CCAM: A Connectivity-Clustered Access Method for Aggregate Queries on Transportation Networks : A Summary of Results

Shashi Shekhar Duen-Ren Liu

Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
shekhar@cs.umn.edu dliu@cs.umn.edu

Abstract

CCAM is an access method for general networks. It uses connectivity clustering. The nodes of the network are assigned to disk pages via the graph partitioning approach to maximize the CRR, i.e., the chances that a pair of connected nodes are allocated to a common page of the file. CCAM supports the operations of insert, delete, create, and find as well as the new operations, get-A-successor and get-successors, which retrieve one or all successors of a node to facilitate aggregate computations on networks. CCAM includes methods for static clustering, as well as dynamic incremental reclustering, to maintain high CRR, in the face of updates without incurring high overheads. Experimental analysis indicates that CCAM can outperform many other access methods for network operations.

1 Introduction

Aggregate queries on networks arise in many important database applications, including transportation planning, air traffic control, electric and gas utilities, telephone networks, sewer maintenance and irrigation canal management. The phenomena of interest in these applications are limited to a finite collection of points (nodes) with locations and the line-segments (edges) connecting the points. Aggregate queries on networks perform connectivity-based computations including route evaluation, path comparison, tour evaluation and location-allocation evaluation [10, 19]. Databases are increasingly being used to store network data and to carry out network computations within a reasonable response time. A lot of research has been carried out within the database area in the design and evaluation of algorithms for the shortest path computation. However, there has been little work in the design of efficient storage and access methods for network data and aggregate queries on networks. Efficient access methods are available for a severely restricted class of networks, namely directed acyclic graphs [12, 7, 3, 18] and directed graphs with limited cycles [2], which do not adequately model many networks of interest, including road-maps. Thus, for general networks, there is a need for the design and evaluation of efficient storage and access methods for aggregate queries on networks such as road maps.

1.1 Application Domain: IVHS and Route Evaluation Queries

IVHS [1] is currently being developed to improve the transportation infra-structure and thus improve safety, increase efficiency, and productivity by applying sensor, database, communication and other emerging technologies [6]. An important part of IVHS is a geographic database containing road maps, public transportation routes, and current travel time on segments of transportation network, which is updated frequently. Route planning [24] is a useful function of the IVHS database to allow travelers to compare alternate routes between their origin and destination before travel begins. Route planning for daily commuters often consists of evaluating a set of familiar routes based on the current travel-time, congestion, restrictions and other attributes of transportation network. This computation can be formulated as instances of an aggregate query, called *route evaluation*, over specified routes in the road-network database.

Route evaluation arises in many other important applications and several GIS support special data-type of a route-unit which represents a collection of arcs with common characteristics (e.g. name) [19]. The attribute data is often aggregated over edges and nodes inside or connected to route-units to present a summary information for decision support applications. For example, the managers of public transit

[1] Intelligent Vehicle Highway Systems, also known as Intelligent Transportation Systems

may like to compare ridership on different bus routes to determine number of buses to be allocated to different routes. Utility companies may track the volume of gas/electricity flowing through major pipeline route-units in their network. Processing aggregate queries over route-units in networks may require the retrieval of all nodes and all edges in the specified route-units to derive aggregate properties.

1.2 Access Methods for Networks

To be efficient, the storage and access methods for networks should take the connectivity properties of networks into account. Aggregate queries on networks and the management of network data require the efficient support of the following set of operations : **Create(), Find(), Insert(), Delete(), Get-A-successor() and Get-successors()**. The first four operations are common to data types other than aggregate queries on networks. The Get-A-successor() and Get-successors() operations are unique to aggregate queries on networks, and they retrieve one or all successors of a node. For example, Get-A-successor() is used in route evaluation queries, while Get-successors() is used in graph search algorithms like A^* [24]. While Get-successors() and Get-A-successor() can be implemented as sequences of Find() on relevant successors, more efficient implementations are possible by defining these operations as distinct. The Get-successors() and Get-A-successor() operations represent the dominant I/O cost of many aggregate queries on networks.

Related Work: The literature on transitive closure and recursive-query processing has evaluated algorithms for path computations. A survey of the work can be found in [13]. The effect of efficient storage and access methods on the performance of path computations is currently being explored. Most of the proposed methods have looked at storing nodes of a directed acyclic graph in topological order [18], using a conventional index like the B-tree. Path computations, such as graph traversal and transitive closure, can be carried out by scanning forward in the file, using a priority queue [18] or a FIFO queue [12]. Topological orders, including depth-first sequence and breadth-first sequence, have been evaluated in [3] for their effectiveness in supporting different graph-traversal problems. Finally, the topological ordering method has been extended to graphs with a few cycles in [2]. In addition, spatial access methods were evaluated for benchmark path computations in [23]. A more detailed survey is available in [20]. Meanwhile, static schemes based on the graph-partitioning heuristic, albeit in a different context, was recently used in [26]. The issues involved in dynamic updating effects during insertion and deletion have not been discussed.

