Available online at www.sciencedirect.com
ScienceDirect
Information Systems 33 (2008) 1–17
www.elsevier.com/locate/infosys

Clustering spatial networks for aggregate query processing: A hypergraph approach ☆

Engin Demir, Cevdet Aykanat*, B. Barla Cambazoglu

Computer Engineering Department, Bilkent University, 06800 Bilkent, Ankara, Turkey

Received 9 January 2006; received in revised form 29 September 2006; accepted 3 April 2007
Recommended by N. Koudas

Abstract

In spatial networks, clustering adjacent data to disk pages is highly likely to reduce the number of disk page accesses made by the aggregate network operations during query processing. For this purpose, different techniques based on the clustering graph model are proposed in the literature. In this work, we show that the state-of-the-art clustering graph model is not able to correctly capture the disk access costs of aggregate network operations. Moreover, we propose a novel clustering hypergraph model that correctly captures the disk access costs of these operations. The proposed model aims to minimize the total number of disk page accesses in aggregate network operations. Based on this model, we further propose two adaptive recursive bipartitioning schemes to reduce the number of allocated disk pages while trying to minimize the number of disk page accesses. We evaluate our clustering hypergraph model and recursive bipartitioning schemes on a wide range of road network datasets. The results of the conducted experiments show that the proposed model is quite effective in reducing the number of disk accesses incurred by the network operations.

Keywords: Spatial networks; Clustering; Record-to-page allocation; Hypergraph partitioning

1. Introduction

1.1. Motivation

In the last two decades, numerous conceptual models, spatial access methods, and query processing techniques are proposed [1,2] to overcome the problems faced within the extensive scale of geographic information systems (GIS). The increasing demand on geographic applications made spatial databases quite popular. The research on spatial databases focused on the Euclidean space, where the distances between the objects are determined by the relative positions of the objects in space. However, the operations in spatial networks, where the data has an underlying network topology, do not solely rely on geographical locations. This attracted many researchers from the areas of transportation GIS, network analysis, moving object databases, operations research, artificial intelligence, and robotics.

☆This work is partially supported by The Scientific and Technological Research Council of Turkey under Grant EEEAG-103E028.
*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), berkant@cs.bilkent.edu.tr (B.B. Barla Cambazoglu).

0306-4379/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2007.04.001

In spatial networks, both the topological and geographical properties of the underlying network are important. The topological properties are usually represented as a finite collection of points and links. For example, in road networks, intersections of roads and road segments connecting the intersections are stored. Types of spatial network queries [3–9] include route evaluation, path computation, tour evaluation and location-allocation evaluation.

In practice, spatial network data is too large to fit into the volatile memory, and a considerable portion of the data must be stored on the secondary storage. Consequently, a high number of disk accesses must be performed during the processing of a query in order to cache the disk pages that store the relevant spatial data in the memory. This makes organization of the spatial data over the disk pages particularly important. There are two primary concerns in organizing the data over the disk pages. First, the number of disk accesses should be kept low for time efficiency in query processing. Second, the utilization of disk pages should be increased to reduce the number of pages that the data spans for space efficiency in data storage.

In query processing over spatial networks, the spatial coherency that exists in accessing data leads to a temporal coherency, that is, connected points are accessed almost concurrently. Taking this fact into consideration, it seems reasonable to place the data associated with connected points in the same disk pages so that the data can be fetched to the memory with fewer disk accesses. Furthermore, the recent query logs can be utilized for predicting the future network usage frequencies and hence disk access patterns, resulting in increased efficiency and effectiveness in data organization. This option requires the data over the disk to be periodically reorganized.

Most data reorganization techniques proposed so far are based on data clustering, which aims to group commonly accessed data in the same disk pages. In this work, we propose a highly accurate clustering hypergraph model utilizing query logs for efficient query processing in spatial networks. The proposed model aims to minimize the number of disk page accesses incurred by the network operations while keeping the number of allocated disk pages at a reasonable amount. Throughout the paper, the number of disk page accesses incurred by network operations will be referred to as the disk access cost of these operations.

1.2. Related work

There are many query types [10–13] in spatial databases, but generally the nearest neighbor, range search, spatial join, and closest pair queries are the widely used types of queries. Query types in spatial networks are somewhat different than those in spatial databases as the evaluation of the network queries highly depends on the topological properties of the underlying network. For accelerating the computation of network queries, materialization techniques are used by pre-computing the results and storing them at the expense of additional storage [14,15].

So far, a considerable number of studies have been carried out on spatial databases to design effective storage schemes [16–23] for efficient query processing. These efforts can be categorized into two groups as proximity- and connectivity-based approaches. In the proximity-based approaches [16,19–21], interrelation of data is dependent on spatial proximity. However, query processing in spatial networks mostly involves route evaluation and path computation queries [3,6,24], which require the use of the connectivity information. As the connectivity information cannot be resolved from spatial proximity, the proximity-based approaches are not directly applicable in indexing and querying of spatial networks [25]. In the connectivity-based approaches [17,18,22,23,26], the connectivity relationship is embedded in graph representations of spatial networks. Based on this fact, efficient access methods are proposed for directed network graphs with no cycles [27–30] and with limited cycles [31]. However, as these proposals rely on the total ordering of the graph vertices, they do not accurately model all kinds of spatial networks including road networks.

In the literature, two approaches which fully utilize the connectivity information are proposed: network-traversal clustering (NTC) by Woo and Yang [23] and connectivity-clustered access method (CCAM) by Shekhar and Liu [22]. Both works use graph partitioning for clustering the spatial network. They model the spatial network as an undirected clustering graph, where partitioning the clustering graph induces a clustering of the spatial network into disk pages and the partitioning objective relates to storing concurrently accessed data in the same pages.

Woo and Yang [23] propose the NTC method, which obtains the minimum number of disk pages

based on the assumption that the size of data records is fixed and the disk page size is a multiple of the record size. This is not appropriate for spatial networks since, in most cases, the record sizes vary depending on the connectivity of the network. Moreover, NTC does not utilize the previous query logs and does not incorporate the usage frequencies of the network into the clustering.

Shekhar and Liu [22] propose the CCAM method to cluster the spatial network into disk pages based on the network connectivity using graph partitioning. CCAM focuses on *Get-a-Successor* (*GaS*) operations to retrieve a successor of a junction in route evaluation queries and *Get-Successors* (*GSs*) operations to retrieve all successors of a junction in path computation queries. It correctly captures the disk access cost of the *GaS* operations. In [22], the authors also evaluate dynamic reclustering strategies. Basic database operations (i.e., *Create*, *Find*, *Insert*, and *Delete*) and aggregate network operations (i.e., *GaS* and *GSs*) are evaluated in the clustering graph model. Experimental results show that CCAM performs better than the previous proximity- and connectivity-based methods in reducing the number of disk page accesses.

1.3. Contributions

Both CCAM [22] and NTC [23] methods fail to correctly capture the number of disk page accesses in aggregate network operations on spatial networks. As mentioned in [22], although the clustering graph model accurately captures the disk access cost of *GaS* operations, it cannot correctly capture the disk access cost of *GSs* operations. When *GaS* and *GSs* operations are not uniformly distributed over the network and the *GSs* operations are more frequent than the *GaS* operations, the performance of the clustering graph model degrades.

In this paper, we propose a novel clustering hypergraph model which utilizes the network usage frequencies obtained from previous query logs and enables the correct estimation of the disk access costs of aggregate network operations. Allocation of data to disk pages according to the clustering of the proposed hypergraph model results in a considerable efficiency improvement in spatial query processing when compared with the earlier proposals that are based on the clustering graph model. Our model is able to use the spatial access methods of [22] in order to support network operations on clustered network data. Note that our model can benefit from the dynamic clustering strategies presented in [22].

As a secondary contribution, we introduce two adaptive partitioning schemes based on recursive bipartitioning (RB). These schemes, which are applicable to both the clustering graph and hypergraph models, try to reduce the number of allocated disk pages while trying to minimize the number of disk page accesses during the network operations.

The rest of the paper is organized as follows. Section 2 presents some background material. In Section 3, the clustering graph model and its flaws are discussed. Section 4 presents our clustering hypergraph model. We present our adaptive partitioning schemes in Section 5. Section 6 overviews the experimental setup and presents experimental results. Finally, we conclude the paper in Section 7.

2. Preliminaries

2.1. Spatial networks

A well-known type of spatial networks is the road networks. Throughout the paper, we use road networks to present the terminology of the clustering models. We represent a road network as a two tuple $(\mathscr{T}, \mathscr{L})$, where $\mathscr{T}$ denotes the spatial locations (junctions) and $\mathscr{L}$ denotes the road segments (links) between pairs of junctions. That is, $\ell_{ij} \in \mathscr{L}$ denotes a link from a junction $t_i \in \mathscr{T}$ to a neighbor junction $t_j \in \mathscr{T}$. There are a number of attributes associated with junctions (e.g., locations, turn restrictions) and links (e.g., length, average speed limit, capacity, type, location related information). Road networks are usually represented as directed graphs with the points located in 2D. Constraints on junctions such as turn restrictions can be modeled by introducing dummy nodes to the graph [32–34]. Fig. 1 illustrates a sample network with eight junctions and 14 links. In the figure, the squares represent the junctions and directed lines represent the links.

