Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

A link-based storage scheme for efficient aggregate query processing on clustered road networks ☆

Engin Demir [a], Cevdet Aykanat [a,*], B. Barla Cambazoglu [b]

[a] Computer Engineering Department, Bilkent University, Ankara, Turkey
[b] Yahoo! Research, Barcelona, Spain

ARTICLE INFO

Article history:
Received 4 December 2007
Received in revised form
20 October 2008
Accepted 18 March 2009
Recommended by: N. Koudas

Keywords:
Storage management
Spatial databases and GIS
Road networks
Link-based storage
Clustering
Hypergraphs

ABSTRACT

The need to have efficient storage schemes for spatial networks is apparent when the volume of query processing in some road networks (e.g., the navigation systems) is considered. Specifically, under the assumption that the road network is stored in a central server, the adjacent data elements in the network must be clustered on the disk in such a way that the number of disk page accesses is kept minimal during the processing of network queries. In this work, we introduce the link-based storage scheme for clustered road networks and compare it with the previously proposed junction-based storage scheme. In order to investigate the performance of aggregate network queries in clustered road networks, we extend our recently proposed clustering hypergraph model from junction-based storage to link-based storage. We propose techniques for additional storage savings in bidirectional networks that make the link-based storage scheme even more preferable in terms of the storage efficiency. We evaluate the performance of our link-based storage scheme against the junction-based storage scheme both theoretically and empirically. The results of the experiments conducted on a wide range of road network datasets show that the link-based storage scheme is preferable in terms of both storage and query processing efficiency.

1. Introduction

1.1. Motivation

An important issue involved in large-scale spatial network database design is storage modeling, which directly affects the performance of query processing on spatial network data. Spatial networks, which include network elements such as data nodes and their pairwise connections, are generally represented as directed graphs, where vertices correspond to nodes and edges correspond to connections between the nodes. In this work, without loss of generality, we focus on road networks, a typical type of spatial networks. A road network is represented as a two-tuple $(\mathscr{T}, \mathscr{L})$, where $\mathscr{T}$ and $\mathscr{L}$, respectively, indicate the junctions and the road segments (links) between pairs of junctions.

In road networks, search queries form a major portion of the overall cost of daily queries since these networks have static topologies and hence the maintenance queries are rare. Basic search queries include aggregate network queries, i.e., route evaluation and path computation queries, which are processed to derive an aggregate property over the network elements. In processing aggregate network queries, a vast amount of data must be iteratively accessed and retrieved from the disk to the memory. Concurrently accessing the data of the connected

☆ This work is partially supported by The Scientific and Technological Research Council of Turkey under Grant EEEAG-106E069.

* Corresponding author. Tel.: +90 312 2901625; fax: +90 312 2664047.
E-mail addresses: endemir@cs.bilkent.edu.tr (E. Demir), aykanat@cs.bilkent.edu.tr (C. Aykanat), barla@yahoo-inc.com (B.B. Cambazoglu).

0306-4379/$ - see front matter © 2009 Published by Elsevier B.V.
doi:10.1016/j.is.2009.03.005

elements is expected to decrease the disk access cost of the queries.

The disk access cost in large databases is higher than the cost of in-memory computations even in multi-dimensional data processing. If the access frequencies of the network elements can be modeled from past query logs, storing frequently and concurrently accessed data in the same disk pages can decrease the total disk access cost in query processing. This can be achieved by data clustering, with an upper bound (equal to the disk page size) on individual cluster sizes. For large networks, this type of clustering can yield data allocations that ensure good performance in query processing. The performance may be maintained by periodically reclustering the data based on the access statistics available in the past query logs.

In the literature, for efficient query processing in road networks, extensive studies have been carried out on indexing [17,35,21–23] and data allocation schemes[25,33,13]. Efficient storage schemes should also be adopted to increase the query performance along with efficient data allocation schemes and index structures. However, so far, disk storage schemes are not explored separately from indexing.

1.2. Related work

There are a few works that study the disk-based storage schemes for road networks. In the storage scheme of [16], links of the network are stored in a separate link table. The link table is clustered in disk pages such that pages store the links of which origin nodes are closely located. This approach is based on spatial locality, and clustering does not utilize the connectivity information.

In the following studies, the importance of connectivity information in networks is realized, and graph clustering models [25,33] are proposed to partition the data into disk pages. In [25], the authors propose the junction-based storage scheme, in which each record corresponds to a junction together with its connectivity information in the network. They evaluate their graph clustering model for the junction-based storage scheme by both uniform access frequencies and frequencies extracted from the past query logs, yielding better performance results. In [33], in clustering the network, the minimum number of disk pages is achieved based on the assumption that records have fixed size. The graph clustering models for the junction-based storage scheme are used in the recent spatial query processing and clustering papers [18,34,35,1].

Recently, in [13], we showed that graph clustering models do not correctly capture the disk access cost of aggregate network operations. We proposed a clustering hypergraph model that captures this cost correctly for the junction-based storage scheme. In this model, records are clustered in disk pages by hypergraph partitioning, where the partitioning objective corresponds to minimizing the disk access cost of aggregate network operations in network queries.

1.3. Contributions

In this work, our contributions are fivefold. First, we introduce the link-based storage scheme. In this storage scheme, each record stores the data associated with a link together with the link's connectivity information. Second, we introduce a clustering hypergraph model for the link-based storage scheme to partition the network data to disk pages. Third, we present a detailed comparative analysis on the properties of the junction- and link-based storage schemes and show that the link-based storage scheme is more amenable to clustering. Fourth, we introduce storage enhancements for bidirectional networks. We show that the link-based storage scheme is more amenable to our enhancements than the junction-based storage scheme and results in better data allocation for processing aggregate network queries. Finally, extensive experimental comparisons are carried out on the effects of page size, buffer size, path length, record size, and dataset size for the junction- and link-based storage schemes. Each parameter is explored for both storage schemes, and relative improvements are observed on real-life datasets with synthetic queries. According to the experimental results, the link-based storage scheme can be a good alternative to the widely used junction-based storage scheme.

The rest of this paper is organized as follows: Section 2 presents some background material. In Section 3, the link-based storage scheme and its advantages over the junction-based storage scheme are discussed. Section 4 presents our clustering hypergraph model for the link-based storage scheme. Section 5 overviews the experimental framework and presents the experimental results. Finally, we conclude the paper in Section 6.

2. Preliminaries

2.1. Hypergraph partitioning

The proposed clustering model heavily relies on hypergraph partitioning. Here, we provide a brief description of hypergraphs and hypergraph partitioning. A hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{N})$ consists of a set of vertices $\mathscr{V}$ and a set of nets $\mathscr{N}$ [5]. Each net $n_j \in \mathscr{N}$ connects a subset of vertices in $\mathscr{V}$, which are referred to as the pins of n_j and denoted as Pins(n_j). The size of a net n_j is the number of vertices connected by n_j, i.e., $|n_j| = |\text{Pins}(n_j)|$. The size of a hypergraph $\mathscr{H}$ is defined as the total number of its pins, i.e., $|\mathscr{H}| = \sum_{n_j \in \mathscr{N}} (|n_j|)$. Each vertex v_i has a weight $w(v_i)$, and each net n_j has a cost $c(n_j)$.

$\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_K\}$ is a K-way vertex partition if each part $\mathscr{V}_k$ is non-empty, parts are pairwise disjoint, and the union of parts gives $\mathscr{V}$. In a given K-way vertex partition Π, a net is said to connect a part if it has at least one pin in that part. The connectivity set $\Lambda(n_j)$ of a net n_j is the set of parts connected by n_j. The connectivity $\lambda(n_j) = |\Lambda(n_j)|$ of a net n_j is equal to the number of parts connected by n_j. If $\lambda(n_j) = 1$, then n_j is an internal net. If $\lambda(n_j) > 1$, then n_j is said to be cut.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

In K-way hypergraph partitioning, the partitioning objective is to minimize a cutsize metric defined over the cut nets. In the literature, a number of cutsize metrics are employed. In connectivity-1 metric, which is widely used in VLSI layout design [2,12] and in scientific computing [3,10,27,28], each net n_j contributes $c(n_j)(\lambda(n_j)-1)$ to the cutsize of a partition Π. That is,

$$\text{Cutsize}(\Pi) = \sum_{n_j \in \mathcal{N}} c(n_j)(\lambda(n_j)-1). \quad (1)$$

The partitioning constraint is to maintain an upper bound on the part weights, i.e., $W_k \leqslant W_{\max}$, for each $k = 1,\ldots,K$, where $W_k = \sum_{v_i \in \mathcal{V}_k} w(v_i)$ denotes the weight of part $\mathcal{V}_k$ and $W_{\max}$ denotes the maximum allowed part weight.

The multi-level framework [8] has been successfully adopted in hypergraph partitioning leading to successful hypergraph partitioning tools hMeTiS [19] and PaToH [11]. In multi-level hypergraph partitioning, the original hypergraph is coarsened into a smaller hypergraph after a series of coarsening levels. At each coarsening level, highly coherent vertices are grouped into supervertices by using various matching heuristics. After the partitioning of the coarsest hypergraph, the generated coarse hypergraphs are uncoarsened back to the original, flat hypergraph. At each uncoarsening level, a refinement heuristic (e.g., FM [14] or KL [20]) is applied to minimize the cutsize while maintaining the partitioning constraint.

Although direct K-way hypergraph partitioning [4] is feasible, the Recursive Bipartitioning (RB) paradigm is widely used in K-way hypergraph partitioning and known to be amenable to produce good solution qualities. This paradigm is especially suitable for partitioning hypergraphs when K is not known in advance. In the RB paradigm, first, a two-way partition of the hypergraph is obtained. Then, each part of the bipartition is further bipartitioned in a recursive manner until the desired number K of parts is obtained or part weights drop below a given maximum allowed part weight, $W_{\max}$. In RB-based hypergraph partitioning, the cut-net splitting scheme [10] is adopted to capture the connectivity-1 cutsize metric given in Eq. (1).

2.2. Aggregate network queries in road networks

Route evaluation and path computation queries are shown to be highly frequent in intelligent transportation systems [24]. In route evaluation queries, a prespecified path is traversed to compute an objective function (e.g., the total travel time). In path computation queries, a path which satisfies a given objective function (e.g., the shortest path in terms of travel time) is determined. These two types of queries are named as aggregate network queries as they depend on the evaluation of a number of nodes at a time.

There are two network operations specific to aggregate queries: Get-a-Successor $GaS(t_i,t_j)$ operation retrieves the network element t_j among the successors of t_i and Get-Successors $GSs(t_i)$ operation retrieves all successor elements of t_i. *GaS* operations are used in route evaluation queries, where a *Find* operation is followed by a sequence of *GaS* operations. Here, the *Find* operation returns the given junction from the memory if it resides in the buffer, otherwise retrieves this junction from the secondary storage using an index. *GSs* operations are used in path computation queries, where a sequence of *Find* and *GSs* operation pairs is performed.

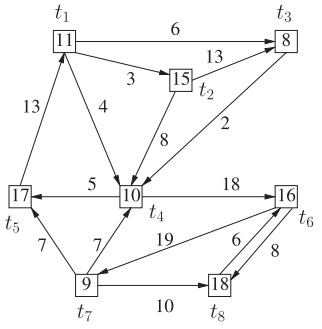

Fig. 1. A sample road network.

Fig. 1 illustrates a sample network with 8 junctions and 15 links, where squares represent the junctions and directed edges represent the links. In the figure, the access frequencies of *GaS* and *GSs* operations, respectively, are given on the directed edges and inside the squares. These values indicate the number of operations performed on the corresponding network elements. Typically, distribution of queries over the network elements is not uniform, and individual access frequencies of the network elements are different. Hence, if the past query logs are available, they can be utilized to estimate the access frequencies of the network elements that will be retrieved by the future queries.

2.3. Junction-based storage scheme

A frequently used approach for storing a road network in the secondary storage is to use the adjacency list data structure, where a record is allocated for each junction of the network. Each record r_i stores the data associated with junction t_i and its connectivity information including the predecessor and successor lists. The data associated with junction t_i contains the coordinate of junction t_i and its attributes. The predecessor list $Pre(t_i)$ denotes the list of incoming links of t_i, whereas the successor list $Succ(t_i)$ denotes the list of outgoing links of t_i. Each element in the predecessor list stores the coordinates of the source junction t_h of an incoming link ℓ_{hi}. The predecessor lists are used in maintenance operations to update the successor lists. In the successor list, each element stores the coordinates of the destination junction t_j of an outgoing link ℓ_{ij} as well as the attributes of ℓ_{ij}. The record sizes are not fixed because of the variation in the predecessor and successor list sizes. If all links of a junction t_i are bidirectional, a storage saving can be achieved since the predecessor and successor lists of t_i contain exactly the same set of junctions. Hence, it suffices to store only the successor list of t_i.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

2.4. Data allocation problem in road networks

The record-to-page allocation problem that we focus on can be defined as follows: given a road network and data access frequencies extracted from the past query logs, allocate a set of data records $\mathscr{R} = \{r_1, r_2, \ldots\}$ to a set of disk pages $\mathscr{P} = \{\mathscr{P}_1, \mathscr{P}_2, \ldots\}$ such that the expected disk access cost is minimized as much as possible while the number of allocated disk pages is kept reasonable. Typically, allocation of data to disk pages can be modeled as a clustering problem, where the clustering objective is to try to store the records that are likely to be concurrently accessed in the same pages. This way, efficiency in query processing can be achieved since the records relevant to the query can be fetched with fewer disk accesses.