Our Approach: In general, graphs can be represented in many different ways. We will focus on the adjacency-list oriented representation, which has been used quite frequently in database research [14]. In this paper, a network (structurally identical to a graph) is modeled as a list of nodes, and each node has attributes named successor-list and predecessor-list, which represent the outgoing and incoming edges. The predecessor-list facilitates updating the successor-lists during the insertion and deletion of nodes. We focus on developing access methods which can provide efficient support of Get-A-successor() and Get-successors(), along with the traditional operations.

The goal of an access method is to minimize the total I/O cost of aggregate queries when accessing the nodes and edges of a given network. In section 3.1, we show that the expected cost of accessing an edge and the expected cost of network operations are minimized by maximizing *Weighted Connectivity Residue Ratio (WCRR)*, where

$$\text{WCRR} = \frac{\text{Sum of w(u, v) such that Page(u) = Page(v)}}{\text{Total sum of weights over all edges}}.$$

The weight $w(u, v)$ associated with $edge(u, v)$ represents the relative frequency of a query accessing nodes u and v together. To gain insight into the result, let us consider *Connectivity Residue Ratio (CRR)* which is a special case of WCRR where each edge in the network is equally likely to be accessed. An unsplit edge (u,v) is characterized by page(u) = page(v).

$$CRR = \frac{Total\ number\ of\ unsplit\ edges}{Total\ number\ of\ edges}.$$

This measure can be understood in the context of the Get-successors() operation as well. The high value of CRR implies that a high fraction of the neighbors of the given node are in the same page as the given node. Thus, fewer pages are to be retrieved to main memory to process the Get-successors(). Similarly, the expected cost of Get-A-successor() is less if the CRR is higher. In the adjacency-list representation of the network, the main effect of the access method on the total I/O cost of many aggregate queries can thus be predicted from the CRR (WCRR) measure. Higher CRR (WCRR) indicates lower I/O cost for aggregate queries on networks.

Contributions: We propose a new access method, CCAM, to efficiently support aggregate queries over general networks such as road maps. We adapt the

heuristic graph-partitioning approach [2] to cluster the nodes of a given network into file pages by the connectivity relationship. Ideally, the clustering maximizes WCRR. In particular, we address the following two issues. First, the static graph-partitioning approach is not efficient when the entire network can not fit into main memory. In general, road-maps are really large databases [16, 1], and thus may not fit inside main memory. Secondly, maintaining higher CRR in the face of Insert() and Delete() operations, without complete reorganization, is a critical problem. To solve the above two issues, we propose *dynamic reclustering strategies* to handle dynamic updating effects.

Outline: Section 2 defines the CCAM access method. Section 3 presents an algebraic analysis, and section 4 describes the experimental evaluation of CCAM. Finally, section 5 summarizes our conclusions and suggests future work.

2 CCAM: Connectivity-Clustered Access Method

CCAM clusters the nodes of the network via graph partitioning, using the ratio-cut heuristic [5]. Other graph partitioning methods can also be used as the basis of our scheme. In this section, we describe the file-structure and procedures used to implement various operations on networks.

2.1 Connectivity-Clustered Data File

For each node, a record stores the node data, successor-list and predecessor-list. A successor-list (predecessor-list) contains a set of outgoing (incoming) edges, each represented by the node-id of its end (start) node and the associated edge cost. The successor-list is also called the adjacency-list, and is used in network computations. The predecessor-list is used in updating the successor-list during the Insert() and Delete() operations. We will refer to the neighbor-list of a node x as the set of nodes whose node-id appears in the successor-list or predecessor-list of x. We note that the records do not have fixed formats, since the size of the successor-list and predecessor-list varies across nodes.

In contrast with the previous topological ordering based approach [18] , we use connectivity clustering to cluster node records into data pages. CCAM assigns nodes to the data page by the graph partitioning approach, which maximizes CRR. Each data page is kept at least half full whenever possible. Records of the data file are not physically ordered by node-id values. A secondary index is created on top of the data file.

Since our benchmark networks are embedded in geographic space, x,y coordinates for each node are also stored in the record. A B^+ tree with Z-ordering [22] of the x, y coordinates is used to order the secondary index. It can support point and range queries on spatial databases. Other access methods such as R-tree [11] and Grid File [21], etc. can alternatively be created on top of the data file as secondary indices in CCAM to suit the application.

Example: In figure 1, a sample network and its CCAM is shown. We assume that the Z-order of the node-id values is represented by the alphabetical order of the node names. The dashed lines drawn in the sample network represent the cut of the graph. The nodes in the network are clustered into three partitions and are stored on three data pages.