2.1.1. Storage

In road networks, a record is allocated for each junction of the network. Each record r_i stores the data associated with junction t_i (i.e., its coordinates and attributes) and its connectivity information (i.e., predecessor and successor lists). A predecessor list corresponds to the set of incoming links while a successor list corresponds to the set of outgoing links. Each element in the predecessor list of a junction t_i stores the coordinates of the source

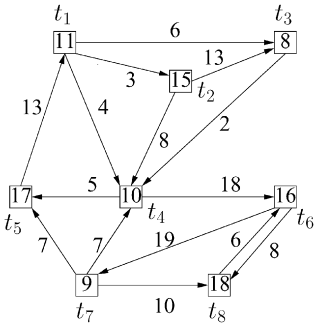

Fig. 1. A sample road network.

junction t_h of the incoming link ℓ_{hi}. Each element in the successor list stores the coordinates of the destination junction t_j of the outgoing link ℓ_{ij} as well as the attributes of ℓ_{ij}. Successor lists are used in network operations, whereas predecessor lists are used in maintenance operations such as *Insert* and *Delete* to update the successor lists. Hence, storing link attributes only in successor lists suffices to evaluate network queries. The record sizes are not fixed since the predecessor and successor list sizes depend on the connectivity of the junctions.

2.1.2. Query processing

In road networks, *Create*, *Find*, *Insert*, *Delete*, *GaS*, and *GSs* operations should be performed efficiently as discussed in [22]. Here, we will provide a brief overview of these operations. The *Create* operation creates a network from a given set of junctions and links. The *Find*, *Insert*, and *Delete* operations perform the actions implied by their names on records of junctions. It should be noted that an auxiliary index structure (e.g., a B^+ tree with Z-ordering) is necessary to retrieve the records efficiently since the records are not ordered. Additionally, in order to support different types of spatial queries using the spatial coordinates, a multidimensional index (e.g., an R-tree) can be build on top of the data points. The *Find* operation retrieves a record from the disk by using the auxiliary index. The *GaS* and *GSs* operations are specific to aggregate network queries. The *GaS* operation retrieves the record of a specified successor of a given junction from the page buffer. If the page that stores the record is not in the page buffer, a *Find* operation is performed in order to retrieve the page from the disk. Similarly, the *GSs* operation retrieves the records of all successors of a given junction by scanning its successor list. While searching for the records of successors, it retrieves the records that are currently in the page buffer. If there are records that are not found in the page buffer, one of the remaining records is retrieved by invoking a *Find* operation. This process is iteratively repeated until all records of successors are retrieved.

In road networks, frequently observed aggregate network queries include route evaluation and path computation queries [35], in which an aggregate property is defined as a function of the attributes of junctions and links. In order to derive the aggregate properties, route evaluation queries require retrieval of all junctions and links in a specified route, which is a sequence of junctions $\langle t_1, t_2, t_3, \ldots, t_k\rangle$ and links $\langle \ell_{12}, \ell_{23}, \ldots, \ell_{(k-1)k}\rangle$. A naive approach for route evaluation is to perform a sequence of *Find* operations on the specified junctions. However, a much better alternative is to perform an initial *Find* operation followed by a sequence of *GaS* operations. Path computation queries deploy iterative search algorithms such as the breadth-first search, best-first search, A* search, and Dijkstra's algorithm [36] on the network to derive the aggregate properties. The Dijkstra's algorithm processes an unvisited junction that is extracted from the priority queue at each iteration, where processing a junction means scanning its successor list to compute the aggregate property. Thus, in path computation queries, records are accessed through a sequence of *Find* and *GSs* operation pairs (i.e., *Find*, *GSs*, ..., *Find*, *GSs*), where the *Find* operations are selectively performed only if the record is not found in the current page buffer.

2.1.3. Problem definition

Since the data records, which contain the topology- and application-dependent attributes, do not fit into the volatile memory, they must be stored in the secondary storage. In processing aggregate network queries, a vast amount of data must be iteratively accessed in such a way that records of connected junctions are likely to be concurrently accessed. Consequently, the disk pages that contain these records must be fetched into the memory for processing.

If the previous query logs are available, they are utilized to compute the access frequencies of the junctions and links; otherwise, a uniform frequency distribution is assumed. We use $f(t_i)$ and $f(t_i, t_j)$ to, respectively, denote the access frequency of junction t_i in $GSs(t_i)$ operations and the access frequency of the link from junction t_i to junction t_j in $GaS(t_i, t_j)$ operations. In the sample road network of Fig. 1,

access frequencies are given for *GSs* and *GaS* operations, respectively, in squares and on directed edges.

Given a road network $(\mathscr{T}, \mathscr{L})$ and the above-mentioned frequency functions extracted from the query logs, the record-to-page allocation problem can be stated as the problem of allocating 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 total disk access cost is reduced 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 store the records that are likely to be concurrently accessed in the same pages as much as possible. This clustering objective relates to minimizing the disk access costs of *GaS* and *GSs* operations in network queries. This way, efficiency in query processing can be achieved since fewer disk accesses are usually required to fetch all relevant records. However, neither the clustering graph model nor our model try to encapsulate the additional disk access cost of *Find* operations incurred by the priority queue processing in path computation queries.

2.2. Graph and hypergraph partitioning

An undirected graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is defined as a set of vertices $\mathscr{V}$ and a set of edges $\mathscr{E}$. Every edge $e_{ij} \in \mathscr{E}$ connects a pair of distinct vertices v_i and v_j. A weight $w(v_i)$ is associated to each vertex $v_i \in \mathscr{V}$ and a cost $c(e_{ij})$ is assigned with each edge $e_{ij} \in \mathscr{E}$. $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_K\}$ is a K-way vertex partition of $\mathscr{G}$ 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 of $\mathscr{G}$, an edge is said to be cut if its pair of vertices fall into two different parts and uncut otherwise. The partitioning objective is to minimize the cutsize defined over the cut edges $\mathscr{E}_{\text{cut}}$, that is,

$$\text{Cutsize}(\Pi) = \sum_{e_{ij} \in \mathscr{E}_{\text{cut}}} c(e_{ij}). \tag{1}$$

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 $W_k = \sum_{v_i \in \mathscr{V}_k} w(v_i)$ denotes the weight of part $\mathscr{V}_k$ and $W_{\max}$ denotes the maximum allowed part weight.

A hypergraph $\mathscr{H} = (\mathscr{V}, \mathscr{N})$ consists of a set of vertices $\mathscr{V}$ and a set of nets $\mathscr{N}$ [37]. Each net $n_j \in \mathscr{N}$ connects a subset of vertices in $\mathscr{V}$, which are called as the pins of n_j and denoted as Pins(n_j). Hence, graph is a special instance of a hypergraph where each net has exactly two pins. Each vertex v_i has a weight $w(v_i)$, and each net n_j has a cost $c(n_j)$.

In a given K-way vertex partition $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots, \mathscr{V}_K\}$ of $\mathscr{H}$, 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 an external net and is said to be cut. The partitioning objective is to minimize a cutsize metric defined over the cut nets. In the literature, a number of cutsize metrics are employed [38]. In connectivity-1 $(\lambda - 1)$ metric, which is widely used in VLSI [39] and in scientific computing [40–42] communities, each net n_j contributes $c(n_j)$ $(\lambda(n_j) - 1)$ to the cutsize of a partition Π. That is,

$$\text{Cutsize}(\Pi) = \sum_{n_j \in \mathscr{N}} c(n_j)(\lambda(n_j) - 1). \tag{2}$$

The partitioning constraint is to maintain an upper bound on part weights as in graph partitioning.

3. Clustering graph model and its flaws

In this section, we briefly describe the clustering graph model proposed by Shekhar and Liu [22] and flaws of this model.