2.5. Clustering hypergraph model for the junction-based storage scheme

In our earlier study [13], we proposed a clustering hypergraph model for the junction-based storage scheme. The proposed model is shown to eliminate the flaws of the clustering graph model [25,33] and to yield effective results in minimizing the number of disk page accesses. Here, we briefly summarize this model.

For a given road network, a clustering hypergraph is created, where a vertex exists for each record of the junction-based storage scheme. Each vertex has a weight denoting the size of the corresponding record. The set of $GaS(t_i, t_j)$ and $GaS(t_j, t_i)$ operations invoked between junctions t_i and t_j is modeled as a two-pin net n_{ij}. The net n_{ij} connects the pair of vertices that correspond to t_i and t_j, and it is associated with a cost which is equal to the total number of $GaS(t_i, t_j)$ and $GaS(t_j, t_i)$ operations. The set of $GSs(t_i)$ operations invoked from a junction t_i is modeled by a multi-pin net n_i. The net n_i connects the vertices that correspond to the junctions in the successor list of t_i together with the vertex corresponding to t_i, and it is associated with a cost which is equal to the total number of $GSs(t_i)$ operations.

After representing the network as a clustering hypergraph, we partition the hypergraph with the disk page size being the upper bound on part weights. A K-way partition of this hypergraph is decoded as assigning the set of records corresponding to the vertices in each vertex part to a distinct page of the K-pages to be allocated for the road network. The partitioning constraint corresponds to enforcing the page size limit on the record-to-page allocation. As shown in [13], the partitioning objective corresponds to minimizing the total number of disk accesses due to *GaS* and *GSs* operations under the single-page buffer assumption.

In [13], we proposed two RB schemes, namely RB1 and RB2 for partitioning the clustering hypergraph, since the number of parts is not known in advance. RB1 and RB2 are based on different bipartitioning constraints. The constraint in RB1 is to obtain nearly equal part weights, whereas the constraint in RB2 is to obtain a bipartition such that one of the part weights is nearly a multiple of page size. After the RB-based partitioning, we pack lightly loaded parts to decrease the number of pages. The algorithm utilized for page packing is based on the best-fit heuristic used in solving the bin-packing problem. The RB2 scheme is found to benefit more from this packing process since it generates a large number of lightly loaded parts/pages. Experimental results show that RB2 performs slightly better than RB1.

3. Link-based storage scheme

3.1. Definition

In the proposed link-based storage scheme, a record is allocated for each link of the network. Each record r_{ij} stores the data associated with link ℓ_{ij} and its connectivity information. The data associated with a link ℓ_{ij} typically contain the coordinates of junctions t_i and t_j, attributes of the destination junction t_j and attributes of ℓ_{ij}. The connectivity information includes the predecessor and successor lists. The predecessor list $Pre(\ell_{ij})$ includes the set of incoming links of the source junction t_i of ℓ_{ij}, whereas the successor list $Succ(\ell_{ij})$ includes the set of outgoing links of the destination junction t_j of ℓ_{ij}. Each element in the predecessor list of a link ℓ_{ij} stores the coordinates of the source junction t_h of an incoming link ℓ_{hi}, whereas each element in the successor list stores the coordinates of the destination junction t_k of an outgoing link ℓ_{jk}.

In this scheme, storage savings can be achieved if the network contains bidirectional links where the link attributes are the same for both directions. For example, if $\ell_{ij}, \ell_{ji} \in \mathscr{L}$, the information in records r_{ij} and r_{ji} can be stored as a single record, where the predecessor and successor lists are updated accordingly. Further savings can be achieved if all links of both junctions of a bidirectional link are also bidirectional. In that case, the predecessor and successor lists of both ℓ_{ij} and ℓ_{ji} can be stored only once since the predecessor list of ℓ_{ij} corresponds to the successor list of ℓ_{ji} and vice versa.

3.2. Comparison of storage schemes

In practice, the storage size of the link attributes is greater than that of the junction attributes, and the number of links is greater than the number of junctions. Depending on these network-specific parameters, one of the two storage schemes may be favorable in terms of the total storage size and/or the average record size. The role of average record size in the disk access cost of network queries can be explained as follows. For a given query distribution, the sum of the frequencies of the *GSs* operations to be invoked from the outgoing links of junction t_j in the link-based storage scheme is equal to the frequency of the *GSs* operations to be invoked from t_j in the junction-based storage scheme. Hence, in processing a query, the number of records to be retrieved in both storage schemes is the same. Since smaller average record size enables clustering more records to a page, the query overhead is expected to decrease with decreasing average record size. Below, we provide a detailed comparative

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

analysis of the storage schemes in terms of both the total storage size and average record size.

The total storage sizes S_T and S_L of the junction- and link-based storage schemes can be computed as

$$S_T = \sum_{t\in\mathscr{T}}(C_{id} + C_T + |Pre(t)|C_{id} + |Succ(t)|(C_{id} + C_L)) = |\mathscr{T}|(C_{id} + C_T) + |\mathscr{L}|(2C_{id} + C_L) \quad (2)$$

and

$$S_L = \sum_{\ell\in\mathscr{L}}(2C_{id} + C_L + C_T + |Pre(\ell)|C_{id} + |Succ(\ell)|C_{id}) = |\mathscr{L}|(2C_{id} + C_L + C_T) + C_{id}\sum_{\ell\in\mathscr{L}}(|Pre(\ell)| + |Succ(\ell)|), \quad (3)$$

where C_{id} denotes the storage size of junction coordinates. C_T and C_L refer to the fixed storage size of junction and link attributes, respectively. The difference between the total storage sizes of the two schemes is

$$S_L - S_T = C_{id}\sum_{\ell\in\mathscr{L}}(|Pre(\ell)| + |Succ(\ell)|) + |\mathscr{L}|C_T - |\mathscr{T}|(C_{id} + C_T) = C_T(|\mathscr{L}| - |\mathscr{T}|) + C_{id}\left(\sum_{\ell\in\mathscr{L}}(|Pre(\ell)| + |Succ(\ell)|) - |\mathscr{T}|\right). \quad (4)$$

In a typical road network, the number of links is greater than the number of junctions (i.e., $|\mathscr{L}| > |\mathscr{T}|$), and each link has at least one predecessor or successor (i.e., $|Pre(\ell)| + |Succ(\ell)| \geqslant 1$ for each ℓ). Hence, both terms in (4) are always positive. As a result, the link-based storage scheme requires more disk space than the junction-based storage scheme.

The average record sizes s_T and s_L of the junction- and link-based storage schemes can be computed as follows under the simplifying assumption that the number of incoming and outgoing links for each junction are both equal to $d_{avg} = |\mathscr{L}|/|\mathscr{T}|$. Under this assumption, S_T remains the same while S_L and $S_L - S_T$, respectively, become

$$S_L = |\mathscr{L}|(2C_{id} + C_L + C_T) + 2C_{id}|\mathscr{L}|d_{avg} \quad (5)$$

and

$$S_L - S_T = C_T(|\mathscr{L}| - |\mathscr{T}|) + C_{id}(2|\mathscr{L}|d_{avg} - |\mathscr{T}|). \quad (6)$$

Hence, the average record sizes are

$$s_T = \frac{S_T}{|\mathscr{T}|} = C_{id} + C_T + d_{avg}(2C_{id} + C_L) \quad (7)$$

and

$$s_L = \frac{S_L}{|\mathscr{L}|} = 2C_{id} + C_L + C_T + 2C_{id}d_{avg}. \quad (8)$$

The difference between the average record sizes of the two schemes is

$$s_T - s_L = C_L(d_{avg} - 1) - C_{id}. \quad (9)$$

In a typical road network, $d_{avg} > 1$ and $C_L > C_{id}$. Hence, the average record size in the link-based storage scheme is always smaller than that of the junction-based storage scheme under the given simplifying assumption. As seen from this comparative analysis, although the link-based storage scheme requires more disk space, its average record size is likely to be smaller. Thus, the link-based storage scheme can be expected to perform better than the junction-based storage scheme in terms of disk access cost.

In bidirectional networks, the storage savings described in Sections 2.3 and 3.1 are expected to increase the efficiency of both storage schemes. The link-based storage scheme is expected to benefit more from the storage savings compared to the junction-based storage scheme since, in the link-based storage scheme, we combine the records storing the two directional links between two junctions into a single record and hence halve the number of records. The total storage size decreases for both schemes as shown below:

$$S_T^b = |\mathscr{T}|(C_{id} + C_T) + |\mathscr{L}|(C_{id} + C_L) \quad (10)$$

and

$$S_L^b = \frac{|\mathscr{L}|}{2}(2C_{id} + C_L + 2C_T) + 2C_{id}|\mathscr{L}|(d_{avg} - 1). \quad (11)$$

Note that (11) is derived by using the simplifying assumption mentioned earlier. The difference between the total storage sizes of the two schemes becomes

$$S_L^b - S_T^b = C_T(|\mathscr{L}| - |\mathscr{T}|) + C_{id}(2|\mathscr{L}|(d_{avg} - 1) - |\mathscr{T}|) - C_L\frac{|\mathscr{L}|}{2}. \quad (12)$$

The comparison of (6) and (12) shows that the total storage size difference between the two schemes decreases in favor of the link-based scheme by $|\mathscr{L}|(2C_{id} + C_L/2)$. As seen in (12), the link-based scheme may even require less total disk space than the junction-based scheme for large C_L values.

In bidirectional networks, the average record sizes become

$$s_T^b = \frac{S_T^b}{|\mathscr{T}|} = C_{id} + C_T + d_{avg}(C_{id} + C_L) \quad (13)$$

and

$$s_L^b = \frac{S_L^b}{|\mathscr{L}|/2} = C_L + 2C_T + 2C_{id}(2d_{avg} - 1). \quad (14)$$

The difference between the average record sizes of the two schemes is

$$s_T^b - s_L^b = C_L(d_{avg} - 1) - 3C_{id}(d_{avg} - 1) - C_T. \quad (15)$$

The comparison of (9) and (15) shows that the difference between the average record sizes decreases in bidirectional networks in general. As seen in (15), the average record size of the link-based scheme remains to be less than that of the junction-based scheme for typical networks, where $d_{avg} > 1$, $C_L > 3C_{id}$, and C_T is quite small.

Even though the average record size difference between the two schemes decreases in bidirectional networks, the link-based storage scheme is still more amenable to record clustering compared to the junction-based scheme. We will explain this advantage of the link-based storage scheme over the junction-based storage scheme for a junction t_j with d links all of which are bidirectional. In the junction-based storage scheme, junction t_j will have d successors. We should cluster record r_j storing t_j together with all the records storing the

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

d successor junctions to the same page to avoid the page access cost for the $GSs(t_j)$ operation. That is, these $d+1$ records need to be clustered in the same page. On the other hand, in the link-based storage scheme, each link incident to junction t_j has $d-1$ successors excluding itself. Since r_{ij} stores both ℓ_{ij} and ℓ_{ji}, we should cluster record r_{ij} together with $d-1$ records storing the links incident to t_j other than ℓ_{ji} in the same page to avoid the page access cost for the $GSs(\ell_{ij})$ operation. This holds for all records storing the links incident to junction t_j. Hence, it is sufficient to cluster these d records in the same page to avoid the page access cost for the *GSs* operations invoked from the links incident to junction t_j. Therefore, in the link-based scheme, each *GSs* operation invoked from a junction connected by only bidirectional links can be accomplished by accessing one less record than the junction-based scheme.

Figs. 2(a) and (b), respectively, show the junction- and link-based storage schemes for a sub-network consisting of a junction t_1 connected by four bidirectional links. The data records are shown in the right sides of Fig. 2, where the successors are separated by bold lines and additional successors are appended as dotted parts to represent the neighbor junctions/links not shown in the figure. In the junction-based storage scheme, $d=5$ records (i.e., r_1, r_2, r_3, r_4, and r_5), whereas in the link-based storage scheme $d-1=4$ records (i.e., r_{12}, r_{13}, r_{14}, and r_{15}) need to be clustered in a page to avoid the page access cost for the same number of *GSs* operations. This explains why the link-based storage scheme will be more amenable to clustering than the junction-based storage scheme even when the average record sizes are equal in the two storage schemes.

In addition to the above-mentioned advantages in storage size and clustering, the link-based storage scheme, as in the dual network concept, which was originally proposed in [9] and later used in [31] and [32], expresses the relations between consecutive links along paths and is more suitable to capture the restrictions in networks such as turn restrictions.