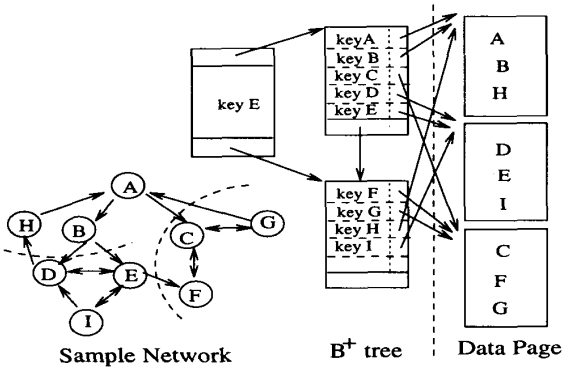

Figure 1: Clustering and storing a sample network

2.2 Create(): Creation of CCAM

The creation of a file is based on the graph partitioning approach. First, the nodes of the network are clustered via the cluster-nodes-into-pages(), which returns a set of pages. Secondly, the nodes(records) which belong to the same subset are stored on the same data page and an index entry for each node is created and inserted into the B^+ tree, based on the node-id values which represent the Z-order of the location of the nodes in space.

We adapt Cheng and Wei's two-way ratio-cut heuristic algorithm [5] as the basis for our connectivity based clustering method. Figure 2 shows the

[2] The literature in the area of graph partitioning [4, 8, 15, 5] albeit in a different context and has only focused on partitioning static graphs, without considering dynamic updates.

connectivity-based clustering algorithm for top-down clustering using the 2-way-partition-graph algorithm. We repeatedly apply the 2-way-partition-graph() to cluster the graph. After applying the 2-way-partition-graph(), two subsets return from the algorithm. We keep on applying the 2-way-partition-graph() to the subset, which exceeds the page-size, until all subset sizes are less than the page-size. Notice that sizeof(A) = $\sum_i$ size of record(i), node i $\in$ A.

```
Procedure: cluster-nodes-into-pages(V: set of nodes;
      E: set of edges; page-size): return set of pages;
 F,P : set of page (partition) of nodes;
 V', A, A' : set of nodes;
begin
 Initialize F = {V}; P = {}; MinPgSize = ⌈ page-size/2 ⌉;
 while F is not empty do
    Choose a V' ∈ F, E'={(u,v)|(u,v)∈E, u∈V' and v∈V'};
    Remove V' form F;
    <A, A'> = 2-way-partition-graph(V', E', MinPgSize);
 // comment: sizeof(A) > MinPgSize;
 // comment: sizeof(A') > MinPgSize;
    if sizeof(A) > page-size then add A to F
                             else add A to P;
    if sizeof(A') > page-size then add A' to F
                             else add A' to P;
 endwhile
 return P;
end;
```

Figure 2: The Connectivity Clustering Algorithm

Other graph partitioning methods can also be used as the basis of our scheme. In fact, M-way partitioning [15, 27] may be used to further improve the result of partitioning, if computation complexity and CPU cost is not a concern.

Static-Create() operation is not efficient when the entire network does not fit inside main memory. Incremental Create() operation is designed to handle very large networks. The **incremental Create()** operation is implemented as a sequence of Add-node() operations, which are similar to Insert() operations described in section 2.4. The Add-node() operation does not need to update the successor and predecessor lists. This operation will, however, use incremental clustering and reorganization to improve the CRR, as discussed in section 2.4. We use **CCAM-D** to denote the implementation of Create() as a sequence of Add-node() operations. **CCAM-S** denotes the implementation of Static-Create().

2.3 Efficient Support of Search Operations

Find(): Retrieve record of a given node-id
Find() operation is implemented by searching the secondary index to read the appropriate data page.
Get-A-successor(): Retrieve a specified successor of the node in memory buffer
In principle, the buffered data-page containing the node is likely to contain the specified successor node if CRR is high. Thus the buffered data-page should be searched first. If the desired successor node is not in the buffer, then a Find() operation is needed to retrieve it.
Get-successors(): Retrieve records for successor nodes of a given node-id
In principle, when a data page is fetched for the purpose of retrieving the current node (i.e., the given node), all successor neighbors stored in the same data page as the current node would be accessed without further I/O. Since CCAM clusters nodes based on CRR, there is a high probability that many successors will be located in the same disk page as node x. The Get-successors() procedure can be improved further by checking all pages brought into main memory buffers by the Find() operation, to check if additional neighbor records can be extracted without additional Find() operations.
Route Evaluation
A route specifies a sequence of nodes $n_1, n_2, \ldots, n_k$ and edges $< n_1, n_2 >, < n_2, n_3 >, \ldots, < n_{k-1}, n_k >$. The aggregate property of a route is a function of the properties of the nodes and edges in the route. The aggregate property of a route can be computed by a sequence of Find() operations on relevant nodes and edges. Alternatively, it can be processed as a sequence of Get-A-successor() operations, e.g. Find(n_1), Get-A-successor(n_1, n_2), ..., Get-A-successor(n_{k-1}, n_k). Thus, efficient implementation of Get-A-successor() operations reduces the total I/O cost for route evaluation queries.