3.1. Clustering graph representation

An undirected clustering graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is created to model the network $(\mathscr{T}, \mathscr{L})$. In $\mathscr{G}$, a vertex $v_i \in \mathscr{V}$ exists for each record $r_i \in \mathscr{R}$ storing the data associated with junction $t_i \in \mathscr{T}$. The size of a record r_i is assigned as the weight $w(v_i)$ of vertex v_i. There exists an edge e_{ij} between vertices v_i and v_j, for $i<j$, if junctions t_i and t_j are connected by at least one link. That is, $e_{ij} \in \mathscr{E}$ if either $\ell_{ij} \in \mathscr{L}$ or $\ell_{ji} \in \mathscr{L}$ or both. The cost $c(e_{ij})$ associated with e_{ij} is

$$c(e_{ij}) = \begin{cases} f(t_i) + f(t_i, t_j) & \text{if } \ell_{ij} \in \mathscr{L},\ \ell_{ji} \notin \mathscr{L}, \\ f(t_j) + f(t_j, t_i) & \text{if } \ell_{ji} \in \mathscr{L},\ \ell_{ij} \notin \mathscr{L}, \\ f(t_i) + f(t_j) + f(t_i, t_j) & \\ \quad + f(t_j, t_i) & \text{if } \ell_{ij}, \ell_{ji} \in \mathscr{L}. \end{cases} \tag{3}$$

3.2. Clustering graph model

Shekhar and Liu [22] formulate the record-to-page allocation problem as the problem of partitioning the clustering graph $\mathscr{G}$ with the disk page size being the upper bound on part weights. Shekhar and Liu partition $\mathscr{G}$ into a number of parts $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots\}$, where 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 objective is to maximize the weighted connectivity residue ratio (WCRR) metric, which corresponds to maximizing the sum of the costs of internal edges in a partition. It can be shown that maximizing WCRR is equivalent to minimizing the cutsize given in Eq. (1). This cutsize relates to the total disk access cost of aggregate network operations. Note that, in the original problem definition given in Section 2.1.3, the number K of parts is not known in advance. Thus, they use a partitioning algorithm based on RB with ratio-cut heuristic in order to create a number of parts, each with a size less than or equal to the disk page size.

Fig. 2 shows the clustering graph $\mathscr{G}$ for the sample network given in Fig. 1. In the figure, circles denote vertices, and lines denote edges. For the sake of clarity, $\mathscr{G}$ is displayed in two parts, where edge costs represent the access frequencies of *GaS* operations in Fig. 2(a) and *GSs* operations in Fig. 2(b). Fig. 2 also shows a 3-way partition $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \mathscr{V}_3\}$ of $\mathscr{G}$, where the shaded regions represent the vertex parts.

3.3. Flaws of clustering graph model

Although the clustering graph model correctly captures the disk access cost of *GaS* operations invoked in route evaluation queries, it fails to correctly capture the cost of *GSs* operations invoked in path computation queries. Consider a junction t_i with $d_{\text{out}}(t_i) > 1$ successive junctions. Assume that the records of these $d_{\text{out}}(t_i) + 1$ junctions span two disk pages. The cost of such an assignment should always be $f(t_i)$. However, the cost estimated by the clustering graph model depends on the distribution of these $d_{\text{out}}(t_i) + 1$ records across the two pages. Consider a distribution in which record r_i is assigned to a page together with $m < d_{\text{out}}(t_i) - 1$ of the records corresponding to successors of t_i and the remaining $d_{\text{out}}(t_i) - m$ records are assigned to the other page. In this case, the graph model estimates the cost as $(d_{\text{out}}(t_i) - m) \times f(t_i)$, which is an overestimation compared to the actual cost $f(t_i)$. This flaw easily extends to the cases where these $d_{\text{out}}(t_i) + 1$ records are distributed among more than two pages. On the other hand, the graph model succeeds in cases where each record corresponding to successors of junction t_i, except the ones in the page of r_i, is allocated to a separate page.

In Fig. 2, we illustrate the deficiency of the clustering graph model in estimating the total disk access cost of *GSs* operations using the sample partition $\Pi = \{\mathscr{V}_1 = \{v_1, v_4, v_5\}, \mathscr{V}_2 = \{v_2, v_3\}, \mathscr{V}_3 = \{v_6, v_7, v_8\}\}$. According to partition Π, the total costs of *GaS* and *GSs* operations, due to the cut edges in $\mathscr{E}_{\text{cut}} = \{e_{12}, e_{13}, e_{24}, e_{34}, e_{46}, e_{47}, e_{57}\}$, are computed as 51 and 73, respectively. Note that 51 is a correct estimation for the cost of *GaS* operations. However, the disk access cost of *GSs* operations is overestimated as 73, whereas the actual cost is 53. This difference $73 - 53 = 20$ stems from the overestimation of the costs of the $GSs(t_1)$ and $GSs(t_7)$ operations by the clustering graph model. For example, the disk access cost of $GSs(t_1)$ operations, where the set of successors of t_1 is $\text{Succ}(t_1) = \{t_2, t_3, t_4\}$, is overestimated as $2 \times 11 = 22$ due to the cut edges e_{12}, e_{13}, each with a cost of 11. However, the actual cost is $f(t_1) = 11$ since page P_2, which contains records r_2 and r_3, is accessed and placed into the page buffer only once to retrieve both r_2 and r_3 at each $GSs(t_1)$ operation.

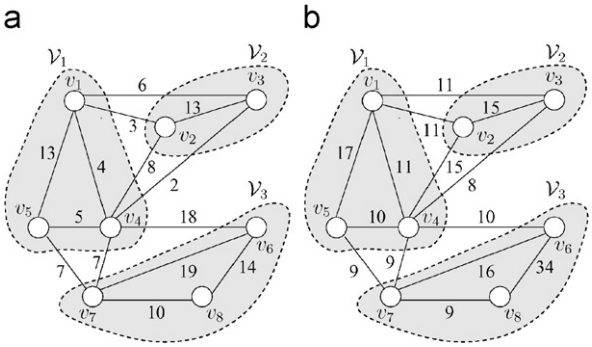

Fig. 2. A 3-way vertex partition, which models disk access costs of (a) *GaS* and (b) *GSs* operations for the clustering graph $\mathscr{G}$ of the sample network given in Fig. 1.

4. Clustering hypergraph model

In this section, we propose a clustering hypergraph model for the record-to-page allocation problem.

4.1. Clustering hypergraph representation

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

We employ two-pin nets in $\mathscr{H}$ to represent the disk access cost of *GaS* operations. In $\mathscr{N}^{GaS}$, there exists a two-pin net n_{ij} with $\text{Pins}(n_{ij}) = \{v_i, v_j\}$, for $i<j$, if junctions t_i and t_j are connected by at least one link. That is, $n_{ij} \in \mathscr{N}^{GaS}$ if either $\ell_{ij} \in \mathscr{L}$ or $\ell_{ji} \in \mathscr{L}$ or both. The cost $c(n_{ij})$ associated with n_{ij} for capturing the costs of $GaS(t_i, t_j)$ and $GaS(t_j, t_i)$ operations is

$$c(n_{ij}) = \begin{cases} f(t_i, t_j) & \text{if } \ell_{ij} \in \mathscr{L},\ \ell_{ji} \notin \mathscr{L}, \\ f(t_j, t_i) & \text{if } \ell_{ji} \in \mathscr{L},\ \ell_{ij} \notin \mathscr{L}, \\ f(t_i, t_j) + f(t_j, t_i) & \text{if } \ell_{ij}, \ell_{ji} \in \mathscr{L}. \end{cases} \tag{4}$$

Note that these two-pin nets correspond to the edges employed in the clustering graph described in Section 3.1 with the difference that their costs do not include the costs of *GSs* operations (i.e., the access frequency $f(t_i)$ of junction t_i). Fig. 3(a) displays the two-pin net construction for a pair of neighbor junctions t_1 and t_2.

We employ multi-pin nets in $\mathscr{H}$ to represent the disk access cost of *GSs* operations. In $\mathscr{N}^{GSs}$, there exists a $(d_{\text{out}}(t_i)+1)$-pin net n_i for each junction t_i with $d_{\text{out}}(t_i)>0$ successors, where n_i connects vertex v_i and the vertices corresponding to the records of the junctions that are in the successor list of t_i. That is,

$$\text{Pins}(n_i) = \{v_i\} \cup \{v_j : t_j \in \text{Succ}(t_i)\},$$

where $\text{Succ}(t_i)$ is the set of successors of t_i. Each net n_i is associated with a cost

$$c(n_i) = f(t_i) \tag{5}$$

to capture the cost of $GSs(t_i)$ operations. Fig. 3(b) displays the multi-pin net construction for junction t_1 with $\text{Succ}(t_1) = \{t_2, t_3, t_4\}$.

The size of the clustering hypergraph $\mathscr{H}$ in terms of the number of pins depends on the topological properties of the network. The number $|\mathscr{N}^{GaS}|$ of two-pin nets varies between $\lceil |\mathscr{L}|/2 \rceil$ and $|\mathscr{L}|$. The number $|\mathscr{N}^{GSs}|$ of multi-pin nets equals $|\mathscr{T}| - \alpha$, where $\alpha = |\{t_i : d_{\text{out}}(t_i) = 0\}|$ is the number of dead ends. The number of pins introduced by multi-pin nets is $|\mathscr{L}| + |\mathscr{T}| - \alpha$. Hence, the total number of pins in $\mathscr{H}$ is between $2\lceil |\mathscr{L}|/2 \rceil + |\mathscr{L}| + |\mathscr{T}| - \alpha$ and $3|\mathscr{L}| + |\mathscr{T}| - \alpha$.

4.2. Clustering hypergraph model

After modeling the network $(\mathscr{T}, \mathscr{L})$ as a clustering hypergraph $\mathscr{H}$, we formulate the record-to-page allocation problem as the problem of partitioning $\mathscr{H}$ with the disk page size, P, being the upper bound on part weights (i.e., $W_{\max} = P$). A hypergraph partitioning algorithm can be used to partition the clustering hypergraph into a number of parts $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots\}$. Here, partition Π induces a record-to-page allocation, where each part $\mathscr{V}_k \in \Pi$ corresponds to the subset of records to be allocated to disk page $\mathscr{P}_k \in \mathscr{P}$. That is, $v_i \in V_k$ means that record r_i is allocated to page $\mathscr{P}_k$.