3.3. Auxiliary index structures

A hash-based index structure is used to locate the network elements in both storage schemes. Data retrieval (i.e., *Find*, *GaS*, and *GSs*) operations needed for querying network elements in the course of execution are performed by using this hash-based index with an average cost of single disk access for each retrieval request if the network element does not already reside in the memory. The storage cost of a hash-based index is in the order of number of network elements to be indexed. So, the storage cost of the hash-based index is in the order of $|\mathscr{T}|$ and $|\mathscr{L}|$ in the junction- and link-based storage schemes, respectively. That is, the hash-based index, respectively, requires an additional storage of size $S_{\text{hash}} = |\mathscr{T}|C_{\text{ptr}}$ and $S_{\text{hash}} = |\mathscr{L}|C_{\text{ptr}}$ in the junction- and link-based storage schemes, where C_{ptr} denotes the size of a pointer to a data record.

In general, the route evaluation or path computation queries are submitted to the GIS systems as point queries, which contain the (x,y) coordinates of a source and a destination point. It is more likely that the query points lie on the links rather than junctions. Here, we refer to the link that a source point lies on as the source link. In the link-based storage scheme, route evaluation and path computation start from the source link, whereas, in the junction-based storage scheme, they start from the destination junction of the source link. In both cases, the source link must be identified. In our architecture, an R-tree index on links is used as an additional index in both storage schemes and the sole purpose of this index is to locate the source link. The R-tree has two types of nodes: non-leaf nodes and leaf nodes [15]. Non-leaf nodes contain index record entries of the form ⟨MBR, ptr⟩ where MBR is the minimum bounding rectangle of all rectangles stored in the entries of the lower level child node pointed to by ptr. The only minor difference between the R-tree implementation in the two storage schemes is the data stored in the leaf nodes. Each leaf node stores an ⟨MBR, ptr⟩ pair for a link, where MBR corresponds to the minimum bounding rectangle of the link and ptr is the

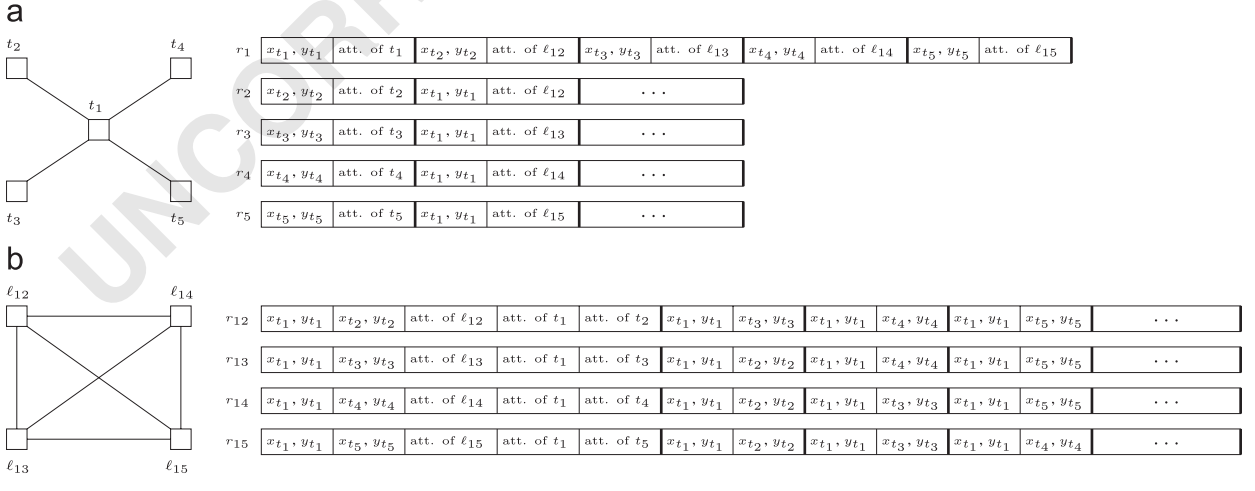

Fig. 2. Storage of records in a bidirectional sub-network using (a) the junction-based and (b) the link-based storage schemes.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

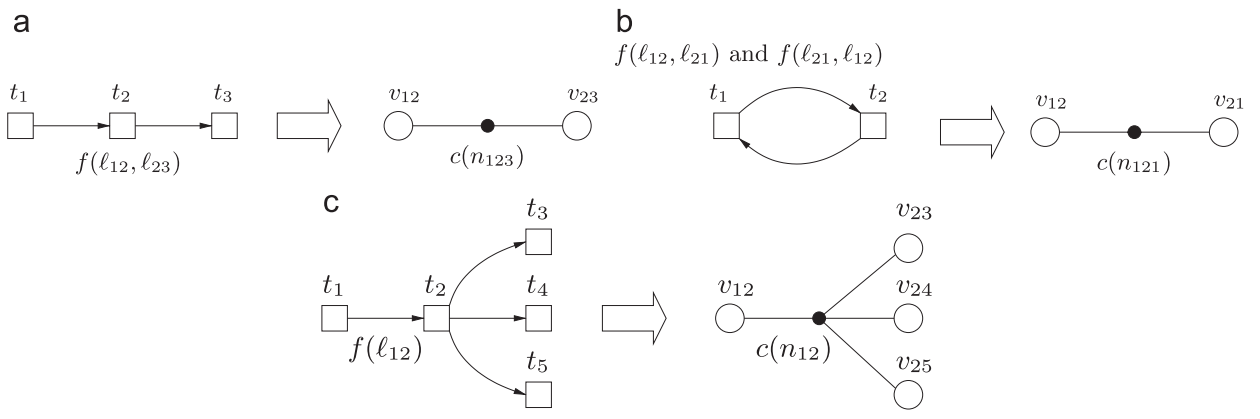

Fig. 3. The clustering hypergraph construction: (a) two-pin net n_{123} for the $GaS(\ell_{12}, \ell_{23})$ operations, (b) coalescence of two two-pin nets incurred by $GaS(\ell_{12}, \ell_{21})$ and $GaS(\ell_{21}, \ell_{12})$ into net n_{121}, (c) multi-pin net n_{12} for the $GSs(\ell_{12})$ operations.

disk page address of the respective record. This record stores data associated with the respective link in the link-based storage scheme, whereas it stores data associated with the endpoint junction of the respective link in the junction-based storage scheme. As the leaf nodes determine the overall storage complexity of the index, both storage schemes require an additional storage of size $S_{\text{Rtree}} = |\mathscr{L}| C_{\text{Rnode}}$ for indexing the links of the network. Here, C_{Rnode} denotes the size of each leaf node.

4. Clustering hypergraph model for the link-based storage scheme

In this section, we present our clustering hypergraph model for the general case of directed networks, where an individual record is stored for each directed link. This model can easily be extended to the bidirectional case, where a single record is stored for each bidirectional link.

4.1. Hypergraph construction

A clustering hypergraph $\mathscr{H}_{\text{L}} = (\mathscr{V}_{\text{L}}, \mathscr{N}_{\text{L}})$ is created to model the network $(\mathscr{T}, \mathscr{L})$. In $\mathscr{H}_{\text{L}}$, a vertex $v_{ij} \in \mathscr{V}_{\text{L}}$ exists for each record $r_{ij} \in \mathscr{R}$ storing the data associated with link $\ell_{ij} \in \mathscr{L}$. The size of a record r_{ij} is assigned as the weight $w(v_{ij})$ of vertex v_{ij}. The net set $\mathscr{N}_{\text{L}}$ is the union of two disjoint sets of nets, $\mathscr{N}_{\text{L}}^{GaS}$ and $\mathscr{N}_{\text{L}}^{GSs}$, which, respectively, encapsulate the disk access costs of *GaS* and *GSs* operations, i.e., $\mathscr{N}_{\text{L}} = \mathscr{N}_{\text{L}}^{GaS} \cup \mathscr{N}_{\text{L}}^{GSs}$.

In $\mathscr{N}_{\text{L}}^{GaS}$, we employ two-pin nets to represent the cost of *GaS* operations. For each incoming and outgoing link pair ℓ_{hi} and ℓ_{ij} of each junction t_i, $GaS(\ell_{hi}, \ell_{ij})$ operations incur a two-pin net n_{hij} with $\text{Pins}(n_{hij}) = \{v_{hi}, v_{ij}\}$. If the source junction of the incoming link is the same as the destination junction of the outgoing link (i.e., $h = j$), the two two-pin nets incurred by the $GaS(\ell_{hi}, \ell_{ij})$ and $GaS(\ell_{ij}, \ell_{hi})$ operations can be coalesced into a single two-pin net with appropriate cost adjustment. Thus, the cost $c(n_{hij})$ associated with net n_{hij} can be written as

$$c(n_{hij}) = \begin{cases} f(\ell_{hi}, \ell_{ij}) & \text{if } \ell_{hi}, \ell_{ij} \in \mathscr{L} \wedge h \neq j, \\ f(\ell_{hi}, \ell_{ij}) + f(\ell_{ij}, \ell_{hi}) & \text{if } \ell_{hi}, \ell_{ij} \in \mathscr{L} \wedge h = j. \end{cases} \quad (16)$$

Here, $f(\ell_{hi}, \ell_{ij})$ denotes the total access frequency of path $\langle \ell_{hi}, \ell_{ij} \rangle$ in $GaS(\ell_{hi}, \ell_{ij})$ operations. Fig. 3(a) shows the two-pin net construction for a pair of neighbor links ℓ_{12} and ℓ_{23}, and Fig. 3(b) shows the two-pin net construction for the cyclic paths $\langle \ell_{12}, \ell_{21} \rangle$ and $\langle \ell_{21}, \ell_{12} \rangle$.

In $\mathscr{N}_{\text{L}}^{GSs}$, we employ multi-pin nets to represent the cost of *GSs* operations. For each link ℓ_{hi} with a destination junction t_i having $d_{\text{out}}(t_i) > 0$ successor(s), $GSs(t_i)$ operations incur a $(d_{\text{out}}(t_i) + 1)$-pin net n_{hi}, which connects vertex v_{hi} and the vertices corresponding to the records of the links that are in the successor list of ℓ_{hi}. That is,

$$\text{Pins}(n_{hi}) = \{v_{hi}\} \cup \{v_{ij} : t_j \in \text{Succ}(t_i)\}. \quad (17)$$

Each net n_{hi} is associated with a cost

$$c(n_{hi}) = f(\ell_{hi}) \quad (18)$$

for capturing the cost of $GSs(\ell_{hi})$ operations. Here, $f(\ell_{hi})$ denotes the total access frequency of link ℓ_{hi} in $GSs(\ell_{hi})$ operations. Fig. 3(c) displays the multi-pin net construction for link ℓ_{12}, which has the successor list $\{\ell_{23}, \ell_{24}, \ell_{25}\}$.

4.2. Clustering hypergraph model

After $\mathscr{H}_{\text{L}} = (\mathscr{V}_{\text{L}}, \mathscr{N}_{\text{L}})$ is constructed, it is partitioned into a number of parts $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots\}$ using the recursive bipartitioning paradigm mentioned in Section 2.1. Here, each part $\mathscr{V}_k \in \Pi$ corresponds to the subset of records to be assigned to disk page $\mathscr{P}_k \in \mathscr{P}$. The partitioning constraint is to enforce the page size as the upper bound on the weight of the vertex parts so that the disk page size is not exceeded in record allocation. The partitioning objective is to minimize the cutsize according to the connectivity-1 metric as defined in Section 2.1. Under the single-page buffer assumption, the connectivity-1 cost incurred to the cutsize by the two-pin cut nets in $\mathscr{N}_{\text{L}}^{GaS}$ and multi-pin cut nets in $\mathscr{N}_{\text{L}}^{GSs}$ exactly corresponds to the disk access cost incurred by the *GaS* operations in the route evaluation queries and *GSs* operations in the path computation queries, respectively. Thus, in our model, minimizing Cutsize (Π) given in (19) exactly minimizes the total number of disk accesses. In the following two paragraphs, we show the correctness of our model for the *GaS* and *GSs* operations:

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

$$\text{Cutsize}(\Pi) = \sum_{n_i \in \mathcal{N}_L^{GaS}} c(n_i)(\lambda(n_i)-1) + \sum_{n_i \in \mathcal{N}_L^{GSs}} c(n_i)(\lambda(n_i)-1)$$
$$= \sum_{n_i \in \mathcal{N}_L} c(n_i)(\lambda(n_i)-1). \quad (19)$$

Consider a partition Π and a two-pin net $n_{hij} \in \mathcal{N}_L^{GaS}$ with $\text{Pins}(n_{hij}) = \{v_{hi}, v_{ij}\}$. If n_{hij} is internal to a part $\mathcal{V}_k$, then records r_{hi} and r_{ij} both reside in page $\mathcal{P}_k$. Since both r_{hi} and r_{ij} can be found in the memory when $\mathcal{P}_k$ is in the page buffer, neither $GaS(\ell_{hi}, \ell_{ij})$ nor $GaS(\ell_{ij}, \ell_{hi})$ operations incur any disk access. Note that $GaS(\ell_{ij}, \ell_{hi})$ operations are possible only if $h = j$. If n_{hij} is a cut net with connectivity set $\Lambda(n_{hij}) = \{\mathcal{V}_k, \mathcal{V}_m\}$, r_{hi} and r_{ij} reside in separate pages $\mathcal{P}_k$ and $\mathcal{P}_m$. Without loss of generality, assume that $r_{hi} \in \mathcal{P}_k$ and $r_{ij} \in \mathcal{P}_m$. In this case, $GaS(\ell_{hi}, \ell_{ij})$ operations incur $f(\ell_{hi}, \ell_{ij})$ disk accesses in order to replace the current page $\mathcal{P}_k$ in the buffer with $\mathcal{P}_m$ in the disk. In a similar manner, $GaS(\ell_{ij}, \ell_{hi})$ operations incur $f(\ell_{ij}, \ell_{hi})$ disk accesses in order to replace the current page $\mathcal{P}_m$ in the buffer with $\mathcal{P}_k$ in the disk. Hence, cut net n_{hij} incurs a cost of $c(n_{hij})$ to the cutsize since $\lambda(n_{hij}) - 1 = 1$.

