Contents lists available at ScienceDirect

Information Sciences

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

Efficient successor retrieval operations for aggregate query processing on clustered road networks

Engin Demir[a], Cevdet Aykanat[b,*]

[a] Department of Computer Science and Engineering, Ohio State University, 43210 Columbus, OH, USA
[b] Department of Computer Engineering, Bilkent University, 06800 Bilkent, Ankara, Turkey

ARTICLE INFO

Article history:
Received 18 December 2008
Received in revised form 12 November 2009
Accepted 19 March 2010

Keywords:
Storage management
Data retrieval operation
Road networks
Clustering
Hypergraphs

ABSTRACT

Get-Successors (*GS*) which retrieves all successors of a junction is a kernel operation used to facilitate aggregate computations in road network queries. Efficient implementation of the *GS* operation is crucial since the disk access cost of this operation constitutes a considerable portion of the total query processing cost. Firstly, we propose a new successor retrieval operation *Get-Unevaluated-Successors* (*GUS*), which retrieves only the unevaluated successors of a given junction. The *GUS* operation is an efficient implementation of the *GS* operation, where the candidate successors to be retrieved are pruned according to the properties and state of the algorithm. Secondly, we propose a hypergraph-based model for clustering successively retrieved junctions by the *GUS* operations to the same pages. The proposed model utilizes query logs to correctly capture the disk access cost of *GUS* operations. The proposed *GUS* operation and associated clustering model are evaluated for two different instances of *GUS* operations which typically arise in Dijkstra's single source shortest path algorithm and incremental network expansion framework. Our simulation results show that the proposed successor retrieval operation together with the proposed clustering hypergraph model is quite effective in reducing the number of disk accesses in query processing.

© 2010 Published by Elsevier Inc.

1. Introduction

1.1. Motivation

In geographic information systems (GIS), with the advance of global positioning systems, the importance of applications that manage spatial data is increasing. GIS software systems store the geographic data either in disk-based files or in large scale database management systems according to the volume of data. Software systems modeling the spatial networks (e.g., ArcGIS, gvSIG, PostLBS) store network data in two different layers, namely the geometric network and logical network. In the geometric network, junctions and links are stored with their class features to visualize the data. On the other hand, in the logical network, a special data structure is stored to represent the topology of the network. In this topology, commodities flow through links, and links connect together at junctions, where flow from one link is transferred to another. In a logical network, geometry is not important but the connectivity of links and junctions is. That is why systems modeling the spatial networks deal with almost exclusively with the logical network.

* Corresponding author. Tel.: +90 312 2901625; fax: +90 312 2664047.
E-mail addresses: demir@cse.ohio-state.edu (E. Demir), aykanat@cs.bilkent.edu.tr (C. Aykanat).

0020-0255/$ - see front matter © 2010 Published by Elsevier Inc.
doi:10.1016/j.ins.2010.03.015

A well-known example of spatial networks is road networks, which form an integral part of many GIS applications such as intelligent traveling systems, vehicle telematics, location-aware advertising, and guided-tours to tourists. A road network is represented as a two-tuple $(\mathcal{T}, \mathcal{L})$, where $\mathcal{T}$ and $\mathcal{L}$, respectively, denote the junctions and the road segments (links) between pairs of junctions. In this representation, $\ell_{ij} \in \mathcal{L}$ denotes a link from a junction $t_i \in \mathcal{T}$ to a successor junction $t_i \in \mathcal{T}$. Several attributes are associated with junctions (e.g., locations, turn restrictions) and links (e.g., length, average speed limit, capacity, type, location related information). Additionally, point-of-interest data is also associated with junctions and links. Due to the large volume of road network data, it is primarily stored in the secondary storage. When the read/write time of network data in the secondary storage is compared with the computation cost of the network queries, disk access cost could be very high during query processing.