In our model, the cutsize of a partition Π relates to the total number of disk accesses incurred by the *GaS* and *GSs* operations. The cutsize can be written as

$$\begin{aligned} &\text{Cutsize}(\Pi) \\ &= \sum_{n_i \in \mathscr{N}^{GaS}} c(n_i)(\lambda(n_i) - 1) + \sum_{n_i \in \mathscr{N}^{GSs}} c(n_i)(\lambda(n_i) - 1) \\ &= \sum_{n_i \in \mathscr{N}} c(n_i)(\lambda(n_i) - 1). \end{aligned} \tag{6}$$

In fact, under the assumption that a single-page buffer is available, the $\lambda - 1$ cost incurred to the partition by the two-pin cut nets in $\mathscr{N}^{GaS}$ exactly corresponds to the disk access cost incurred by the

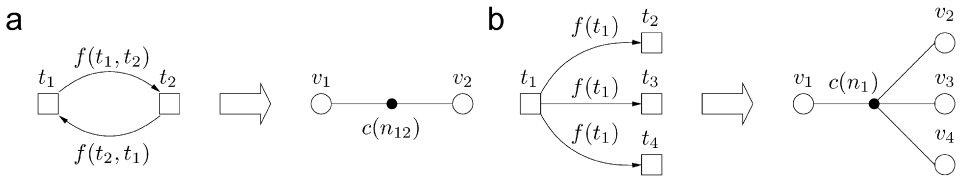

Fig. 3. The clustering hypergraph: (a) Two-pin net n_{12} for the $GaS(t_1, t_2)$ and $GaS(t_2, t_1)$ operations; (b) multi-pin net n_1 for the $GSs(t_1)$ operations.

GaS operations in the route evaluation queries. With the assumption of a single-page buffer, the $\lambda - 1$ metric is also able to encapsulate the disk access cost of *GSs* operations in the path computation queries. Our model correctly captures the aggregate disk access costs of *GaS* and *GSs* operations. Consequently, in our model, minimizing Cutsize(Π) given in Eq. (2) exactly minimizes the total number of disk accesses.

To illustrate the correctness of our model, we provide the following example. Consider a partition Π and a two-pin net $n_{ij} \in \mathcal{N}^{GaS}$ with $\text{Pins}(n_{ij}) = \{v_i, v_j\}$. If n_{ij} is internal to a part $\mathcal{V}_k$, then records r_i and r_j both reside in page $\mathscr{P}_k$. Since both r_i and r_j can be found in the memory when $\mathscr{P}_k$ is in the page buffer, neither $GaS(t_i, t_j)$ nor $GaS(t_j, t_i)$ operations incur any disk access. If n_{ij} is a cut net with connectivity set $\Lambda(n_{ij}) = \{\mathcal{V}_k, \mathcal{V}_m\}$, r_i and r_j reside in separate pages $\mathscr{P}_k$ and $\mathscr{P}_m$. Without loss of generality, assume that $r_i \in \mathscr{P}_k$ and $r_j \in \mathscr{P}_m$. In this case, $GaS(t_i, t_j)$ operations incur $f(t_i, t_j)$ disk accesses in order to replace the current page $\mathscr{P}_k$ in the buffer with $\mathscr{P}_m$ in the disk. In a similar manner, $GaS(t_j, t_i)$ operations incur $f(t_j, t_i)$ disk accesses in order to replace the current page $\mathscr{P}_m$ in the buffer with $\mathscr{P}_k$ in the disk. Hence, cut net n_{ij} incurs a cost of $c(n_{ij})$ to the cutsize since $\lambda(n_{ij}) - 1 = 1$.

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

In Fig. 4, we illustrate the correctness of the clustering hypergraph $\mathscr{H}$ for the network given in Fig. 1 using partition $\Pi = \{\mathcal{V}_1 = \{v_1, v_5\}, \mathcal{V}_2 = \{v_2, v_3, v_4\}, \mathcal{V}_3 = \{v_6, v_7, v_8\}\}$. For the sake of simplicity, $\mathscr{H}$ is given in two parts which separately show the net sets $\mathcal{N}^{GaS}$ and $\mathcal{N}^{GSs}$ with the associated costs of *GaS* and *GSs* operations shown in parentheses. According to partition Π, the disk access cost of *GaS* operations, due to the set $\{n_{12}, n_{13}, n_{14}, n_{45}, n_{46}, n_{47}, n_{57}\}$ of cut nets, is computed as $(3 + 6 + 4 + 5 + 18 + 7 + 7)(2 - 1) = 50$ since each of these nets has a connectivity of 2. The disk access cost of *GSs* operations, due to the set $\{n_1, n_4, n_7\}$ of cut nets, is computed as $11 \times (2 - 1) + 10 \times (3 - 1) + 9 \times (3 - 1) = 49$ since the connectivities of these nets are 2, 3, and 3, respectively. Note that internal nets do not incur any cost for neither *GaS* nor *GSs* operations since they have a connectivity of 1. In Fig. 4(b), consider

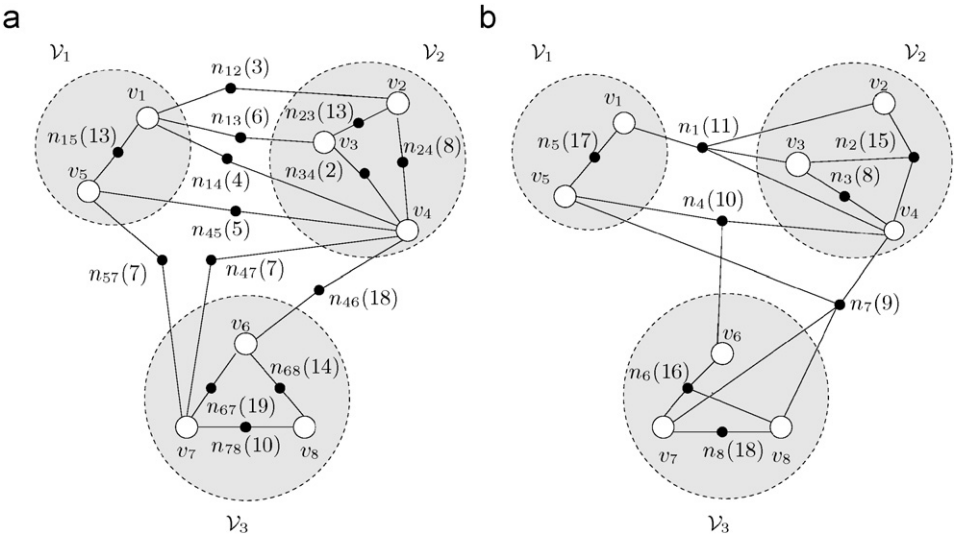

Fig. 4. The clustering hypergraph $\mathscr{H}$ for the network given in Fig. 1 and a 3-way vertex partition separately shown on net-induced subhypergraphs: (a) $(\mathcal{V}, \mathcal{N}^{GaS})$ and (b) $(\mathcal{V}, \mathcal{N}^{GSs})$, respectively, modeling disk access costs of *GaS* and *GSs* operations.

cut net n_1 with $\text{Pins}(n_1) = \{v_1, v_2, v_3, v_4\}$ and $\Lambda(n_1) = \{\mathscr{V}_1, \mathscr{V}_2\}$. Since v_1 is in vertex part $\mathscr{V}_1$, page $\mathscr{P}_1$ must be the single page in the buffer when $GSs(t_1)$ operations are invoked. Since v_2, v_3, and v_4 are all in $\mathscr{V}_2$, each of the 11 $GSs(t_1)$ operations will incur only one disk access for page $\mathscr{P}_2$ to bring it into the page buffer for retrieving records r_2, r_3 and r_4.

It is worthwhile to elaborate on the difference between the partitions produced by the clustering graph and hypergraph models for the network given in Fig. 1. In Figs. 2 and 4, both models achieve their optimum partitions under the partitioning constraint of three records per page. The clustering hypergraph model achieves a better record-to-page allocation with a disk access cost of 99 compared to the clustering graph model which has a cost of 124. This is basically due to the difference in the assignment of vertex v_4 to parts; v_4 is in $\mathscr{V}_1$ in the clustering graph model, whereas it is in $\mathscr{V}_2$ in the clustering hypergraph model. The clustering graph model fails to obtain this better allocation since the high cost of edge e_{14} due to $GSs(t_1)$ operation prevents vertex v_4 from moving to $\mathscr{V}_2$ although introducing this edge to the cut actually incurs no additional cost.