2.4 Maintenance and Dynamic Reclustering Strategies

There are two basic maintenance operations: Insert() and Delete(). Each can take an argument of an edge or a node. These operations change connectivity relationships, and may make the existing partitioning of the network to pages obsolete. Local reorganization of the data pages may be needed to improve the CRR. Intuitively, the data set chosen for reorganization should be those data pages which are related via

Reorganization Policy	Set of Pages to be Reorganized		
	argument = edge(u,v)	**argument = node x**	**Guiding Principle**
First order	none handle underflow/overflow	none handle underflow/overflow	avoid or delay reorganization
Second order	{Page(u), Page(v)}	{Page(x)} ∪ PagesOfNbrs(x)	reorganize pages which must be updated anyhow
Higher order	1. NbrPages(Page(u))∪{Page(u)} ∪NbrPages(Page(v))∪{Page(v)} or 2. {Page(u)}∪PagesOfNbrs(u)∪ {Page(v)}∪PagesOfNbrs(v) or 3. all pages in data file	1. {Page(x)} ∪ PagesOfNbrs(x) ∪ NbrPages(Page(x)) or 2. all pages in data file	reorganize more pages than second order policy
Page(x) = page selected to place x in Insert() or page containing x in Delete()			

Table 1: Set of Pages reorganized by different Policies for Maintenance

the connection between nodes. We adapt the notion of the page access graph (PAG) [17] to formalize the connectivity relationship between data pages.

Definition 1 (Page Access Graph) *Let $G = (V, E)$ be the given network. P is called a page of G if and only if P is a set of records, such that for each record(x) $\in P$, $x \in V$ and all records $\in P$ are stored in the same disk data page, i.e., the total size of the records included in P is at most full disk page size. Let each of $P_1, P_2, \cdots, P_n$ be a page of G. Then the page access graph (PAG) $G_p = (V_p, E_p)$, where V_p is a set of pages and E_p is a set of edges defined as follows:*
$V_p = \{P_1, P_2, \cdots, P_n\}$,
$E_p = \{(P_i, P_j) \mid \exists\, x, y$ *such that* $x \in V, y \in V, (x, y) \in E$, *record*$(x) \in P_i$, *and record*$(y) \in P_j\}$

Definition 2 :
· *Is-Neighbor-Page(P,Q) = true iff either (P,Q) $\in E_p$ or (Q,P) $\in E_p$.*
· *NbrPages(P$\in V_p$) = {Q | Q$\in V_p$ and Is-Neighbor-Page(P,Q)}.*
· *Page(x $\in$ V) = Q, where Q$\in V_p$ and record(x)$\in$ Q.*
· *PagesOfNbrs(x$\in$ V) = {Page(u) | u$\in$ succ(x) ∪ pred(x)}.*

The principle of our dynamic reclustering strategy is to reorganize a suitable set of pages which are connected in the page access graph. The reorganization is performed by applying the cluster-nodes-into-pages() algorithm described in figure 2 to recluster the subnetwork formed from nodes in the set of pages to be reorganized. For such a set of pages to be reorganized, the choice might be based not only on maximizing the CRR but also on reducing the overhead required in reorganization.

The key issue in the design of maintenance operations is the identification of a reorganization policy which yields a high CRR without incurring high I/O costs. The reorganization policies can be defined in terms of the concept of a page access graph, as shown in table 1. The table identifies a set of pages to be reorganized, given the argument type (edge or node) and the reorganization policy (1st, 2nd, higher). It assumes that Page(x) represents the page containing x, or selected to contain x in the event of Insert(x). To simplify this table, overflow and underflow events are abstracted and are discussed separately. The second order policies are designed to avoid additional I/O overhead in reorganization. Second order and higher order policies can incur a high CPU cost if the average degree of nodes increases. Other reorganization policies can be built around the basic policies shown in table 1. For example, a lazy or delayed reorganization policy may reorganize NbrPages(P) after a certain number of updates to page P. The order of reorganization policy represents the order of overhead required during the update. In general, a higher order policy can yield higher CRR, but it incurs higher overhead.

The efficient implementation of the higher order policies may require additional data structures. NbrPages(P$\in V_p$) can be retrieved efficiently if the page access graph is materialized, to avoid repeated traversal of the secondary index. We choose not to materialize the page access graph, since it requires additional redundant data structures.

Insert(): Insert a new node or edge
The Insert() operation is used to add a new edge or a node to the data file. During insertion of a new node x, a data page must be selected in which to store the new node. To maximize CRR, the new node should

be placed in a page containing many of its neighbors. Page selection may be accomplished by ranking the pages by the number of neighbors of x located in the page, to choose the page with the maximum number of neighboring nodes of x which also has space to accommodate x. In case of overflow in any of the updated pages, the overflow page is split into two pages, via the cluster-nodes-into-pages() procedure. Reorganization may be carried out on the pages specified by table 1, via the cluster-node-into-pages() procedure described in figure 2. Finally, the index is updated to reflect the changes to the data file. Figure 3 shows a procedural description of Insert() for node arguments.

```
Procedure: Insert(x: node-id; record(x): node-properties;
          policy: reorganization-policy)
begin
 retrieve PagesOfNbrs(x);
 if PagesOfNbrs(x) is empty then
    insert record(x) into an available disk page P;
    insert index entry (node-id x, disk address of P);
 Otherwise,
    update succ-list and pred-list of neighbors(x);
    select a page P from PagesOfNbrs(x) to put record(x);
    if (policy == first-order policy) then
       for each page Q in PagesOfNbrs(x) do
          if Q overflow then split Q into two pages
          else if Q has been modified then Write Q;
    else
       Reorganize(x, policy);
end;
```