Now, consider the same partition Π and a multi-pin net $n_{ij} \in \mathcal{N}_T^{GSs}$. If n_{ij} is internal to a part $\mathcal{V}_k$, then record r_{ij} and all records storing the links in the successor list of ℓ_{ij} reside in page $\mathcal{P}_k$. Consequently, $GSs(\ell_{ij})$ operations do not incur any disk access since page $\mathcal{P}_k$ is already in the page buffer. If n_{ij} is a cut net with connectivity set $\Lambda(n_{ij})$, record r_{ij} and the records storing the links in the successor list of ℓ_{ij} are distributed across the pages corresponding to the vertex parts that belong to $\Lambda(n_{ij})$. Without loss of generality, assume that r_{ij} resides in page $\mathcal{P}_k$, where $\mathcal{V}_k$ must be in $\Lambda(n_{ij})$. In this case, each $GSs(\ell_{ij})$ operation incurs $\lambda(n_{ij}) - 1$ page accesses in order to retrieve the records storing the links in the successor list of ℓ_{ij} by fetching the pages corresponding to the vertex parts in $\Lambda(n_{ij}) - \{\mathcal{V}_k\}$. Hence, cut net n_{ij} incurs a cost of $c(n_{ij})(\lambda(n_{ij}) - 1)$ to the cutsize.

Fig. 4 shows the clustering hypergraph $\mathcal{H}_L$ for the network given in Fig. 1 in two parts, which separately show the net sets $\mathcal{N}_L^{GaS}$ and $\mathcal{N}_L^{GSs}$ with the associated costs of *GaS* and *GSs* operations shown in parentheses. In Fig. 4(a), consider two-pin cut net n_{246} with $\text{Pins}(n_{246}) = \{v_{24}, v_{46}\}$ and $\Lambda(n_{246}) = \{\mathcal{V}_1, \mathcal{V}_3\}$. Since v_{24} is in vertex part $\mathcal{V}_1$, page $\mathcal{P}_1$ must be the single page in the buffer when $GSs(\ell_{24})$ operations are invoked. Since v_{46} is in part $\mathcal{V}_2$, $\lambda(n_{246}) - 1 = 2 - 1 = 1$ disk access is required to retrieve record r_{46} into the buffer. Similarly, in Fig. 4(b), consider multi-pin cut net n_{24} with $\text{Pins}(n_{24}) = \{v_{24}, v_{45}, v_{46}\}$ and $\Lambda(n_{24}) = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\}$. Since v_{24} is in vertex part $\mathcal{V}_1$, page $\mathcal{P}_1$ must be the single page in the buffer when $GSs(\ell_{24})$ operations are invoked. Since v_{45} and v_{46} are, respectively, in parts $\mathcal{V}_2$ and $\mathcal{V}_3$, each of the four $GSs(\ell_{24})$ operations will incur $\lambda(n_{24}) - 1 = 3 - 1 = 2$ disk accesses for pages $\mathcal{P}_2$ and $\mathcal{P}_3$ to bring them into the buffer for processing records r_{45} and r_{46}. Note that internal nets do not incur any cost for neither *GaS* nor *GSs* operations since they have a connectivity of 1. The total cost of *GaS* operations, due to the cut nets $\{n_{134}, n_{146}, n_{245}, n_{246}, n_{345}, n_{346}, n_{512}, n_{675}, n_{678}, n_{686}, n_{745}, n_{751}, n_{867}\}$, is $(1+2+1+5+1+1+3+3+9+4+1+7+3) \times (2-1) = 41$ and the total cost of *GSs* operations, due to the cut nets $\{n_{13}, n_{14}, n_{24}, n_{34}, n_{51}, n_{67}, n_{68}, n_{74}, n_{75}, n_{86}\}$, is $3 \times (2-1) + 3 \times (2-1) + 4 \times (3-1) + 2 \times (3-1) + 11 \times (2-1) + 9 \times (2-1) + 7 \times (2-1) + 1 \times (2-1) + 7 \times (2-1) + 4 \times (2-1) = 57$.

The clustering hypergraph models for the junction- and link-based storage schemes are accurate as long as the

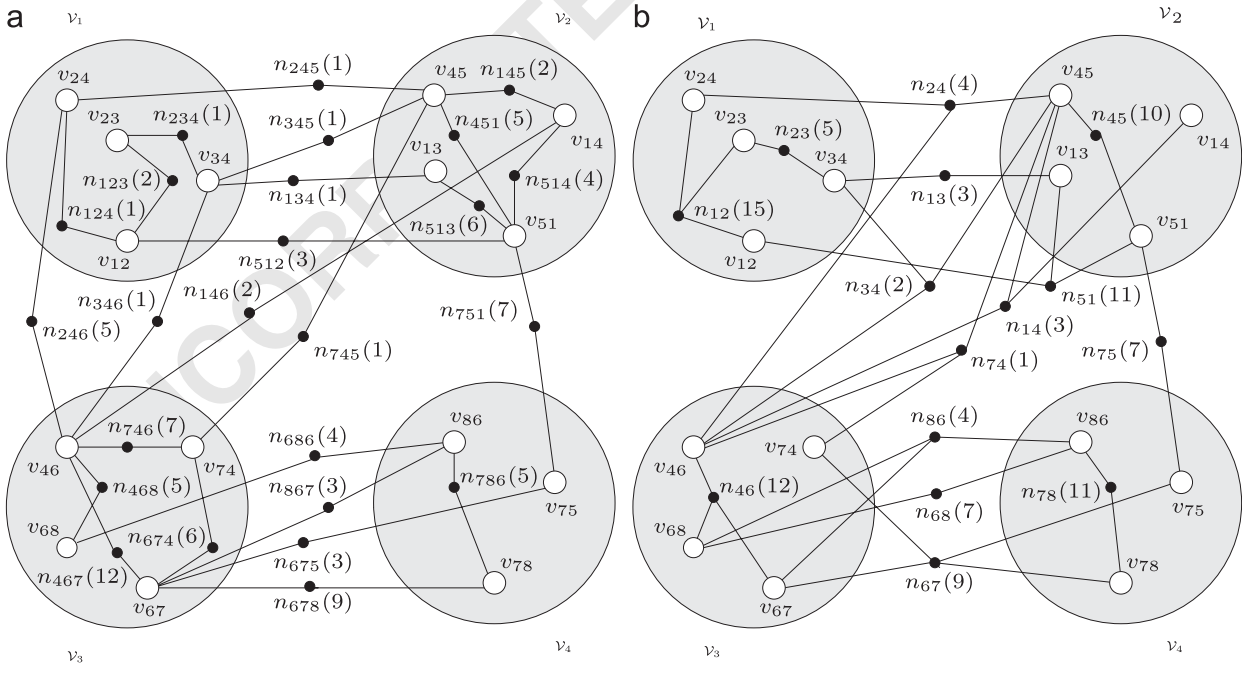

Fig. 4. The clustering hypergraph $\mathcal{H}_L$ for the network given in Fig. 1 and a 4-way vertex partition separately shown on net-induced subhypergraphs (a) $(\mathcal{V}_L, \mathcal{N}_L^{GaS})$ and (b) $(\mathcal{V}_L, \mathcal{N}_L^{GSs})$, respectively, modeling the disk access cost of *GaS* and *GSs* operations.

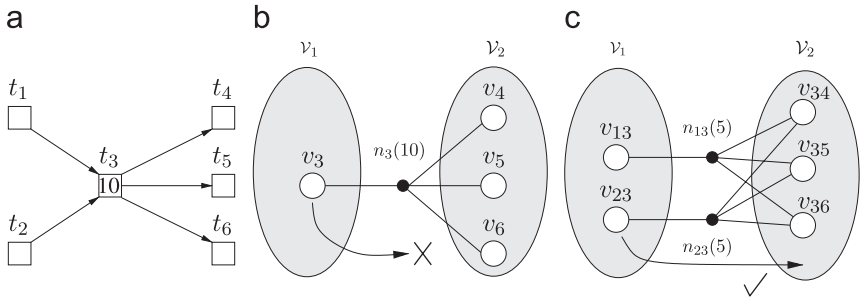

Fig. 5. (a) A sub-network with $GSs(t_3)$, (b) $\mathscr{H}_T$: a four-pin net n_3 for the $GSs(t_3)$ operations with $f(t_3) = 10$, (c) $\mathscr{H}_L$: two four-pin nets n_{13} for the $GSs(\ell_{13})$ operations with $f(\ell_{13}) = 5$ and n_{23} for the $GSs(\ell_{23})$ operations with $f(\ell_{13}) = 5$.

queries in the past query log tend to reappear in the current time window. Disk pages can be periodically reorganized to capture the characteristics of query logs in different time windows. Furthermore, incremental clustering approaches can be adapted to reflect the changes in time.

4.3. Comparison of clustering hypergraph models

The clustering hypergraph models for the junction- and link-based storage schemes are closely related in representing a given road network for solving the record-to-page allocation problem under the respective storage scheme. In both clustering hypergraphs, vertices represent the records, whereas nets represent the aggregate network operations. The set of vertices connected by a net correspond to the set of records concurrently accessed by the respective operation. Vertex weights correspond to records sizes, whereas net costs correspond to the frequency of the respective network operation. In both models, records are clustered into disk pages by partitioning the respective hypergraph, where the partitioning objective corresponds to minimizing the disk access cost of aggregate network operations in network queries. The topological difference between these two hypergraph models stems from the difference between the two storage schemes. Topologically, vertices correspond to junctions and links in the former and latter hypergraph models, respectively.

The sizes of the constructed hypergraphs in our clustering models play an important role in computational and space requirements of the partitioning process. These sizes depend on the topological properties of the network. In the clustering hypergraph $\mathscr{H}_T$ for the junction-based storage scheme, the number $|\mathscr{N}_T^{GaS}|$ of two-pin nets varies between $\lceil|\mathscr{L}|/2\rceil$ and $|\mathscr{L}|$. The number $|\mathscr{N}_T^{GSs}|$ of multi-pin nets is equal to $|\mathscr{T}| - \alpha$, where $\alpha = |\{t_i : d_{out}(t_i) = 0\}|$ is the number of dead ends. The number of pins introduced by multi-pin nets is $|\mathscr{L}| + |\mathscr{T}| - \alpha$. Hence, we have

$$\begin{aligned}&|\mathscr{V}_T| = |\mathscr{T}|,\\ &\lceil|\mathscr{L}|/2\rceil + |\mathscr{T}| - \alpha \leqslant |\mathscr{N}_T| \leqslant |\mathscr{L}| + |\mathscr{T}| - \alpha,\\ &2\lceil 1.5\,|\mathscr{L}|\rceil + |\mathscr{T}| - \alpha \leqslant |\mathscr{H}_T| \leqslant 3|\mathscr{L}| + |\mathscr{T}| - \alpha.\end{aligned} \tag{20}$$

In the clustering hypergraph $\mathscr{H}_L$ for the link-based storage scheme, the number $|\mathscr{N}_L^{GaS}|$ of two-pin nets is $\sum_{t_i\in\mathscr{T}}(d_{in}(t_i) \times d_{out}(t_i)) - \beta$, where $d_{in}(t_i)$ denotes the number of predecessors of t_i and $\beta = |\{\ell_{ij} : \ell_{ij} \in \mathscr{L} \wedge \ell_{ji} \in \mathscr{L}\}|$ is the number of bidirectional links. The number $|\mathscr{N}_L^{GSs}|$ of multi-pin nets is equal to $|\mathscr{L}| - \sum_{t_i\in\mathscr{T}, d_{out}(t_i)=0} d_{in}(t_i)$. The number of pins introduced by multi-pin nets is $\sum_{t_i\in\mathscr{T}, d_{out}(t_i)>0} d_{in}(t_i) \times (d_{out}(t_i) + 1)$. Hence, we have

$$\begin{aligned}&|\mathscr{V}_L| = |\mathscr{L}|,\\ &|\mathscr{N}_L| = \sum_{t_i\in\mathscr{T}}(d_{in}(t_i) \times d_{out}(t_i)) - \beta + |\mathscr{L}| - \sum_{t_i\in\mathscr{T}, d_{out}(t_i)=0} d_{in}(t_i),\\ &|\mathscr{H}_L| = 3\sum_{t_i\in\mathscr{T}}(d_{in}(t_i) \times d_{out}(t_i)) + \sum_{t_i\in\mathscr{T}, d_{out}(t_i)>0} d_{in}(t_i) - 2\beta.\end{aligned} \tag{21}$$