5. Recursive graph/hypergraph bipartitioning schemes

In the record-to-page allocation problem, a secondary objective, in addition to the main objective of minimizing the number of disk accesses, is to keep the number of allocated disk pages as small as possible. Since the size of each record varies depending on the topological properties of the network, the total number K of pages to be allocated is not known in advance. The lower bound on K is equal to the ratio of the total size of the records to the disk page size. It is hard to achieve this lower bound since the problem of minimizing the number of disk pages is NP-hard (bin-packing problem [43]) even without the main objective of minimizing the number of disk accesses.

The RB paradigm is widely used in K-way graph/hypergraph partitioning and known to be amenable to produce good solution qualities. This paradigm is inherently suitable for partitioning graphs and hypergraphs when K is not known in advance. Hence, the RB paradigm can be successfully employed in the clustering graph and hypergraph models for solving the record-to-page allocation problem.

In the RB paradigm, first, a two-way partition of the graph/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 graph partitioning, the cut-edge removal scheme is adopted, that is, the cut edges of the bipartition are removed after each bipartitioning step. In RB-based hypergraph partitioning, the cut-net splitting scheme [41] is adopted to capture the $\lambda - 1$ cutsize metric given in Eq. (2). In both graph and hypergraph partitioning, balancing the part weights of the bipartition is enforced as the bipartitioning constraint.

5.1. Determining the number of pages via RB

Here, we introduce two RB schemes, based on different bipartitioning constraints, to partition the records into pages while trying to minimize the total number of allocated pages. In both schemes, bipartitioning is recursively employed for partitioning the clustering graphs/hypergraphs until all parts have weights less than or equal to $W_{\max}$, which is set to be equal to the disk page size, P. In a resulting partition $\Pi = \{\mathscr{V}_1, \mathscr{V}_2, \ldots\}$, a part $\mathscr{V}_k$ is said to be lightly loaded if $(W_k/W_{\max}) \leqslant 0.5$ and heavily loaded otherwise.

In the first scheme, RB1, the bipartitioning constraint on part weights is to obtain two nearly equal-sized parts. That is,

$$W_1, W_2 \leqslant \frac{W_1 + W_2}{2}(1 + \varepsilon), \tag{7}$$

where W_1 and W_2 are the weights of the parts of the bipartition, and ε is the allowed deviation ratio over the predetermined load distribution. The deviation ratio is introduced to provide a flexibility to the bipartitioning heuristics for achieving lower cutsize values. In essence, each bipartitioning step tries to balance the part weights to maintain a uniform space utilization among the pages. We slightly adapt the bipartitioning constraint given in Eq. (7) when the weight of a part to be bipartitioned drops below $3P$. One of the parts is forced to have a weight close to P in order to increase the load factor of heavily loaded parts thus increasing the number of lightly loaded parts. By this adaptation, it is possible to further decrease K since lightly loaded parts are likely to be generated and packed into pages.

On the other hand, in the second scheme, RB2, the bipartitioning constraint on part weights is

$$W_1, W_2 \leqslant P\left\lceil \frac{W_1 + W_2}{2P} \right\rceil (1+\varepsilon) \quad (8)$$

to obtain a distribution such that one of the part weights is nearly a multiple of P. In essence, weight distribution of parts is adaptively tuned to decrease K by increasing the load factors of the heavily loaded parts with the assumption of a following phase in which the lightly loaded parts will be packed. This objective is tried to be achieved by making one of the part weights approximately a multiple of the disk page size, instead of dividing a part into two nearly equal-sized parts. However, due to the allowed deviation ratio, this partitioning approach may generate large numbers of lightly loaded parts.

5.2. Packing lightly loaded parts

Elimination of lightly loaded parts can be formulated as an instance of the bin-packing problem [43], where the parts correspond to items, pages correspond to bins, and the disk page size corresponds to bin capacity. The best-fit-decreasing heuristic used in solving the bin-packing problem is adopted to obtain a final distribution of parts to pages. Parts are assigned to pages in decreasing size order, where the best-fit criterion corresponds to assigning a part to a page which currently has the minimum space utilization.

It is also possible to further improve the primary objective of minimizing the total disk access cost while reducing the number of allocated pages. This can be done by modifying the best-fit criterion such that a part is assigned to a page that already contains part(s) with the highest weighted net connectivity to the part to be assigned. However, experimental results show that the gain is at most 0.5% of the total cutsize. The improvement of packing is very small since the lightly loaded parts are generated at relatively distant branches of the recursive bipartitioning tree and the cutsize contribution of the nets that connect such parts is typically very small.

6. Experimental results

In this section, we first describe the experimental setup. Then, we evaluate the performance of RB1 and RB2 schemes in terms of both the number of allocated pages and the cutsize and investigate the effect of packing lightly loaded parts. Finally, we evaluate the performance of the clustering graph and hypergraph models with changing page and page buffer sizes in terms of both the cutsize, which gives the total number of disk page accesses incurred by the *GaS* and *GSs* operations, and the total number of disk accesses in aggregate network queries.

6.1. Experimental setup

Experiments are conducted on a wide range of real-life road network data sets collected from US Tiger/Line [44] (Minnesota7 including 7 counties Anoka, Carver, Dakota, Hennepin, Ramsey, Scott, Washington; San Francisco; Oregon; New Mexico; Washington), US Department of Transportation [45] (California Highway Planning Network), and Brinkhoff's network data generator [46] (Oldenburg; San Joaquin). These data sets are primarily composed of points and polylines connecting the points. Since there is no embedded direction information in these data sets, we assume that all links are bidirectional. In general, self-loops and multi-links can be modeled by introducing dummy nodes to generate a simple network graph. But, for simplicity, we perform a preprocessing over these data sets to eliminate self-loops, multi-links, and points that do not correspond to a junction (i.e., a junction must be connected to at least three points). All experiments are conducted on these preprocessed data sets, whose properties are given in Table 1, where the data sets are listed in the order of increasing network size. In Table 1, d_{avg} denotes the average number of predecessors and successors of a junction.

The clustering graph and hypergraph models are constructed based on the assumption that network usage frequencies are available in the query logs. One way to gather the frequencies is to record the access frequencies of junctions and links. In our experiments, we generate a number of queries and their access frequencies using a synthetic query generation approach. Specifically, Brinkhoff's network generator for moving objects framework [46] is employed with the network-based node selection option to generate a set of source and destination junction pairs. For route evaluation queries, shortest paths between the source and destination junctions are computed, whereas for path computation queries, Dijkstra's algorithm is executed over

Table 1
Properties of road network data sets and queries performed on these networks

Tag	Data set	Road network			Avg. # of junctions accessed in queries	
		$\lvert\mathscr{T}\rvert$	$\lvert\mathscr{L}\rvert$	d_{avg}	Route eval.	Path comp.
D1	Oldenburg	4465	10778	2.41	11	109
D2	California HPN	10141	28370	2.80	16	217
D3	San Joaquin	17444	45974	2.64	19	419
D4	Minnesota7	34222	92206	2.69	31	1207
D5	San Francisco	166558	426742	2.56	48	3577
D6	New Mexico	448959	1112230	2.48	99	8767
D7	Oregon	507212	1203344	2.37	105	10214
D8	Washington	548901	1304126	2.38	107	12696

Table 2
Properties of the clustering graphs and hypergraphs used in the models

Tag	$\mathscr{G}$		$\mathscr{H}$			w_{avg}	W_{total}
	$\lvert\mathscr{V}\rvert$	$\lvert\mathscr{E}\rvert$	$\lvert\mathscr{V}\rvert$	$\lvert\mathscr{N}\rvert$	S		
D1	4465	5389	4465	7886	22085	42.62	190308
D2	10141	14181	10141	17369	52965	48.76	494484
D3	17444	22801	17444	26225	80785	46.17	805360
D4	34222	45419	34222	46056	149493	47.11	1612184
D5	166558	212916	166558	197630	654990	44.99	7494104
D6	448959	554448	448959	510477	1682429	43.64	19591516
D7	507212	601340	507212	574353	1844579	41.96	21282352
D8	548901	650268	548901	613558	1980535	42.01	23051620

the network between the source and destination junctions. For each data set, 1000 queries are generated. These queries are used for first generating the query logs and then measuring the total disk access cost during the experiments. The number of junctions accessed in the queries varies depending on the topological properties and the size of the data set. The average number of junctions accessed in the queries for each data set is given in Table 1.

Table 2 displays the properties of the clustering graphs and hypergraphs used in the experiments. In the table, S refers to the size of clustering hypergraphs and is equal to the total number of pins in these hypergraphs. Throughout the experiments, we reserve 12 bytes for link attributes and 4 bytes for the coordinates of a junction. No space is reserved for junction attributes. As described in Section 4, in our model, vertex weights correspond to record sizes in bytes. The average vertex weight, w_{avg}, and the total vertex weight, W_{total}, for each data set are also given in the table. Since all of the links in the data sets are bidirectional, in the hypergraph model, coordinates in the predecessor list of a junction will be equivalent to the ones in the successor list of the junction. Hence, we store only the successor list of the corresponding junction.