Figure 3: The Insert Algorithm for Nodes (records)

Delete(): Delete a node or edge

The Delete() operation can be used to delete an edge or a node from the data file. In the case of underflow, data-page merging may be required. In addition, reorganization may take place according to the specified policy. Figure 4 shows the delete algorithm.

```
Procedure: Delete(x: node-id; policy: reorganize-policy)
begin
 retrieve P = Page(x); retrieve PagesOfNbrs(x);
 update succ-list and pred-list of neighbors(x);
 delete x from P; delete index entry of x;
 if (policy == first-order) then
    if page P underflow then
       select a page Q from PagesOfNbrs(x);
       perform data page merging on {P, Q};
    for each page Q in {PagesOfNbrs(x), P} do
       if Q has been modified then Write Q;
 else
    Reorganize(x, policy);
end;
```

Figure 4: The Delete Algorithm for Nodes

3 Analytical Evaluation and Cost Models

In this section, we illustrate theorem 1 and provide simple algebraic cost models for network operations and route evaluation queries.

3.1 Is WCRR the Right Metric?

Theorem 1 *The expected cost of accessing an edge and the expected cost of network operations are minimized by maximizing WCRR.*

Proof: The proof is based on binary cost model. The cost of Get-A-successor() is high if successor is not in the same page as the node. The cost is low, otherwise. In this model, the expected cost of Get-A-successor() can be expressed as a linear monotonically decreasing function of WCRR. The details are available in [20]. □

Theorem 1 suggests that the expected cost of aggregate queries over a network is minimized by designing an access method customized to maximize WCRR or the sum of weights over unsplit edges. It can be easily shown that the problem of partitioning the nodes of a network into pages of a given size, so as to maximize the WCRR, is an instance of the graph/circuit partitioning problem defined in [15]. Although the graph partitioning problem is NP-complete [9], many good heuristics based on iterative approaches [8, 15, 5] have been proposed to solve this problem efficiently. The implementation of CCAM operations takes advantage of these heuristics.

3.2 Cost Modeling

For simplicity, we assume that each edge is equally likely to be accessed by network operations. This assumption is made only to simplify the analysis. The experimental evaluation considers the general case where weight distribution is not uniform. Table 2 lists the symbols used to develop our cost formulas. The neighbor list of a node x includes all the neighbors of node x, while the successor (adjacency) list of a node x only contains the successor neighbors of node

$\|A\|$	Average number of nodes in the successor-list of a node
α	CRR = Pr.[Page(i)=Page(j)] for edge(i, j)
λ	Average number of nodes in the neighbor-list of a node
γ	Average blocking factor
L	Number of nodes in aggregate queries over routes

Table 2: Notations used in Cost Analysis

x. In the analysis, we focus on the number of data page accesses. A detailed analysis is provided in [20].

The algebraic cost for search operations and route evaluation queries is listed in the table 3. The Get-successors() operation retrieves all successors of a given node x. We assume that the data page containing node x is located in main memory. In general, route evaluation queries can be modeled as a sequence of Get-A-successor() operations. Then the number of data page access for route evaluation queries over L nodes is approximately $1 + (L-1) * (1-\alpha)$, assuming buffering with one data page.

Operation	Data Page Accesses
Get-successors()	$(1-\alpha) * \|A\|$
Get-A-successor()	$1-\alpha$
Route Evaluation	$1 + (L-1) * (1-\alpha)$

Table 3: Cost Analysis

Table 4 summarizes the worst case retrieval (*Read*) cost for Insert() and Delete() operations, under different reorganization policies. To simplify our cost modeling, we assume that the index pages are buffered in main memory and that sufficient buffers are provided for update operations. Thus, we will only focus on the number of data pages accessed in each update operation. In general, the *Write* cost is equal to the *Read* cost, unless there is underflow or overflow. To simplify our comparison, we assume they are the same.

Policies	Data Page Accesses	
	Insert()	Delete()
first-order	λ	$1 + \lambda * (1-\alpha)$
second-order	λ	$1 + \lambda * (1-\alpha)$
higher-order	$\lambda + \gamma * \lambda * (1-\alpha)$	$\gamma * \lambda * (1-\alpha)$

Table 4: Simplified worst case retrieval cost for update operations

The analysis of cost modeling for network operations shows that the efficiency of the Get-A-successor(), Get-successors() and Delete() operations depends on the parameter α, i.e., CRR. With a higher CRR, the cost of these operations is lower. It is interesting to note that the cost of the Insert() operation cannot be predicted from the CRR, since the model cannot capture the clustering efficiencies for neighbors of a new node being inserted.

4 Experimental Evaluation

Access methods for networks, and the update policies for CCAM, are evaluated by a series of experiments. Due to space constraints, we only present a subset of the experiments. A full description of these experiments and the results can be found in [20].