In this work, we claim that the clustering hypergraph model provides more flexibility in partitioning for the link-based storage scheme compared to the junction-based storage scheme. We illustrate this by the following example. Fig. 5(a) shows a sample sub-network $(\mathscr{T}, \mathscr{L})$ with a junction t_3 having two incoming and three outgoing links. Figs. 5(b) and (c) show the net-induced subhypergraphs $(\mathscr{V}_T, \mathscr{N}_T^{GSs})$ and $(\mathscr{V}_L, \mathscr{N}_L^{GSs})$ corresponding to the sub-network given in Fig. 5(a) for the junction- and link-based storage schemes, respectively. Ten *GSs* operations are assumed to be performed on junction t_3, five *GSs* operations for each incoming link of t_3. As seen in the figure, junction t_3 induces only one net n_3 in $\mathscr{H}_T$, whereas the two incoming links ℓ_{13} and ℓ_{23} of t_3 induce nets n_{13} and n_{23} in $\mathscr{H}_L$. Figs. 5(b) and (c) also show 2-way partitions for $\mathscr{H}_T$ and $\mathscr{H}_L$. In this example, if there were no part size constraints, moving vertex v_3 from $\mathscr{V}_1$ to $\mathscr{V}_2$ would remove net n_3 from the cut, thus reducing the cutsize by 10. However, this move may not be feasible due to the maximum part size constraint on $\mathscr{V}_2$. Since the record sizes in the link-based storage scheme are less than those in the junction-based storage scheme as shown in Section 3.2, either v_{13} or v_{23} can move to $\mathscr{V}_2$ without violating the maximum part size constraint, respectively, removing n_{13} or n_{23} from the cut with a saving of 5 on the cutsize. In general, the partitioning of theclustering hypergraph for the link-based storage scheme has a better solution space as there is greater flexibility in moving vertices between parts.

In bidirectional networks, the storage saving in the link-based scheme results in higher improvements in query processing performance compared to the junction-

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

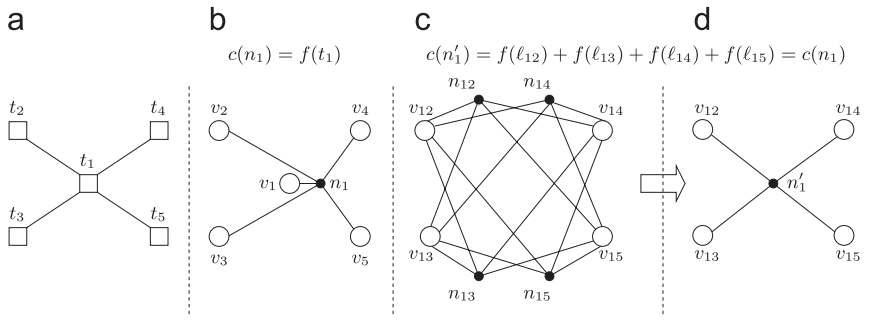

Fig. 6. (a) A bidirectional sub-network with $GSs(t_1)$, (b) $\mathscr{H}_T$: a five-pin net n_1 for the $GSs(t_1)$ operations with $c(n_1) = f(t_1)$, (c) $\mathscr{H}_L$: four identical four-pin nets n_{12}, n_{13}, n_{14}, and n_{15} for $GSs(\ell_{12})$, $GSs(\ell_{13})$, $GSs(\ell_{14})$, and $GSs(\ell_{15})$, respectively, (d) $\mathscr{H}_L$: identical nets n_{12}, n_{13}, n_{14}, and n_{15} coalesced into net n_1' with cost $c(n_1') = c(n_1)$.

based scheme. We provide Fig. 6 to validate this claim. Fig. 6(a) shows a sample sub-network $(\mathscr{T}, \mathscr{L})$ with a junction t_1 having four bidirectional incoming/outgoing links. Figs. 6(b) and (c) show the net-induced subhypergraphs $(\mathscr{V}_T, \mathscr{N}_T^{GSs})$ and $(\mathscr{V}_L, \mathscr{N}_L^{GSs})$ corresponding to the sub-network for the junction- and link-based storage schemes, respectively. Note that the sum of the number of *GSs* operations performed on the incoming links of junction t_1 in the link-based storage scheme is equal to the number of *GSs* operations performed on junction t_1. That is, $f(\ell_{21}) + f(\ell_{31}) + f(\ell_{41}) + f(\ell_{51}) = f(t_1)$.

As seen in Fig. 6(b), in $\mathscr{H}_T$, for the $GSs(t_1)$ operation, there is a five-pin net with $\text{Pins}(n_1) = \{v_1, v_2, v_3, v_4, v_5\}$ and $c(n_1) = f(t_1)$. In the construction of the clustering hypergraph for the link-based storage scheme, two directional links between the same junctions (i.e., ℓ_{ij} and ℓ_{ji}) are represented with a bidirectional link ℓ_{ij}, where $i<j$. Hence, a vertex v_{ij} exists for each record r_{ij} storing link ℓ_{ij}. As seen in Fig. 6(c), $\mathscr{H}_L$ has four four-pin nets n_{12}, n_{13}, n_{14}, and n_{15} to capture the costs of the $GSs(\ell_{21})$, $GSs(\ell_{31})$, $GSs(\ell_{41})$, and $GSs(\ell_{51})$ operations, respectively. Note that these four four-pin nets connect the same set of pins, i.e., $\text{Pins}(n_{12}) = \text{Pins}(n_{13}) = \text{Pins}(n_{14}) = \text{Pins}(n_{15}) = \{v_1, v_2, v_3, v_4, v_5\}$. Such nets, which connect exactly the same set of pins, are called identical nets. Identical nets can be coalesced into a single representative net. The representative net's cost is set to the total cost of all constituting nets. Here, n_{12}, n_{13}, n_{14}, and n_{15} can be coalesced into a representative net n_1' with $\text{Pins}(n_1') = \{v_{12}, v_{13}, v_{14}, v_{15}\}$ and $c(n_1') = c(n_{12}) + c(n_{13}) + c(n_{14}) + c(n_{15})$ as shown in Fig. 6(d). Comparison of Figs. 6(b) and (d) shows that, for *GSs* operations, the clustering hypergraphs for the two storage schemes have the same set of nets with equal costs. However, the size of each net in $\mathscr{H}_L$ is one less than the size of the respective net in $\mathscr{H}_T$. This finding conforms with the fact that, in query processing, each *GSs* operation in the link-based storage scheme accesses one record less compared to the junction-based storage scheme. Thus, the partitioning of $\mathscr{H}_L$ is expected to lead to smaller cutsizes compared to that of $\mathscr{H}_T$ because of smaller net sizes in the link-based storage scheme.

In bidirectional networks, the sizes of the clustering hypergraphs for the two storage schemes become

Table 1
Properties of road network datasets.

Tag	Dataset	Road network		
		$\lvert\mathscr{T}\rvert$	$\lvert\mathscr{L}\rvert$	d_{avg}
D1	California HPN	10 141	28 370	2.80
D2	SanJoaquin	17 444	45 974	2.64
D3	Minnesota7	34 222	92 206	2.69
D4	Sanfrancisco	166 558	426 742	2.56

$$|\mathscr{V}_T| = |\mathscr{T}|,$$
$$|\mathscr{N}_T| = |\mathscr{L}|/2 + |\mathscr{T}|,$$
$$|\mathscr{H}_T| = 2|\mathscr{L}| + |\mathscr{T}| \quad (22)$$

and

$$|\mathscr{V}_L| = |\mathscr{L}|/2,$$
$$|\mathscr{N}_L| = \sum_{t_i \in \mathscr{T}} d(t_i)^2 - |\mathscr{L}| + |\mathscr{T}| - \tau,$$
$$|\mathscr{H}_L| = 2 \sum_{t_i \in \mathscr{T}} d(t_i)^2 - |\mathscr{L}| - \tau, \quad (23)$$

where $d(t_i) = d_{in}(t_i) = d_{out}(t_i)$ and $\tau = |\{t_i : d(t_i) = 1\}|$.

5. Experimental results

5.1. Experimental setup

In order to show the validity of the proposed link-based storage scheme and the clustering model, we have conducted a wide range of experiments on four real-life road network datasets collected from U.S. Tiger/Line [26] (Minnesota7 including 7 counties Anoka, Carver, Dakota, Hennepin, Ramsey, Scott, Washington; Sanfrancisco), U.S. Department of Transportation [29] (California Highway Planning Network), and Brinkhoff's data files [7] (SanJoaquin). We eliminate the self-loops and multi-links in the datasets through a preprocessing step. The properties of the preprocessed datasets are given in Table 1. In the table, d_{avg} refers to the average number of links per junction.

It is important to note that all links in our datasets are bidirectional. This enables the use of the storage savings mentioned in Sections 2.3 and 3.1. In the junction-based storage scheme, we store only the successor list of each junction. In the link-based storage scheme, we combine

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

Table 2
Storage requirements of junction- and link-based storage schemes (in bytes).

Dataset	$C_T = 0$											
	$C_L = 16$				$C_L = 28$				$C_L = 40$			
	S_T^b	S_L^b	s_T^b	s_L^b	S_T^b	S_L^b	s_T^b	s_L^b	S_T^b	S_L^b	s_T^b	s_L^b
D1	607 964	813 624	60.0	57.4	948 404	983 844	93.5	69.4	1 288 844	1 154 064	127.1	81.4
D2	989 256	1 298 856	56.7	56.5	1 540 944	1 574 700	88.3	68.5	2 092 632	1 850 544	120.0	80.5
D3	1 981 008	2 650 184	57.9	57.5	3 087 480	3 203 420	90.2	69.5	4 193 952	3 756 656	122.6	81.5
D4	9 201 072	11 850 952	55.2	55.5	14 321 976	14 411 404	86.0	67.5	19 442 880	16 971 856	116.7	79.5
Averages normalized w.r.t. storage sizes of the junction-based scheme												
	1.00	1.29	1.00	0.99	1.00	1.01	1.00	0.77	1.00	0.87	1.00	0.67

S_T^b and S_L^b denote the total storage sizes for the junction- and link-based storage schemes, respectively. s_T^b and s_L^b denote the average record sizes for the junction- and link-based storage schemes, respectively.

the records storing the two directional links between two junctions into a single record and hence halve the number of records.

In the experiments, 4 bytes are reserved for the coordinates of a junction (i.e., $C_{id} = 4$) and no space is reserved for junction attributes (i.e., $C_T = 0$). We used three different sizes of 16, 28, and 40 bytes for the link attributes (i.e., $C_L = 16$, 28, and 40) in both storage schemes. These attribute sizes, which are even smaller than the recent proposals [30], are selected to show the actual pattern of performance difference between the two storage schemes. This way, we are able to evaluate the effect of the average record size and total storage size on the relative performance of the two storage schemes. Table 2 displays the total storage sizes and the average record sizes for the junction- and link-based storage schemes for each dataset and link attribute size pair. The S_T^b and s_T^b values given in Table 2 are exactly the same with those that can be obtained by substituting the network-specific parameters in Table 1 and the appropriate C_L, C_{id}, and C_T values into (10) and (13). However, the S_L^b and s_L^b values computed by using (11) and (14) differ by 10% (on the average) from the values in Table 2 because of the simplifying assumption used in these equations.

As seen in Table 2, for $C_L = 16$, the average record sizes are almost equal in the two storage schemes, whereas the link-based scheme requires 29% more total storage than the junction-based scheme, on the average. For $C_L = 28$, the total storage sizes are almost equal in the two storage schemes, whereas the average record size of the link-based scheme is 23% less than that of the junction-based scheme, on the average. For $C_L = 40$, both the total storage size and the average record size of the link-based scheme are less than those of the junction-based scheme (on the average 13% and 33%, respectively). Although, in general, the link-based scheme requires more storage than the junction-based scheme, the link-based scheme becomes more favorable than the junction-based scheme for $C_L = 40$. This is mainly due to the fact that the proposed way of handling bidirectional links enables higher storage savings in the link-based scheme compared to the junction-based scheme. Note that the link-based storage scheme has a slightly larger average record size than the junction-based storage scheme for D4 with $C_L = 16$. This does not comply with the analytical evaluation given in Section 3.2 because of the underlying assumption on the average record size.