The state-of-the-art tools MeTiS [47] and PaToH [48] are used with default parameters for bipartitioning the clustering graphs and hypergraphs in the RB1 and RB2 schemes. The deviation ratio ε in Eqs. (7) and (8) is set to 0.10 throughout the experiments. Here, the allocation of the records of a given data set for a given page size is referred to as an allocation instance. The experiments are conducted on 32 different allocation instances for storing the records of the eight data sets in four different page sizes of $P = 1, 2, 4$, and 8 KB. Due to the randomized nature of the partitioning heuristics, the experiment for each allocation instance is repeated 100 times and the average performance results are reported in the following figures.

Furthermore, simulations are conducted in order to observe the effect of reducing the total disk access cost of *GaS* and *GSs* operations on the total number of disk accesses incurred by the aggregate network queries. A B^+ tree with Z-ordering is implemented for efficient record retrieval. In order to evaluate the effect of disk caching on our models, simulations are performed using page buffers with a capacity of 1, 2, 4, and 8 pages. The least recently used (LRU) page replacement algorithm is employed as the caching algorithm.

All experiments are performed on a PC that is equipped with an Intel Pentium IV 2.6 GHz processor, 2 GB of RAM. As the operating system, Mandrake 10.0 is installed.

6.2. Comparison of RB1 and RB2 schemes

In Fig. 5, we compare the performance of the RB1 and RB2 schemes in terms of the number K of allocated pages and show the effect of packing on K. If no packing is employed, RB1 achieves 10.3% smaller K values than RB2 on the average. However, as expected, packing results in much better improvement in K in RB2 when compared with RB1. As seen in Fig. 5, in RB1, packing achieves a small reduction in K (only 4.1% on the average) since the partitions created by this scheme involve few lightly loaded pages, thus resulting in

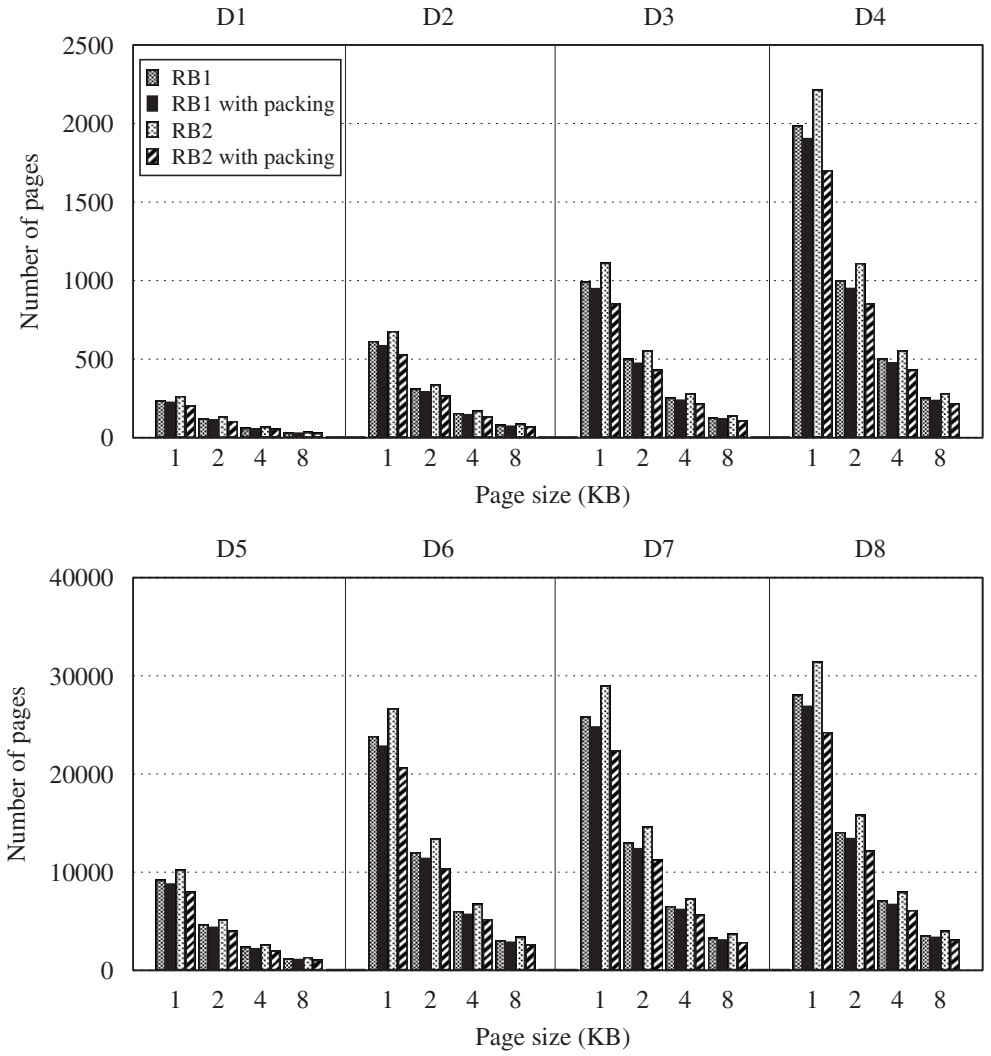

Fig. 5. Comparison of RB1 and RB2 schemes for D1–D8 data sets in terms of number of allocated pages.

restricted solution spaces for the packing algorithm. In contrast, RB2 creates many lightly loaded pages, providing a flexibility in the solution space for packing. In RB2, packing reduces K by 22.7% on the average.

In the same way, in Fig. 6, we compare the performance of the RB1 and RB2 schemes in terms of the cutsize and provide the effect of packing on the cutsize. If no packing is employed, RB1 achieves 1.7% smaller cutsize values than RB2 on the average. Although reducing K decreases the cutsize in general, the improvement of packing in the cutsize is quite small for both RB1 and RB2 schemes as the cutsize among the packed parts is small.

According to the results, RB2 with packing achieves 10.1% smaller K values while yielding 1.7% higher cutsize values than RB1 with packing on the average. In the following subsections, we only present the performance results for the clustering graph and hypergraph models employing the RB2 scheme with packing since the models employing the RB1 scheme with packing give a similar performance in reducing the cutsize. Note that the RB2 scheme allocates the data over a number of pages which is at most 9.8% (8.1% on the average) higher than the lower bound on K.

6.3. Comparison of clustering graph and hypergraph models

Fig. 7 displays the variation in the percent cutsize improvement of the clustering hypergraph model

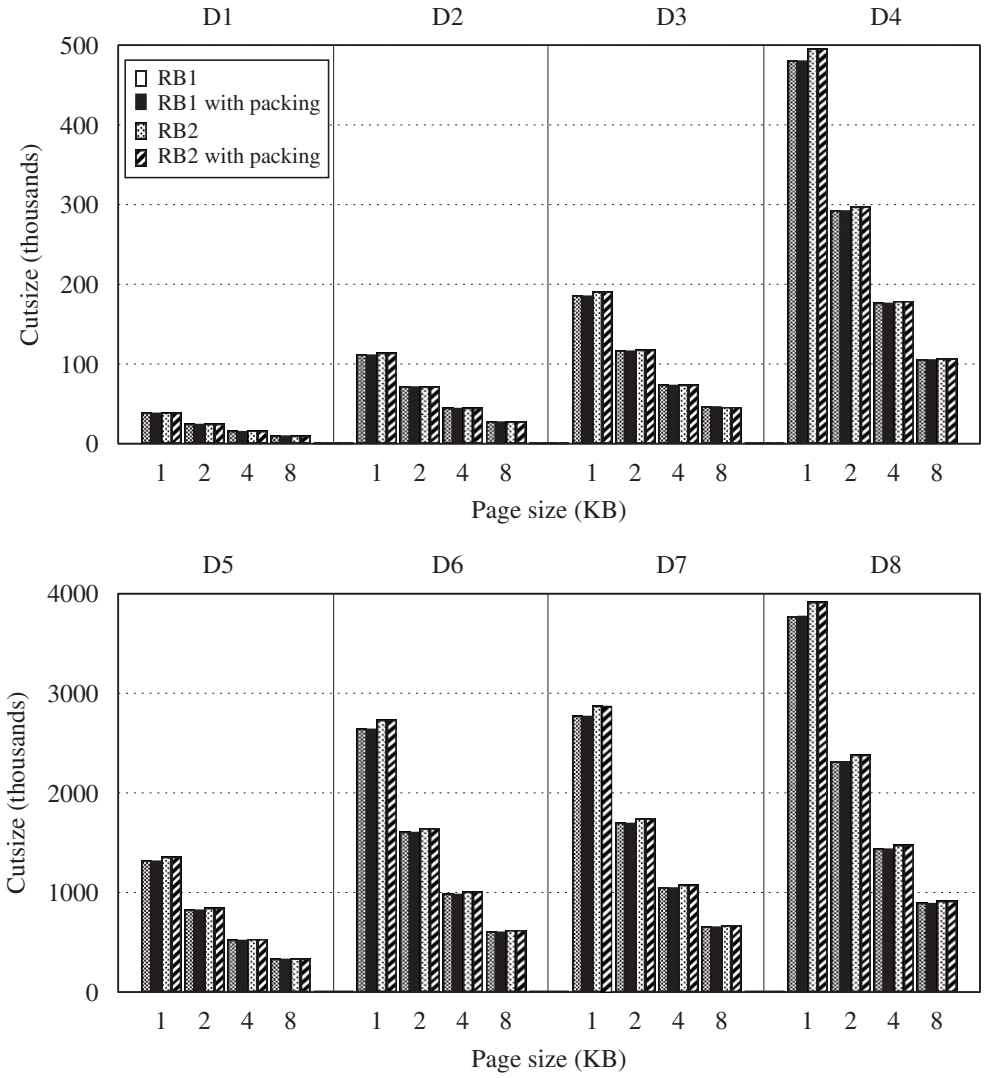

Fig. 6. Comparison of RB1 and RB2 schemes for D1–D8 data sets in terms of the cutsize.