We compare our proposed connectivity-based clustering schemes CCAM with the other three schemes, namely, Grid file, DFS-AM and BFS-AM. DFS-AM and BFS-AM are extensions of topological-ordering based files to general graphs. DFS-AM orders the nodes by a depth-first traversal and BFS-AM orders the nodes by a breadth-first traversal from a random starting node. In addition, we compare one variant of DFS-AM, WDFS-AM, which performs a depth first search according to the order of the weights on the edges. CCAM-S uses the static create operation. CCAM-D uses an incremental create() operation with the second-order reorganization policy. The experiments are conducted on the Minneapolis road map consisted of 1079 nodes and 3057 edges, representing the road intersections and highway segments for a 20-square-mile section of the Minneapolis area.

To simplify the comparison, the I/O cost represents the number of data page accessed. This represents the relative performance of various methods for very large databases. In subsection 4.1, 4.2 and 4.4, we use uniform weights (i.e., all weight on edges = 1) to simplify the interpretation of results. In subsection 4.3, we use non-uniform weights on the edges (derived from a given set of routes). We examine the WCRR measure in that set of experiments dealing with route evaluation queries.

4.1 Measurement of CRR Values

Figure 5 shows the CRR for the five access methods on various disk block sizes. As we can see, the CRR values for all access methods increase as the disk block size increases, since a higher disk block size implies a higher blocking factor. The results show that CCAM-S consistently outperforms all the other methods for all four disk block sizes. CCAM-D also performs quite well for all disk block sizes. It has higher CRR than DFS-AM, Grid file and BFS-AM. Although the Grid file is a proximity-based algorithm, it takes

Operation Method	Get-successors()		Get-A-successor()		Delete()		Insert()	α =
	Actual	Predicted	Actual	Predicted	Actual	Predicted	Actual	CRR
CCAM	0.627	0.680	0.209	0.239	3.364	3.532	4.710	0.7606
DFS-AM	1.032	1.108	0.359	0.391	4.280	4.504	4.804	0.6088
Grid File	1.158	1.300	0.448	0.459	4.524	4.935	3.907	0.5414
BFS-AM	1.910	2.555	0.889	0.902	6.392	7.732	5.560	0.0981
			A = 2.833	λ = 3.20	γ = 12.55			

Table 5: I/O cost for Network Operations

advantage of the correlation between connectivity and spatial proximity, and outperforms DFS-AM for large disk block sizes (4k).

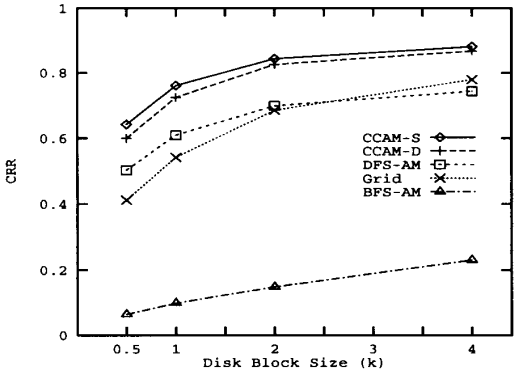

Figure 5: The effect of disk block size on CRR

4.2 I/O Cost for Network Operations

The experiments use the Minneapolis road map with disk block size = 1 k. The cost for the Get-successors() operation is measured via performing the Get-successors() operation on a randomly chosen 50% of the total number of nodes. The costs for other operations are measured similarly. Page underflows and overflows in the Delete() and Insert() operations are ignored to filter out the effect of reorganization policies, which are studied separately. Table 5 shows the average number of data page accesses for each operation under various methods. The predicted cost for the operations is computed via the cost model described in table 3 and 4.

As shown in table 5, the number of data page accesses during Get-A-successor(), Get-successors() and Delete() operations for CCAM is the lowest among all methods. This is expected, since CCAM has the highest CRR. Although CCAM has a slightly lower I/O cost than DFS-AM and BFS-AM in the Insert() operation, its I/O cost is higher than that for the Grid file. For the insert() operation, there might exist few or no connections between neighbors of the node to be inserted, and those neighbors might be in different disk pages before the insertion. Connectivity-based methods (e.g. CCAM, DFS-AM, BFS-AM) may not cluster these neighbors of a new node. The spatial proximity of the neighbors of the new node being inserted helps the Grid file to reduce the I/O cost of Insert().