The clustering hypergraphs for the two storage schemes are constructed as described in [13] and Section 4.1. The vertex weights are set to be equal to the size of the respective records. We generated synthetic query logs for each dataset in order to be able to obtain a cost distribution over the nets of the constructed hypergraphs. For this purpose, a set of source and destination junction pairs, which have a predetermined shortest path length, is generated by slightly modifying the network-based node selection option of Brinkhoff's Network Generator for Moving Objects [6]. Queries that traverse the junctions on the shortest paths between the source and destination junction pairs are added into the query log as route evaluation queries. Queries that seek the shortest paths (using Dijkstra's algorithm) are added into the query log as path computation queries. The number of queries is set to be the same in both route evaluation and path computation queries.

In order to span most network elements in the network and hence to create a hypergraph large enough to represent the network, we adaptively determined a separate query count and a path length for each dataset. According to the path lengths in the queries, we formed three query logs: Q_{short}, Q_{medium}, and Q_{long}. We selected the path lengths and the number of queries in each query log as follows: for Q_{short}, Q_{medium}, and Q_{long}, the path length is, respectively, set to the $\frac{1}{18}$, $\frac{1}{6}$, and $\frac{1}{2}$ of the diameter of the road network. The number of queries in each dataset is picked linearly proportional to the number of junctions. For Q_{short}, Q_{medium}, and Q_{long}, the number of queries is, respectively, set to the $\frac{5}{10}$, $\frac{3}{10}$, and $\frac{1}{10}$ of the number of junctions in the network. Table 3 displays the path length and the number of queries used for each dataset and query log pair. Table 3 also displays the number of *GaS* and *GSs* operations, respectively, invoked by the route evaluation and path computation queries for each dataset and query log pair. Although the total number of queries is set to be equal in both query types, *GSs* operations constitute 97.7% of all operations in the

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

Table 3
Properties of query logs.

Dataset	Q_{short}				Q_{medium}				Q_{long}			
	Path length	Number of			Path length	Number of			Path length	Number of		
		queries	*GaS*	*GSs*		queries	*GaS*	*GSs*		queries	*GaS*	*GSs*
D1	8	5071	30 420	498 478	25	3042	69 943	3 108 062	75	1014	74 022	3 977 814
D2	8	8722	52 230	823 121	25	5233	119 572	4 830 266	76	1744	127 948	9 033 815
D3	26	17 111	405 910	14 583 559	78	10 267	766 892	61 064 163	233	3422	774 053	70 111 055
D4	27	83 279	2 080 352	129 398 112	81	49 967	3 944 006	604 478 026	242	16 656	3 995 328	959 588 281

Table 4
Properties of the clustering hypergraphs for the junction- and link-based storage schemes.

Dataset	$\lvert\mathscr{V}\rvert$	Q_{short}			Q_{medium}			Q_{long}		
		$\lvert\mathscr{N}\rvert$	$\lvert\mathscr{H}\rvert$	$\lvert n\rvert_{avg}$	$\lvert\mathscr{N}\rvert$	$\lvert\mathscr{H}\rvert$	$\lvert n\rvert_{avg}$	$\lvert\mathscr{N}\rvert$	$\lvert\mathscr{H}\rvert$	$\lvert n\rvert_{avg}$
Junction-based storage scheme										
D1	10 141	19 344	56 913	2.9	15 691	49 607	3.2	14 576	47 376	3.3
D2	17 444	30 033	88 575	2.9	25 926	80 359	3.1	23 987	76 449	3.2
D3	34 222	50 970	159 836	3.1	49 439	156 747	3.2	45 128	148 033	3.3
D4	166 558	250 116	760 252	3.0	243 853	747 713	3.1	225 476	710 905	3.2
Link-based storage scheme										
D1	14 185	18 400	45 302	2.5	14 553	37 603	2.6	13 092	34 680	2.6
D2	22 987	28 768	72 090	2.5	22 991	60 526	2.6	20 423	55 367	2.7
D3	46 103	47 080	125 054	2.7	44 659	120 200	2.7	38 581	107 968	2.8
D4	213 371	222 231	576 712	2.6	211 869	555 947	2.6	186 466	504 947	2.7

query workload. This is because of the fact that, for a given source and destination junction pair, the number of *GSs* operations in the path computation queries using Dijkstra's algorithm is much larger than the number of *GaS* operations in the route evaluation queries. Here, we should note that the total net costs in the clustering hypergraphs generated for the two storage schemes are exactly equal for a given query log. This enables a fair comparison between the clustering hypergraph models for the two storage schemes.

Table 4 displays the properties of the clustering hypergraphs used in the experiments for the junction- and link-based storage schemes. In this table, $|n|_{avg} = |\mathscr{H}|/|\mathscr{N}|$ denotes the average net size of a hypergraph. Since the *GaS* and *GSs* operations incurred by the generated queries may not traverse all network elements, the number of nets for each hypergraph is less than the number of all possible nets that can be induced. As mentioned in Section 4.3, bidirectional links lead to identical nets in both storage schemes. These nets are detected and eliminated by a preprocessing step. Table 4 displays the values after this identical net elimination step.

As seen in Table 4, $\mathscr{H}_L$ contains considerably more (25.1% on the average) vertices than $\mathscr{H}_T$. Note that the total number of vertices corresponds to the number of records in a storage scheme. In a bidirectional road network, the junction- and link-based storage schemes, respectively, have $|\mathscr{T}|$ and $|\mathscr{L}|/2$ records, and typically $|\mathscr{T}| < |\mathscr{L}|/2$ since $d_{avg} > 2$. In terms of the number of nets, $\mathscr{H}_L$ contains fewer (10.5% on the average) nets than $\mathscr{H}_T$. This is mainly due to the junctions with degree one, which do not incur multi-pin nets in $\mathscr{H}_L$. In Table 4, the average net size in $\mathscr{H}_L$ is smaller than that of $\mathscr{H}_T$ in accordance with the discussion given in Section 4.3 on multi-pin nets.

As in our earlier proposal for the junction-based storage scheme [13], we use a recursive bipartitioning scheme to partition $\mathscr{H}_L$ into parts (see Section 2.5). Similar to the results in our previous work, the RB2 scheme is experimentally found to give slightly better results than the RB1 scheme. The slightly better performance of RB2 in the link-based storage scheme is again due to the fact that it benefits more from page packing as it generates more lightly loaded pages after partitioning. Hence, in our implementation, we adopt the recursive bipartitioning scheme RB2 and page packing approach described in [13].

For bipartitioning the hypergraphs, we use the state-of-the-art multi-level hypergraph partitioning tool PaToH [10,11]. Partitioning quality for each dataset is evaluated for four different page sizes of $P = 1$, 2, 4, and 8 KB. Due to the randomized nature of the heuristics used in PaToH, the experiments are repeated 100 times, and the average performance results are reported in the following figures and tables.

The running time of the hypergraph partitioning tool PaToH is $O(\log|\mathscr{V}|\sum_{n_j\in\mathscr{N}}|n_j|^2)$ at each bisection step of the RB2 scheme, where $\mathscr{V}$ and $\mathscr{N}$ denote the vertex and net

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

sets of the remaining hypergraph at that bisection step (see the net splitting process used in RB in [11]). In terms of network parameters, the running time of the first bisection step is $O(d_{avg}^2|\mathscr{T}|\log|\mathscr{T}|)$ and $O(d_{avg}^2|\mathscr{L}|\log|\mathscr{L}|)$ for the junction- and link-based storage schemes, respectively, under the simplifying assumption that the number of incoming and outgoing links for each junction are both equal to $d_{avg}=|\mathscr{L}|/|\mathscr{T}|$. Assuming a balanced recursive bisection tree for the RB2 scheme, the overall running time becomes $O(d_{avg}^2|\mathscr{T}|\log|\mathscr{T}|\log K)$ and $O(d_{avg}^2|\mathscr{L}|\log|\mathscr{L}|\log K)$ for the junction- and link-based storage schemes, respectively. However, these are rather loose upper bounds and the partitioning tool PaToH is quite fast while generating high quality results. For example, the overall RB2-based partitioning times for the D1, D2, D3, and D4 datasets are, respectively, 3.2, 5.6, 26.7, and 317.1 s, on the average, including the read/write operations of input/output files. These timings are reported on a PC that is equipped with an Intel Pentium IV 2.6 GHz processor and 2 GB of RAM, and hypergraph

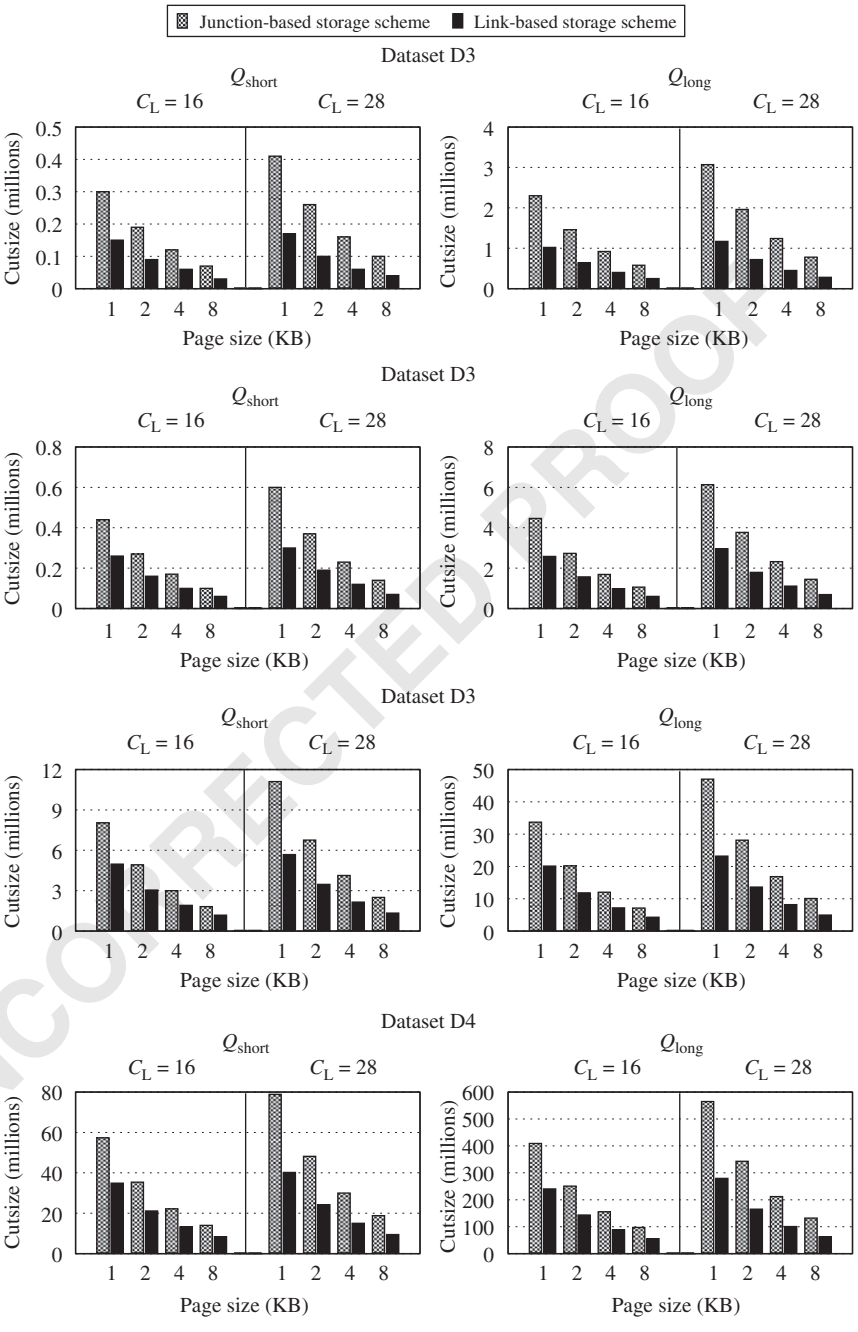

Fig. 7. Partitioning quality of the clustering hypergraph models for the junction- and link-based storage schemes with $C_T = 0$. Cutsize is equal to the number of total disk accesses for the *GaS* and *GSs* operations under the single-page buffer assumption.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

Table 5
Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme.

Query set	P	$C_T = 0$					
		$C_L = 16$		$C_L = 28$		$C_L = 40$	
		K	Cutsize	K	Cutsize	K	Cutsize
Q_{small}	1	−30.9	42.2	−1.6	51.6	12.8	56.4
	2	−31.0	42.6	−1.9	52.1	12.0	56.8
	4	−31.6	42.1	−2.2	52.0	11.6	57.0
	8	−31.2	41.5	−2.2	51.3	11.6	56.4
Q_{medium}	1	−30.8	43.7	−1.5	53.0	13.0	57.4
	2	−31.1	44.4	−1.9	53.7	12.1	58.2
	4	−31.5	44.0	−1.8	53.5	11.6	58.2
	8	−31.3	44.0	−2.0	53.0	11.5	57.8
Q_{large}	1	−30.7	44.8	−1.5	53.7	12.9	58.0
	2	−31.1	45.8	−1.8	54.8	12.1	59.3
	4	−31.1	45.6	−2.1	55.0	11.6	59.7
	8	−31.6	45.7	−2.4	54.9	11.2	59.4

representations for all datasets and parameters fit into the main memory.