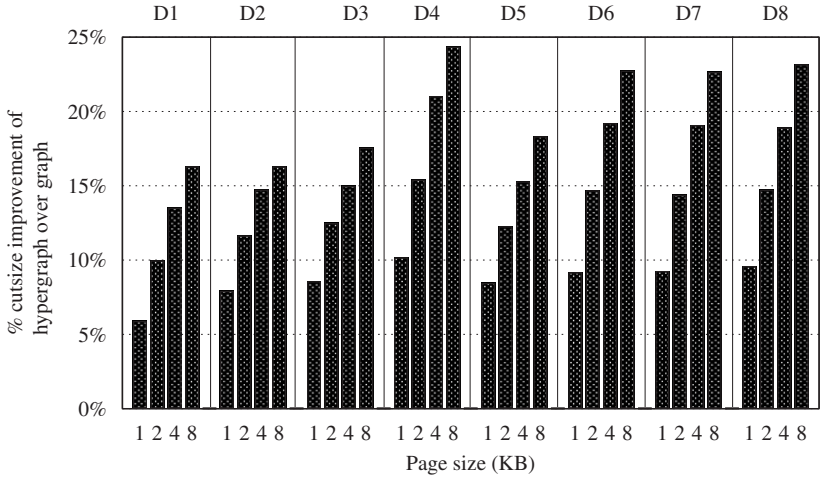

Fig. 7. Percent cutsize improvement of the clustering hypergraph model over the clustering graph model.

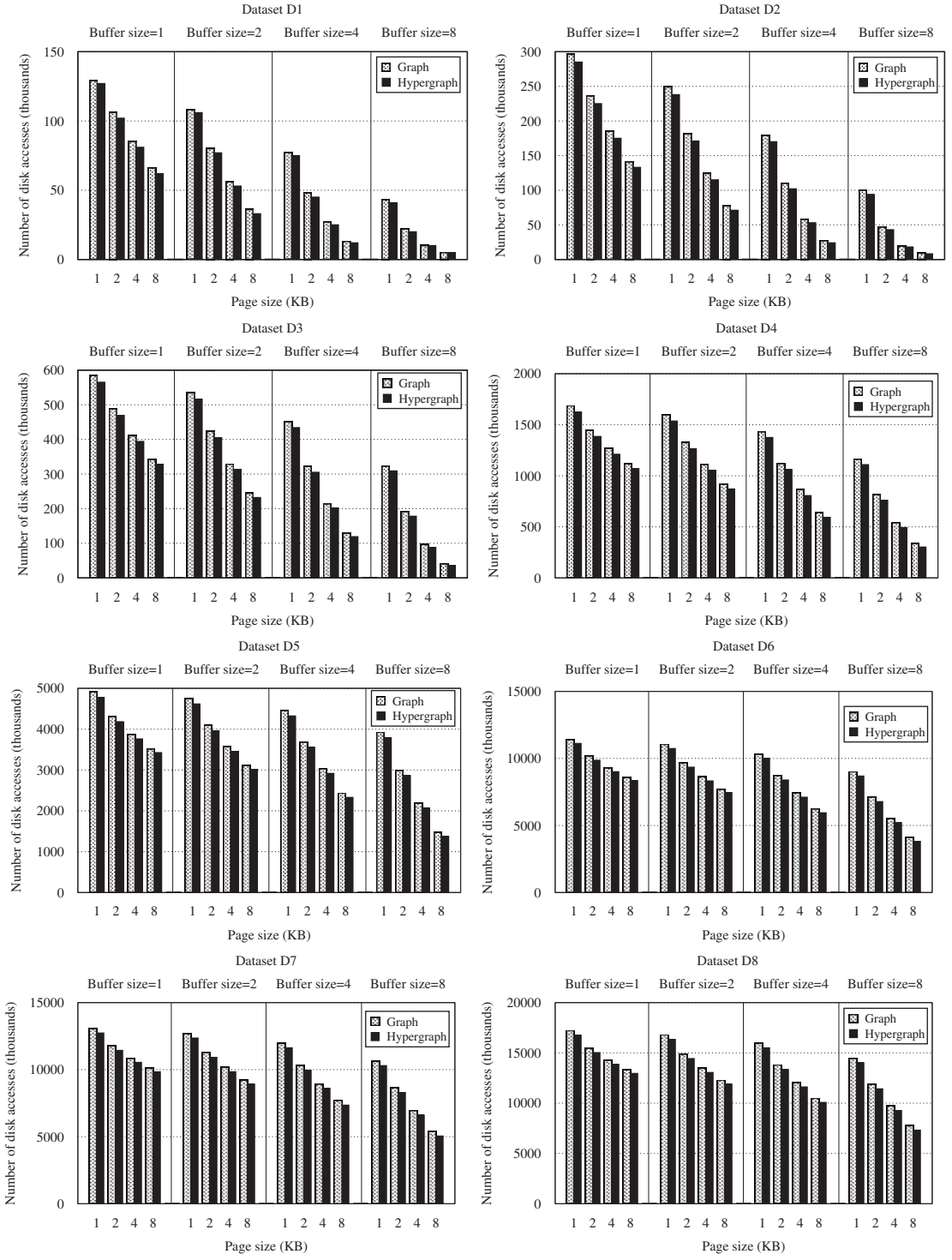

Fig. 8. The total disk access costs of aggregate network queries for each data set in the clustering graph and hypergraph models with increasing page size P in KB and page buffer size in number of pages.

over the clustering graph model with increasing page size. Recall that cutsize values encapsulate the total disk access costs of *GaS* and *GSs* operations. It is important to note that, in these experiments, the models obtain rather close numbers of disk pages. The graph model achieves K values only 0.03% smaller than the hypergraph model on the overall average. This enables a fair comparison of the cutsize values attained by these two models.

As seen in Fig. 7, the hypergraph model performs better than the graph model in terms of the cutsize for every allocation instance. In all data sets, the performance gap between the two models increases with increasing page size in favor of the hypergraph model. When P is doubled, the cutsize values obtained by the clustering graph and hypergraph models, respectively, decrease by 35.6% and 38.4%, on the average. This is because decreasing the page size increases the likelihood of distributing the records of the successors of a junction across separate pages, thus allowing the graph model to avoid the flaw mentioned in Section 3.3. On the average over all allocation instances, the performance improvement of the hypergraph model over the graph model is 14.7%.

In Fig. 8, we compare the performance of the clustering graph and hypergraph models via simulation in terms of the total number of disk accesses incurred by aggregate network queries and provide the effect of changing P and page buffer size. Recall that the cost of aggregate network query processing includes the costs of *GaS* and *GSs* operations and the cost of priority queue operations. Increasing P and the buffer size independently decrease the number of disk accesses in both models since the chance of assigning concurrently accessed records to the pages that are already in the memory increases. In all simulation results with different buffer sizes, the hypergraph model performs better than the graph model in reducing the number of disk page accesses. On the average, the performance gap between the two models increases with increasing page size in favor of the hypergraph model.

The effect of page buffer size on the performance of these models is also important. In almost all data sets, the percent performance improvement of the hypergraph model over the graph model increases with increasing buffer size independent of the page size. There are only two exceptions in simulations on the smallest data sets D1 and D2 using the largest buffer size of eight pages and the largest page size of 8 KB. In these cases, a considerable portion of the data reside in the memory, and hence the clustering models loose their effectiveness.

Comparison of the total disk access cost of *GaS* and *GSs* operations captured by the cutsize and the total disk access cost of aggregate network queries shows that, although the average improvement in the total disk access cost of *GaS* and *GSs* operations is 14.7%, the average improvement in the total disk access cost of aggregate network queries remains around 4.4%. This stems from the difference between the cutsize and the total disk access cost of aggregate network queries, which is due to the additional overhead of *Find* operations incurred by the priority queue processing in path computation queries. This overhead varies depending on the location in the memory hierarchy of the records matching the ids extracted from the priority queue. Hence, in the worst case, where all records must be retrieved from the disk, the overhead is equal to the total number of records accessed in path computation queries. In the experiments, for the single-page buffer case, this overhead is found to be around 80% of the total number of disk accesses on the average. The overhead of the network operations still remains around 20% despite our explicit effort towards minimizing this overhead.