4.3 I/O Cost for Route Evaluation

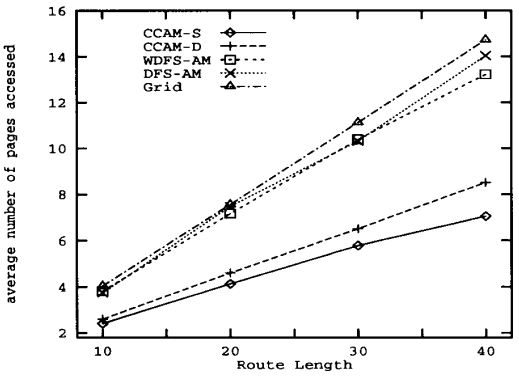

Figure 6: Effect of Route Length

We generate routes by performing random walks on the network. Four sets of routes with lengths equal to 10, 20, 30 and 40 are generated. A route of length L has L nodes and $L-1$ edges. Each set contains 100 routes. The weights on the edges of the network are derived by counting the number of times that an edge is accessed by those routes. The route queries are processed by issuing a Find() followed by a sequence of Get-A-successor() operations. We assume one buffer with the size of one data page. Figure 6 shows the experiment results conducted on block size 2048. We can see an increase in the number of data pages accessed as the length of the routes increases. Both CCAM-S

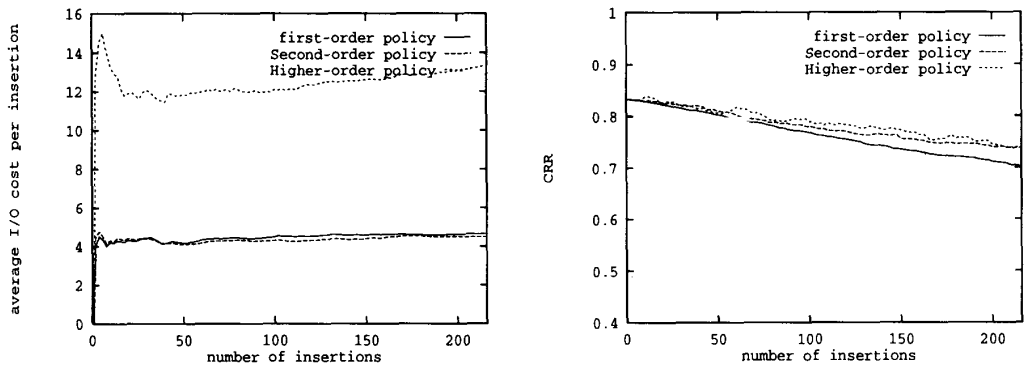

Figure 7: Effect of the Reorganization Policies

and CCAM-D outperform all the other methods under all the route lengths.

4.4 Evaluation of Reorganization Policies

Figure 7 shows the I/O cost and CRR results during the insertion of 20% of the nodes of the Minneapolis road map. It shows that the higher order policy has a much higher I/O cost than the first-order and second-order policy. The average I/O costs for the first-order and second-order policy are very close, as expected. Notice that the average I/O cost for the higher-order policy increases slightly as the number of insertions increases. This is to be expected, since the CRR value decreases as the number of insertions increases, resulting in the increase of the number of data pages accessed for neighboring pages. The CRR might increase during some insertions, but it decreases in the long run, because the average connectivity increases and the average blocking factor is reduced, as more nodes are inserted into the file. The first-order policy has the lowest CRR, and the higher-order policy has a slightly higher CRR than the second-order policy in most cases. The second-order policy is a good choice, since its I/O cost is almost the same as that of the first-order policy, and its CRR is competitive with the higher-order policy.

5 Conclusions and Future Work

Route evaluation and related aggregate queries on networks are used in many important applications of databases, including IVHS. It is important to design storage and access methods to support these applications. We have presented a connectivity-clustered access method (CCAM) for general networks to support aggregate queries over networks. An algebraic cost model for network operations has been derived. The analysis shows that the efficiency of the Delete(), Get-A-successor() and Get-successors() operations depends primarily on the CRR. Experiments show that CCAM provides higher CRR (WCRR) values and incurs lower I/O costs for network operations and route evaluation queries relative to other access methods. We also identify and evaluate alternate reorganization policies to maintain a high CRR, without incurring high reorganization costs, during insertion and deletion of nodes and edges.

Future work includes developing a formal analysis for achievable CRR under different access methods. Also, access cost for secondary indexes should be modeled and evaluated. A more efficient index access structure should be designed that will tolerate incremental reorganization during update operations. Further, the CPU cost for reorganization should be taken into account. Reorganization policies that do not incur high CPU costs are currently being investigated. Finally, we will evaluate CCAM for various aggregate computations over networks and benchmarks (such as the sequoia benchmark [25]). We would also like to evaluate CCAM for other aggregate queries on networks, including tour evaluation, location-allocation evaluation etc.

Acknowledgments

This research was supported by the Federal Highway Authority (FHWA), Minnesota Dept. of Transportation and the Center for Transportation Studies at the University of Minnesota. We would like to thank Dr. H.V. Ja-

gadish (AT&T Bell Labs), and Prof. K. Hua (University of Florida) for helping with the survey and focus of this research. We would also like to thank Prof. C.K. Cheng (University of California, San Diego) and Dr. L.T. Liu (AT&T Bell Labs) for helping with the ratio-cut program.

References

[1] R. Agrawal and H.V. Jagadish. "Algorithms for Searching massive Graphs". *IEEE Trans. on Knowledge and Data Engineering*, 6(2), April 1994.

[2] R. Agrawal and Jerry Kiernan. "An Access Structure for Generalized Transitive Closure Queries". In *Proc. of the Ninth Intl Conference on Data Engineering*, pages 429–438. IEEE, April 1993.