Query processing simulations are performed using page buffers with a size of $B = 1$, 2, 4, and 8 pages. Our selection of buffer sizes may look small for a realistic setting; however, they are proportional with the dataset sizes we have. The buffer sizes are selected such that only a small portion of a dataset resides in the memory at any time. The Least Recently Used (LRU) page replacement algorithm is employed as the caching algorithm. Our intention is not to show the effects of buffer replacement policies and cache mechanisms used in the systems. Instead, the experiments are conducted to show that it is still viable to use the clustering approach for increasing number of buffer pages. The synthetic queries used for query log generation are also used in simulations for measuring the total disk access cost.

We evaluate the performance of the clustering hypergraph models for the junction- and link-based storage schemes in two aspects. First, we evaluate the partition quality in terms of cutsize, which corresponds to the total number of disk accesses incurred by *GaS* and *GSs* operations under the single-page buffer assumption. Second, we assess the total number of disk accesses in aggregate network queries through simulations.

5.2. Partitioning quality

Fig. 7 displays the partitioning quality of the clustering hypergraph models for the junction- and link-based storage schemes with the link attribute sizes $C_L = 16$ and 28. These experiments are conducted on the hypergraphs generated using the query logs Q_{short} and Q_{long}. As seen in Fig. 7, in all cases, the link-based storage scheme achieves smaller cutsize values than the junction-based storage scheme. As expected, the cutsize values decrease with increasing page size in both storage schemes, whereas the performance gap between these two schemes does not vary significantly with varying page size.

Table 5 shows the average performance improvements of the clustering hypergraph model for the link-based storage scheme over that for the junction-based storage scheme for all query logs and C_L values. In the table, positive values indicate percent decrease in the K and cutsize values, whereas negative values indicate percent increase in the K values, achieved by the link-based storage scheme compared to the junction-based storage scheme. As seen in Table 5, the two storage schemes achieve almost equal K values for the $C_L = 28$ case. The junction-based storage scheme achieves 31.2% smaller K values for the $C_L = 16$ case, whereas the link-based storage scheme results in 12.0% smaller K values for the $C_L = 40$ case, on the average. These percent differences are approximately equal to the percent differences for the total storage sizes reported in Table 2.

As seen in Table 5, for the $C_L = 28$ case, which incurs almost equal K values for both storage schemes, the link-based storage scheme achieves 53.2% less cutsize values than the junction-based storage scheme, on the average. The relative performance improvement of the link-based storage scheme over the junction-based storage scheme increases to 57.9% when the size of the link attributes increases to $C_L = 40$. These experimental findings are in accordance with our expectations discussed in Section 4.3. However, it is interesting to note that, for $C_L = 16$, although the link-based storage scheme leads to considerably higher K values, it achieves considerably lower cutsize values (43.9% on the average). This can be attributed to the properties of the clustering hypergraphs modeling the networks with bidirectional links.

The effect of query logs on the relative performance between the two storage schemes is also important. As seen in Table 5, for fixed page size and C_L values, the performance gap between the two storage schemes increases as the path length increases in favor of the

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

link-based storage scheme. This finding can be attributed to the increase in the number of *GSs* operations with increasing path length. As mentioned in Section 4.3, the performance difference between the two storage schemes is expected to be higher for *GSs* operations compared to the *GaS* operations.

5.3. Disk access simulations

Figs. 8 and 9 display the relative performance comparisons of the two storage schemes in terms of the number of disk accesses for both route evaluation and path computation queries. The simulation results in these figures are presented for the link attribute sizes $C_L = 16$ and 28 with the varying page and buffer sizes. The query logs Q_{short} and Q_{long} are, respectively, evaluated in Figs. 8 and 9 to show the effect of path length and number of queries in simulations. The average improvements over all datasets are given in Table 6 for all query logs and all C_L values.

As seen in Figs. 8 and 9, the link-based storage scheme outperforms the junction-based storage scheme for almost all simulation cases. In Figs. 8 and 9, for the $C_L = 16$ case with a single-page buffer, the link-based storage

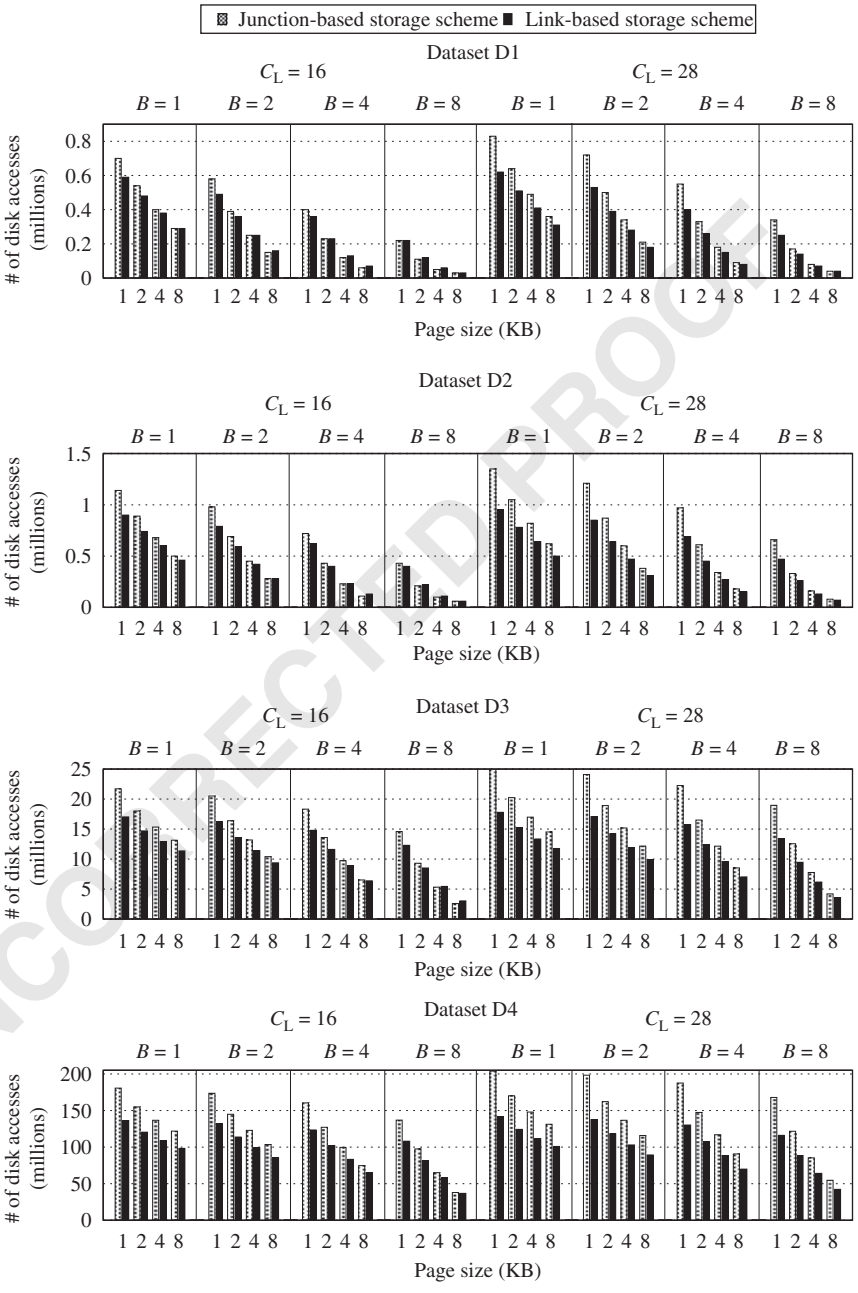

Fig. 8. Disk access comparisons of the two storage schemes in aggregate network query simulations for the Q_{short} query log and $C_T = 0$.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

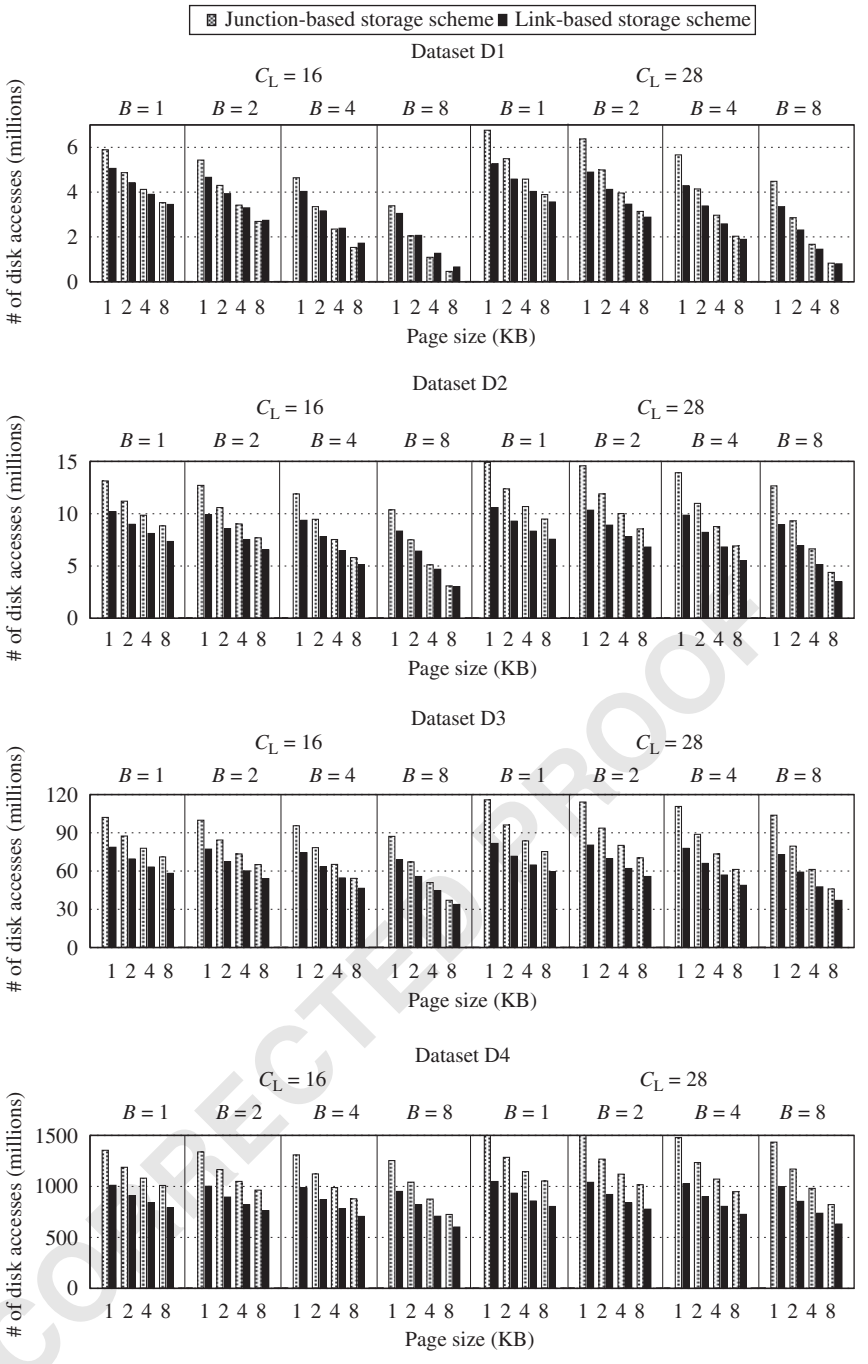

Fig. 9. Disk access comparisons of the two storage schemes in aggregate network query simulations for the Q_{long} query log and $C_T = 0$.

scheme performs better than the junction-based storage scheme in all simulations except for the case of D1 with $P = 8$ and Q_{short}. For the $C_L = 16$ case with larger page and buffer sizes, especially with short queries, the junction-based storage scheme performs slightly better than the link-based storage scheme. This is due to the fact that average record sizes are almost equal, but the total storage of the link-based storage scheme is 29% larger than that of the junction-based storage scheme.

The comparison of the two storage schemes in Table 6 is consistent with the results presented in Table 5. However, the final improvements in the simulations are less than the improvements in actual total costs of *GaS* and *GSs* operations. As seen in Table 5, the average improvement in the total disk access cost of *GaS* and *GSs* operations for a single-page buffer is 43.9% and 53.2% for $C_L = 16$ and for $C_L = 28$, respectively. Nevertheless, in Table 6, the average improvement in the total disk access

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

Table 6
Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme.