7. Conclusion

We investigated the record-to-page allocation problem in road network databases. We showed that the state-of-the-art clustering graph model does not correctly capture the cost of the *Get-Successors* (*GSs*) operations incurred in path computation queries, and hence it is not suitable for road networks where the path computations occur frequently. In order to overcome this flaw, we proposed a clustering hypergraph model. Our model correctly captures the costs of disk accesses for both *Get-a-Successor* (*GaS*) operations incurred in route evaluation queries and *GSs* operations incurred in path computation queries. We also presented two recursive bipartitioning schemes to reduce the number of allocated disk pages while trying to minimize the number of disk page accesses. Experimental results obtained on a wide range of road networks verify the validity of our hypergraph model.

In this work, we solely concentrated on *GaS* and *GSs* network operations. However, the conducted simulations indicate that the cost of *Find* operations incurred by the priority queue processing in path

computations is also important. As a future work, we are planning to extend our work to incorporate the cost of the *Find* operations.

References

[1] P. Rigaux, M. Scholl, A. Voisard, Spatial Databases with Application to GIS, Morgan Kaufmann, Los Altos, CA, 2002.

[2] S. Shekhar, S. Chawla, Spatial Databases: A Tour, Prentice-Hall, Englewood Cliffs, NJ, 2003.

[3] M.F. Goodchild, Towards an enumeration and classification of GIS functions, in: Proceedings of International Geographic Information Systems Symposium: The Research Agenda, NASA, 1987, pp. 62–77.

[4] C.S. Jensen, J. Kolar, T.B. Pedersen, I. Timko, Nearest neighbor queries in road networks, in: Proceedings of the International Symposium on Advances in Geographic Information Systems, ACM, New York, 2003, pp. 1–8.

[5] M.R. Kolahdouzan, C. Shahabi, Alternative solutions for continuous K nearest neighbor queries in spatial network databases, GeoInformatica 9 (4) (2005) 321–341.

[6] R. Laurini, D. Thompson, Fundamentals of Spatial Information Systems, The A.P.I.C Series, vol. 37, Academic Press, New York, 1992 (Chapters 2.5.4, 5).

[7] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, S. Teng, On trip planning queries in spatial databases, in: Ninth International Symposium on Spatial and Temporal Databases, 2005, pp. 273–290.

[8] M.L. Yiu, N. Mamoulis, D. Papadias, Aggregate nearest neighbor queries in road networks, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 820–833.

[9] F.B. Zhan, C.E. Noon, Shortest path algorithms: an evaluation using real road networks, Transp. Sci. 32 (1) (1998) 65–73.

[10] N. Mamoulis, D. Papadias, M.L. Yiu, Aggregate nearest neighbor queries in road networks, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 820–833.

[11] D. Papadias, Y. Tao, M. Kyriakos, K.H. Chun, Aggregate nearest neighbor queries in spatial databases, ACM Trans. Database Syst. 30 (2) (2005) 529–576.

[12] Y. Tao, D. Papadias, Range aggregate processing in spatial databases, IEEE Trans. Knowl. Data Eng. 16 (12) (2004) 1555–1570.

[13] M. Yiu, D. Papadias, N. Mamoulis, Y. Tao, Reverse nearest neighbors in large graphs, IEEE Trans. Knowl. Data Eng. 18 (4) (2006) 540–553.

[14] N. Jing, Y.W. Huang, E.A. Rundensteiner, Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation, IEEE Trans. Knowl. Data Eng. 10 (3) (1998) 409–432.

[15] S. Yung, S. Pramanik, An efficient path computation model for hierarchically structured topological road maps, IEEE Trans. Knowl. Data Eng. 14 (5) (2002) 1029–1046.

[16] V. Gaede, O. Günther, Multidimensional access methods, ACM Comput. Surv. 30 (2) (1998) 170–231.

[17] Y.W. Huang, N. Jing, E.A. Rundensteiner, Effective graph clustering for path queries in digital map databases, in: Proceedings of the International Conference on Information and Knowledge Management, ACM, New York, 1996, pp. 215–222.

[18] 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.

[19] H. Sagan, Space-filling Curves, Springer, Berlin, 1994.

[20] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, Reading, MA, 1990.

[21] H. Samet, Spatial data structures, Modern Database Systems: The Object Model, Interoperability, and Beyond, Addison-Wesley/ACM Press, Reading, MA, New York, 1995, pp. 361–385.

[22] S. Shekhar, D.R. Liu, A connectivity-based access method for networks and network computation, IEEE Trans. Knowl. Data Eng. 9 (1) (1997) 102–117.

[23] S.H. Woo, S.B. Yang, An improved network clustering method for I/O-efficient query processing, in: Proceedings of the Symposium on Advances in Geographic Information Systems, ACM, New York, 2000, pp. 62–68.

[24] S. Shekhar, A. Fetterer, Path computation in advanced traveler information systems, in: Proceedings of the Sixth Annual Meeting and Exposition of the Intelligent Transportation Society of America, Houston, TX, August 1996.

[25] S. Shekhar, A. Kohli, M. Coyle, Can proximity-based access methods efficiently support network computations, Technical Report, Computer Science Department, University of Minnesota, 1993.

[26] M.L. Yiu, N. Mamoulis, Clustering objects on a spatial network, in: Proceedings of International Conference on Information and Knowledge Management, ACM, New York, 2005, pp. 443–454.

[27] J. Banerjee, S. Kim, W. Kim, J. Garza, Clustering DAG for CAD databases, IEEE Trans. Software Eng. 14 (11) (November 1988) 1684–1699.

[28] S. Dar, H.V. Jagadish, A spanning tree transitive closure algorithm, in: Proceedings of International Conference on Data Engineering, IEEE, New York, 1992, pp. 2–11.

[29] K. Hua, J. Su, C. Hua, Efficient evaluation of traversal recursive queries using connectivity index, in: Proceedings of the International Conference on Data Engineering, IEEE, New York, 1993, pp. 549–558.

[30] P.A. Larson, V. Deshpande, A file structure supporting traversal recursion, in: Proceedings of the International Conference on Information and Knowledge Management, ACM, New York, 1989, pp. 243–252.

[31] R. Agrawal, J. Kiernan, An access structure for generalized transitive closure queries, in: Proceedings of International Conference on Data Engineering, IEEE, New York, 1993, pp. 429–438.

[32] T. Caldwell, On finding minimum routes in a network with turn penalties, Commun. ACM (1961) 107–108.

[33] C. Huang, L. Meng, C. Zhao, A road network data model and its application in vehicle navigation system, in: Proceedings of the Symposium on Integrated System for Spatial Data Production, Custodian and Decision Support, 2002.

[34] J. Jiang, G. Han, J. Chen, Modeling turning restrictions in traffic network for vehicle navigation system, in: Proceedings of the Symposium on Geospatial Theory, Processing, and Applications, 2002.

[35] S. Shekhar, A. Kohli, M. Coyle, Path computation algorithms for advanced traveler information systems, in: Proceedings of the International Conference on Data Engineering, IEEE, New York, 1993, pp. 31–39.

[36] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Math. 1 (1959) 269–271.
[37] C. Berge, Graphs and Hypergraphs, North-Holland Publishing Company, Amsterdam, 1973.
[38] C.J. Alpert, A.B. Kahng, Recent directions in netlist partitioning: a survey, VLSI J. 19 (1–2) (1995) 1–81.
[39] A. Dasdan, C. Aykanat, Two novel multiway circuit partitioning algorithms using relaxed locking, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 16 (2) (1997) 169–178.
[40] C. Aykanat, A. Pinar, Ü.V. Çatalyürek, Permuting sparse rectangular matrices into block-diagonal form, SIAM J. Sci. Comput. 25 (6) (2004) 1860–1879.
[41] Ü.V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition of parallel sparse-matrix vector multiplication, IEEE Trans. Parallel Distrib. Syst. 10 (7) (1999) 673–693.
[42] B. Ucar, C. Aykanat, Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix–vector multiplies, SIAM J. Sci. Comput. 25 (6) (2004) 1837–1859.
[43] E. Horowitz, S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Rockville, MD, 1978.
[44] U.S. Census Bureau, Topologically integrated geographic encoding and referencing system (TIGER), ⟨http://www.census.gov/geo/www/tiger/⟩, 2002.
[45] U.S. Department of Transportation, Federal Highway Administration, The National Highway Planning Network, ⟨http://www.fhwa.dot.gov/planning/nhpn/⟩, 2004.
[46] T. Brinkhoff, A framework for generating network-based moving objects, GeoInformatica 6 (2) (2002) 153–180.
[47] G. Karypis, V. Kumar, MeTiS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, Version 4.0, Computer Science and Engineering Department, University of Minnesota, 1998, ⟨http://www-users.cs.umn.edu/~karypis/metis/⟩.
[48] Ü.V. Çatalyürek, C. Aykanat, PaToH: partitioning tool for hypergraphs, Technical Report, Computer Engineering Department, Bilkent University, 1999, ⟨http://www.cs.bilkent.edu.tr/~aykanat/pargrp/patoh/⟩.