[3] J. Banerjee, S. Kim W. Kim, and J. Garza. "Clustering a DAG for CAD Databases". *IEEE Trans. on Software Engineering*, 14(11):1684–1699, Nov. 1988.

[4] E.R. Barnes. "An Algorithm for Partitioning the Nodes of a Graph". *SIAM Journal Alg. Disc. Meth.*, 3(4):541–550, December 1982.

[5] C.K. Cheng and Y.C. Wei. "An Improved Two-Way Partitioning Algorithm with Stable Performance". *IEEE Trans. on Computer-Aided Design*, 10(12):1502–1511, December 1991.

[6] W.C. Collier and R.J. Weiland. "Smart Cars, Smart Highways". *IEEE Spectrum*, April 1994.

[7] S. Dar and H.V. Jagadish. "A Spanning Tree Transitive Closure Algorithm". In *Proc. of Intl Conference on Data Engineering*. IEEE, 1992.

[8] C.M. Fiduccia and R.M. Mattheyses. "A Linear Time Heuristic for Improving Network Partitions". In *Proc. of 19th Design Automation Conference*, pages 175–181, 1982.

[9] M.R. Garey and D.S. Johnson. *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*. W.H. Freeman and Company, San Francisco, 1979.

[10] M.F. Goodchild. "Towards an Enumeration and Classification of GIS Functions". In *Proc. of Intl Geographic Info. Systems Symp.*, 1987.

[11] A. Guttman. "R-Trees: A Dynamic Index Structure for Spatial Searching". In *Proc. of SIGMOD Intl Conference on Management of Data*, pages 47–57. ACM, 1984.

[12] K. Hua, J. Su, and C. Hua. "An Efficient Strategy for Traversal Recursive Query Evaluation". In *Proc. of the Ninth Intl Conference on Data Engineering*. IEEE, April 1993.

[13] Y. Ioannidis, R. Ramakrishnan, and L. Winger. "Transitive Closure Algorithms Based on Graph Traversal". *ACM Trans. on Database Systems*, 18(3), September 1993.

[14] B. Jiang. "I/O Efficiency of Shortest Path Algorithms: An Analysis". In *Proc. of the Intl Conference on Data Engineering*. IEEE, 1992.

[15] B.W. Kernighan and S. Lin. "An Efficient Heuristic Procedure for Partitioning Graphs". *Bell Syst. Tech. J.*, 49(2):291–307, February 1970.

[16] R. Kung, E. Hanson, and et. al. "Heuristic Search in Data Base Systems". In *Proc. Expert Database Systems*. Benjamin Cummings Publications, 1986.

[17] Y. Kusumi, S. Nishio, and T. Hasegawa. "File Access Level Optimization Using Page Access Graph on Recursive Query Evaluation". In *Proc. Conference on Extending Database Technology*. EDTB, 1988.

[18] P.A. Larson and V. Deshpande. "A File Structure Supporting Traversal Recursion". In *Proc. of the SIGMOD Conference*, pages 243–252. ACM, 1989.

[19] R. Laurini and D. Thompson. *"Fundamentals of Spatial Information Systems"*, chapter 5 and 2.5.4. Number 37 in The A.P.I.C. Series. Academic Press, 1992.

[20] D.R. Liu and S. Shekhar. "CCAM: Connectivity-Clustered Access Method for Networks and Network Computations". Technical Report TR 93-78, Computer Science Dept. University of Minnesota, 1993.

[21] J. Nievergelt, H. Hinteberger, and K.D. Sevcik. "The Grid File: An Adaptable, Symmetric Multi-Key File Structure". *ACM Trans. on Database Systems*, 9(1):38–71, 1984.

[22] A. Orenstein and T. Merrett. "A Class of Data Structures for Associative Searching". In *Proc. Symp. on Principles of Database Systems*, pages 181–190. SIGMOD-SIGACT PODS, 1984.

[23] S. Shekhar, A. Kohli, and M. Coyle. "Can Proximity-Based Access Methods Efficiently Support Network Computations?". Technical Report TR-93-57, Computer Science Dept. University of Minnesota, 1993.

[24] S. Shekhar, A. Kohli, and M. Coyle. "Path Computation Algorithms for Advanced Traveler Information System". In *Proc. of the Ninth Intl Conference on Data Engineering*, pages 31–39. IEEE, April 1993.

[25] M. Stonebraker, J. Frew, K. Gardels, and Jeff Meredith. "The Sequoia 2000 Benchmark". In *Proc. of Intl Conference on Management of Data*. ACM, 1993.

[26] M. M. Tsangaris and Jeffrey.F. Naughton. "A Stochastic Approach for Clustering in Object Bases". In *Proc. of SIGMOD Conference on Management of Data*, pages 12–21. ACM, 1991.

[27] C.W. Yeh, C.K. Cheng, and T.T. Y. Lin. "A General Purpose Multiple Way Partitioning Algorithm". In *Proc. of 28th ACM/IEEE Design Automation Conference*, pages 421–426, 1991.