B	P	$C_T = 0$								
		$C_L = 16$			$C_L = 28$			$C_L = 40$		
		Q_{short}	Q_{medium}	Q_{long}	Q_{short}	Q_{medium}	Q_{long}	Q_{short}	Q_{medium}	Q_{long}
1	1K	20.7	20.9	21.2	28.5	27.9	27.8	33.4	32.6	32.5
	2K	17.0	17.9	18.2	24.3	23.6	23.6	28.6	27.5	27.4
	4K	13.6	15.3	16.1	21.0	20.4	20.5	25.3	23.6	23.6
	8K	10.2	13.6	14.7	18.3	18.0	18.4	22.6	20.8	20.8
2	1K	19.7	20.6	21.0	29.1	28.2	28.2	34.5	33.1	33.0
	2K	15.0	17.1	17.7	24.8	23.9	23.9	30.1	28.1	28.0
	4K	9.8	13.7	15.0	21.4	20.5	20.6	27.0	24.3	24.2
	8K	4.4	11.1	12.8	17.9	17.8	18.3	24.5	21.6	21.4
4	1K	16.9	19.5	20.3	29.3	28.5	28.4	35.6	33.7	33.4
	2K	10.3	15.2	16.3	24.8	24.2	24.1	31.7	29.1	28.8
	4K	2.7	10.1	12.4	20.8	20.5	20.7	29.2	25.6	25.3
	8K	−4.3	5.4	8.3	16.3	17.2	17.9	26.2	23.1	22.6
8	1K	11.0	17.2	18.6	28.7	28.9	28.8	36.7	34.9	34.3
	2K	2.4	10.8	12.9	23.3	24.5	24.4	33.1	31.0	30.2
	4K	−4.7	1.3	5.9	18.3	20.5	20.6	29.9	28.1	27.2
	8K	−10.7	−10.3	−3.4	13.3	15.9	16.8	24.7	26.2	24.9

Table 7
Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme.

B	P	$C_L = 28$								
		$C_T = 4$			$C_T = 8$			$C_T = 16$		
		Q_{short}	Q_{medium}	Q_{long}	Q_{short}	Q_{medium}	Q_{long}	Q_{short}	Q_{medium}	Q_{long}
1	1K	28.5	29.6	27.7	26.5	26.3	26.3	25.0	25.0	25.1
	2K	24.2	24.4	23.4	22.4	22.2	22.4	20.9	21.1	21.3
	4K	21.0	20.8	20.3	19.0	19.1	19.4	17.3	18.1	18.5
	8K	18.4	18.2	18.2	15.9	16.6	17.3	14.0	15.6	16.3
2	1K	29.1	29.1	28.0	26.7	26.4	26.5	24.8	24.9	25.2
	2K	24.8	23.4	23.7	22.2	22.2	22.5	20.1	20.8	21.2
	4K	21.4	19.8	20.4	17.9	18.8	19.2	15.4	17.4	18.0
	8K	18.1	16.6	18.0	14.0	15.6	16.7	10.6	14.1	15.3
4	1K	29.3	30.4	28.3	25.8	26.2	26.4	23.2	24.4	24.8
	2K	24.8	25.0	23.9	20.7	21.7	22.2	17.5	19.8	20.5
	4K	20.9	20.7	20.4	15.2	17.8	18.5	11.0	15.7	16.7
	8K	16.5	17.4	17.5	10.2	13.5	15.1	4.4	10.8	12.8
8	1K	28.7	31.1	28.6	23.7	25.7	26.0	19.7	23.2	24.0
	2K	23.3	25.5	24.1	17.2	20.7	40.0	12.2	17.7	18.9
	4K	18.5	20.9	20.1	10.6	15.5	35.8	4.1	11.8	13.8
	8K	13.3	16.1	16.3	5.4	8.3	30.9	−1.6	2.7	6.9

cost of aggregate network queries for a single-page buffer is 11.7% and 22.6% for $C_L = 16$ and for $C_L = 28$, respectively. This is mainly due to the additional overhead of *Find* operations incurred by the internal steps of the shortest path algorithm used in path computation queries.

According to Figs. 8 and 9, as expected, increasing page size and increasing buffer size independently decrease the number of disk accesses in the two storage schemes. The performance gap between the storage schemes decreases with increasing P. For $C_L = 16$, there are even cases where the junction-based storage scheme performs better than the link-based storage scheme. This experimental finding can be attributed to the total number of records fetched to the memory for query processing and the total storage

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

size difference between the two schemes. As seen from Table 3, the length of queries is quite small in D1 and D2 for Q_{short}, and hence the hit rate for records in the buffer increases with increasing page and buffer sizes. On the other hand, for $C_L = 16$, the total number of records and the total storage size of the junction-based scheme is smaller than the link-based scheme and thus the junction-based scheme is expected to perform better with large buffers that can store a considerable portion of the dataset.

Recall that C_T and C_L are two important factors that affect the average record size and total storage size. The experimental results reported and discussed so far were obtained for a fixed $C_T = 0$ with varying C_L values of 16, 28, and 40 bytes. In order to represent a study on varying C_T, we also conducted a set of experiments for a fixed $C_L = 28$ with varying C_T values of 4, 8, and 16 bytes. The total disk access simulation results of these experiments are displayed in Table 7. As seen in the table, the link-based scheme performs better than the junction-based scheme for every combination of simulation parameters except for $B = 8$, $P = 8K$, $C_T = 16$ and Q_{short}. The performance gap between the two storage schemes decreases with increasing C_T values, as expected. However, even for the largest C_T value of 16 bytes, the link-based storage scheme, respectively, incurs 14.9%, 17.7%, and 18.7% less disk accesses than the junction-based storage scheme for Q_{short}, Q_{medium}, and Q_{long} query logs, on the average.

6. Concluding remarks

We introduced the link-based storage scheme for efficient aggregate query processing on clustered road networks. In this storage scheme, each record stores the data associated with a link together with the link's connectivity information. We introduced a clustering hypergraph model for the link-based storage scheme to partition the network data to disk pages where data would be periodically reorganized using the past query logs. Our detailed comparative analysis on the properties of the junction- and link-based storage schemes showed that the link-based storage scheme is more amenable to clustering. Moreover, we introduced storage enhancements for bidirectional networks. We showed that the link-based storage scheme is more amenable to our enhancements than the junction-based storage scheme and results in better data allocation for processing aggregate network queries. Extensive experimental comparisons were carried out on the effects of page size, buffer size, path length, record size, and dataset size for the junction- and link-based storage schemes. Experimental results showed that the link-based storage scheme outperforms the widely-used junction-based storage scheme in terms of both storage and query processing efficiency.

Although this work focused on route evaluation and path computation queries, the developed framework can easily be applied to other types of network queries such as dynamic path computation [30], nearest neighbor [1], range search and closest pairs [23]. Some of these types of queries such as variants of nearest neighbors and closest pairs require storing points-of-interests (POIs). The storage of POIs is generally handled separately from the network topology, as the updates on POIs are frequent when compared to the changes in the network topology. To handle such queries, we are also conducting research on embedding POIs into our storage schemes. The storage schemes mentioned in this work are generic representations of networks, and hence any index can be built on top of these storage schemes. Application of the link-based storage scheme in graph topologies may also be beneficial for research problems in other fields.

Acknowledgment

Some experiments were carried out on TR-Grid e-Infrastructure.

References

[1] V.T. Almeida, R.H. Güting, Using Dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases, in: Proceedings of the ACM International Symposium on Applied Computing, 2006, pp. 23–27.
[2] C.J. Alpert, A.B. Kahng, Recent directions in netlist partitioning: a survey, VLSI Journal 19 (1–2) (1995) 1–81.
[3] C. Aykanat, A. Pinar, Ü.V. Çatalyürek, Permuting sparse rectangular matrices into block-diagonal form, SIAM Journal of Scientific Computing 25 (6) (2004) 1860–1879.
[4] C. Aykanat, B.B. Cambazoglu, B. Uçar, Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices, Journal of Parallel and Distributed Computing 68 (5) (2008) 609–625.
[5] C. Berge, Graphs and Hypergraphs, North-Holland Publishing Company, Amsterdam, 1973.
[6] T. Brinkhoff, A framework for generating network-based moving objects, GeoInformatica 6 (2) (2002) 153–180.
[7] T. Brinkhoff, Data files: San Joaquin, 2007. ⟨http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/⟩.
[8] T.N. Bui, C. Jones, A heuristic for reducing fill in sparse matrix factorization, in: Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 445–452.
[9] T. Caldwell, On finding minimum routes in a network with turn penalties, in: Communications of the ACM, 1961, pp. 107–108.
[10] Ü.V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition of parallel sparse-matrix vector multiplication, IEEE Transactions on Parallel and Distributed Systems 10 (7) (1999) 673–693.
[11] Ü.V. Çatalyürek, C. Aykanat, PaToH: partitioning tool for hypergraphs, Technical Report BU-CE-9915, Computer Engineering Department, Bilkent University, 1999 ⟨http://www.cs.bilkent.edu.tr/~aykanat/pargrp/patoh/⟩.
[12] A. Dasdan, C. Aykanat, Two novel multiway circuit partitioning algorithms using relaxed locking, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 16 (2) (1997) 169–178.
[13] E. Demir, C. Aykanat, B.B. Cambazoglu, Clustering spatial networks for aggregate query processing: a hypergraph approach, Information Systems 33 (1) (2008) 1–17.
[14] C.M. Fiduccia, R.M. Mattheyses, A linear-time heuristic for improving network partitions, in: Proceedings of the 19th ACM/IEEE Design Automation Conference, 1982, pp. 175–181.
[15] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1984, pp. 47–57.
[16] Y.-W. Huang, N. Jing, E.A. Rundensteiner, Effective graph clustering for path queries in digital map databases, in: Proceedings of the ACM International Conference on Information and Knowledge Management, 1996, pp. 215–222.
[17] C.S. Jensen, J. Kolar, T.B. Pedersen, I. Timko, Nearest neighbor queries in road networks, in: Proceedings of the ACM International Workshop on Geographic Information Systems, 2003, pp. 1–8.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005

[18] S. Jung, S. Pramanik, An efficient path computation model for hierarchically structured topographical road maps, IEEE Transactions on Knowledge and Data Engineering 14 (5) (2002) 1029–1046.
[19] G. Karypis, V. Kumar, hMeTiS: a hypergraph partitioning package version 1.5.3, Technical Report, University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN, 1998.
[20] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal 49 (1970) 291–307.
[21] M.R. Kolahdouzan, C. Shahabi, Alternative solutions for continuous K nearest neighbor queries in spatial network databases, GeoInformatica 9 (4) (2005) 321–341.
[22] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, S. Teng, On trip planning queries in spatial databases, in: Proceedings of the 9th International Symposium on Spatial and Temporal Databases, 2005, pp. 273–290.
[23] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, Query processing in spatial network databases, in: Proceedings of the International Conference on Very Large Data Bases, 2003, pp. 790–801.
[24] S. Shekhar, A. Kohli, M. Coyle, Path computation algorithms for advanced traveler information systems, in: Proceedings of the IEEE International Conference on Data Engineering, 1993, pp. 31–39.
[25] S. Shekhar, D.R. Liu, A connectivity-based access method for networks and network computation, IEEE Transactions on Knowledge and Data Engineering 9 (1) (1997) 102–117.
[26] Topologically integrated geographic encoding and referencing system (TIGER), 2002 ⟨http://www.census.gov/geo/www/tiger/⟩.
[27] B. Ucar, C. Aykanat, Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies, SIAM Journal of Scientific Computing 25 (6) (2004) 1837–1859.
[28] B. Ucar, C. Aykanat, Revisiting hypergraph models for sparse matrix partitioning, SIAM Review 49 (4) (2007) 595–603.
[29] US Department of Transportation Federal Highway Administration, The National Highway Planning Network, 2004 ⟨http://www.fhwa.dot.gov/planning/nhpn/index.html⟩.
[30] Q. Wang, L. Xu, J. Qui, Research and realization of the optimal path algorithm with complex traffic regulations in GIS, in: Proceedings of the IEEE International Conference on Automation and Logistics, Jinan, China, August 2007, pp. 516–520.
[31] S. Winter, Weighting the path continuation in route planning, in: Proceedings of the 9th ACM International Symposium on Advances in GIS, 2001, pp. 173–176.
[32] S. Winter, Route specifications with a linear dual graph, in: Proceedings of the International Symposium on Advances in Spatial Data Handling, 2002, pp. 329–338.
[33] S.-H. Woo, S.-B. Yang, An improved network clustering method for I/O-efficient query processing, in: Proceedings of the ACM Symposium on GIS, 2000, pp. 62–68.
[34] M.L. Yiu, N. Mamoulis, Clustering objects on a spatial network, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 13–18.
[35] M.L. Yiu, N. Mamoulis, D. Papadias, Aggregate nearest neighbor queries in road networks, IEEE Transactions on Knowledge and Data Engineering 17 (6) (2005) 820–833.

Please cite this article as: E. Demir, et al., A link-based storage scheme for efficient aggregate query processing on clustered road networks, Informat. Systems (2009), doi:10.1016/j.is.2009.03.005