustrated in Figure 5. Note the reduction in the number of comparisons performed in the i+1 pass.

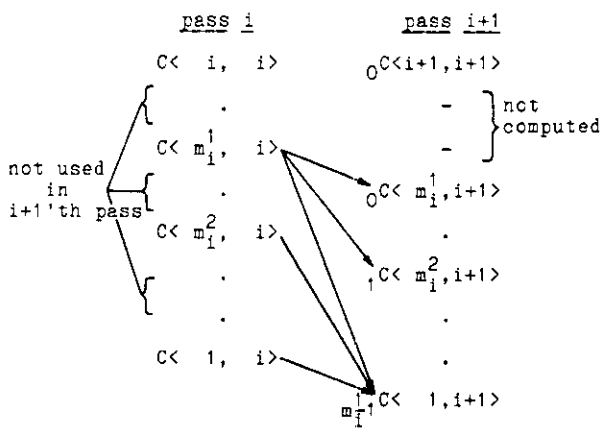

Figure 5. Implications of Observation 2

Based on the above observations, equations (7) and (8) can be revised once again as **Version 3**:

$$C<i,i> = 0 \qquad (11)$$

$$C<j,i> = \min_{j \in IS_{i-1}} \begin{cases} C<j,i-1> + s(J_{i-1} = <j,i-1>) & (12.a) \\ \min_{t \in IS_{i-1},\, t>j} \{ * \} & (12.b) \end{cases}$$

$$C<j,i> = \min_{\substack{j \in IS_{i-1} \\ j<m^1_{i-1}}} \{ * \} \quad t \in IS_{i-1},\ t>j \qquad (13)$$

where * is as given in equation (2). The index set IS_i can be determined by scanning the cost terms $C<j,i>$ for $j=1,...,m^1_{i-1}$ and i, in that order, starting with $C<1,i>$. The index of each successive smaller term is included in IS_i.

The following pseudo-code is used:

```
IS_i: = {1};

prev: = 1;

for j: = 2,...,m^1_{i-1} and i do
    begin
        if C<j,i> <C<prev,i> then
            begin IS_i: = jUIS_i; prev: = j end;
    end;
```

It should be noted that introduction of the procedure that finds the index-set IS_i at each pass adds on to the time complexity of the algorithm by $O(n^2)$ in the worst case (i.e., i terms are computed at each pass i = 1,...,n, see Figure 2) and O(n) in the best case (i.e., only two terms are computed at each pass, see Figure 4). With the above modifications the worst case and best case time performance of the algorithm does not change in order. On the other hand, the average case performance has been investigated through a simulation where all three versions were run against 50 randomly generated distributions of relations over n nodes. Estimates on the sizes of the intermediate relations were calculated based on the "uniform distribution of values in the join-attribute" assumption (Scheuermann and Gursel 1984). As a part of the simulation, we have also compared our algorithm with the dynamic programming algorithm originally developed by Chiu (1980) for chain queries after adopting it for our environment and modifying it for efficiency. Chiu's algorithm has a worst case and average case time complexity of $O(n^3)$. The results of the simulation are shown in Figure 6 where the average response time of each version is plotted as a function of n. It is clear that Version 3 of our algorithm outperforms all the other versions for n>16 despite the overhead due to the computation of the index sets. Figure 6. Average Computation Times versus n.

4. CONCLUSION

A dynamic programming algorithm for the optimal processing of simple queries in chain networks was introduced. The algorithm is known to be applicable to similar problems in ring networks. It can also be incorporated into heuristic approaches to process general queries in networks not necessarily completely connected. The algorithm was modified in two steps and its performance was improved under favorable input conditions. The best case time complexity for the algorithm is shown to be O(n).

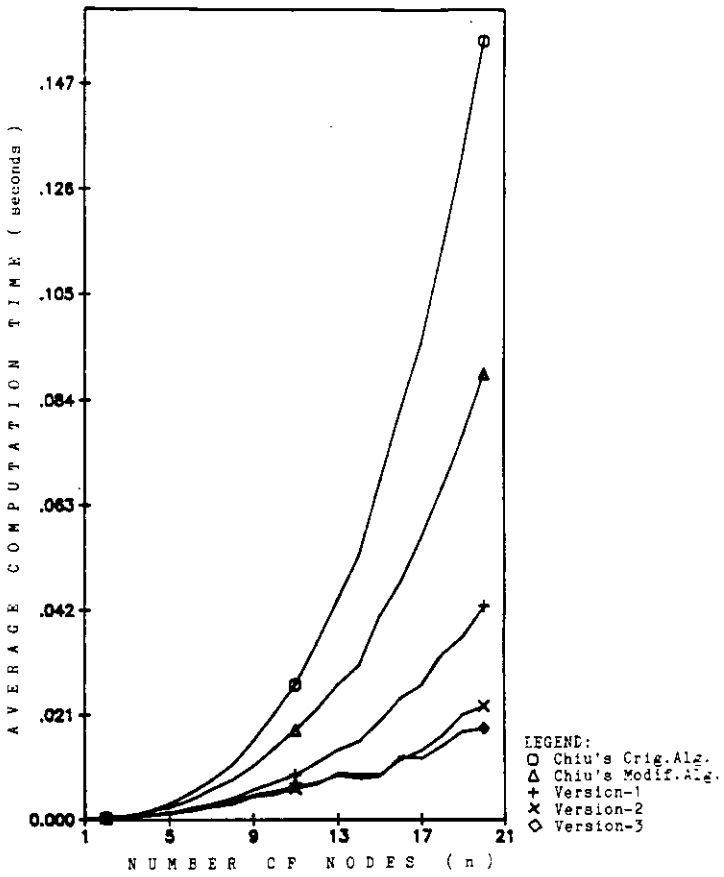

Figure 6. Average Computation Times versus n

5. REFERENCES

Apers, P. M.; Hevner, A. R.; and Yao, S. B. "Optimization Algorithms for Distributed Queries." *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 1, pp. 57-68, January 1983.

Chiu, D. M. "Optimal Query Interpretation for Distributed Databases." Unpublished Ph.D. Thesis, Harvard University, Division of Applied Sciences, December 1980.

Gursel, G. "Optimization of Query Processing in Distributed Database Systems." Unpublished Ph.D. Thesis, Northwestern University, Department of Electrical Engineering and Computer Science, August 1983.

Gursel, G., and Scheuermann, P. "Asserting the Optimality of Serial SJRPs in Processing Simple Queries in Chain Networks." *Information Processing Letters*, Vol. 19, No. 5, pp. 255-260, November 1984.

Gursel, G. "Optimal Processing of Simple Queries in Ring Networks." *IEEE Proceedings of COMPCON Conference*, pp. 422-428, March 1986.

Hevner, A. R. "The Optimization of Query Processing in Distributed Database Systems." Unpublished Ph.D. Thesis, Purdure University, December 1979.

Kerschberg, L.; Ting, P. D.; and Yao, S. B. "Query Optimization in Star Computer Networks." *ACM Transactions on Database Systems*, Vol. 7, No. 4, pp. 678-711, December 1982.

Scheuermann, P., and Gursel, G. "Optimal Processing of Simple Queries in Chain Networks." *IEEE Proceedings of Trends and Applications Conference*, pp. 183-189, May 1984.

Sugihara, K.; Miyao, J.; Kikuno, T.; and Yoshida, N. "Optimization Algorithms for Processing Simple Queries in Star Networks." *IEEE Proceedings of COMPSAC Conference*, pp. 547-554, November 1984.

Yu, C. T., and Chang, C. C. "Distributed Query Processing." *ACM Computing Surveys*, pp. 399-433, December 1984.