In location-based services, the position and accessibility of spatial objects are constrained to underlying networks such as roads, highways, and railways. Route planning applications such as MapQuest, MapPoint, map services of major search engines and mobile cell-phone operators have become essential tools for obtaining driving directions. In such applications, the distance between two objects is determined by the length of the shortest path connecting these two objects in the network. In the network-based query processing, network distances can be either computed on-the-fly [17,29,41] using the state-of-the-art shortest path algorithms [14,19] or pre-computed and stored on disk to support efficient distance computations [12,21,22,24]. On the other hand, the underlying network can be transformed into another representation in which a network distance between two objects can be found in constant time [30,31]. There is no best strategy for network distance computation as the performance of these strategies depend on the network properties such as network size, object density, and frequency of network updates. In general, the performance optimization in network query processing has a focus on minimizing the cost of network data accesses and network distance computations.

In query processing, shortest path computation algorithms traverse the network using the connectivity information rather than geometric proximity information. Hence, network queries use topological operations such as *Get-Successors* (*GS*) and aggregate sequence operations such as *Find* and *Get-A-Successor* (*GaS*). The *GaS* and *GS* operations are unique to aggregate network queries including route evaluation and path computations, and they, respectively, retrieve one and all successors of a junction to facilitate aggregate computations on networks. Efficient implementations of the *GaS* and *GS* operations are crucial since the disk access cost of these successor retrieval operations constitute a considerable portion of the total query processing cost.

The expected disk access cost of successor retrieval operations can be reduced by clustering successively accessed data into the same disk pages [18]. This way, a junction with its successors that are most likely to be accessed together via the *GaS* and *GS* operations are allocated in the same disk page. Furthermore, query logs can be used to predict the future access patterns. Recent query logs can be used to discover the access frequencies of the data so that both the connectivity information and the access frequencies of junctions can be utilized to achieve efficiency gains in query processing [26].

1.2. Related work

In this section, related work on disk-based data allocation schemes for road networks is presented. Huang et al. [20] describe a general scheme, where links of the network are stored in a separate link table. In their approach, the link table is clustered into disk pages such that each page stores the information of links whose source coordinates are closely located. This approach is based on spatial locality, and hence the clustering of links does not utilize the connectivity information.

In the following studies, the importance of connectivity information in networks is realized, and graph models [32,39] are proposed to cluster the data into disk pages. Shekhar and Liu [32] propose the junction-based storage scheme, where each junction together with its connectivity information is stored in a data record. They evaluated their graph clustering model for the junction-based storage scheme by both uniform access frequencies and frequencies extracted from query logs. The usage of query logs is reported to yield better performance results. The clustering model proposed by Woo and Yang [39] achieves the minimum number of disk pages based on the assumption that records have fixed size. These graph clustering models are used in the recent spatial query processing and clustering works [1,22,40,41].

Papadias et al. [29] propose a data structure that integrates connectivity information with the spatial properties. The successor lists of junctions that are close in space according to their Hilbert ordering are placed in the same disk page. This approach uses the connectivity information in order to cluster data into disk pages but does not utilize the access frequencies of junctions in queries.

In a recent work [15], we show that although the clustering graph models accurately capture the disk access cost of *GaS* operations, it cannot correctly capture the disk access cost of *GS* operations. In the same work, we propose a clustering hypergraph model that correctly captures the cost of *GS* operations for the junction-based storage scheme. The record access patterns of previous queries are extracted from the query logs and used to correctly capture the disk access cost of operations in the upcoming queries. In this model, records are clustered into disk pages by hypergraph partitioning, where the partitioning objective corresponds to minimizing the disk access cost of *GS* operations in network queries. In [16], we introduce the link-based storage scheme, where each link together with its connectivity information is stored in a data record. We also propose a clustering hypergraph model for this new storage scheme.

1.3. Contributions

Our contributions are twofold. First, *Get-Unevaluated-Successors* (*GUS*) is introduced as a new successor retrieval operation for spatial network queries, which is overlooked in the literature. In network traversal algorithms, all successors of a junction need not be retrieved in each invocation of the *GS* operations since some of them may already be accessed and evaluated during processing a query. If the network is highly connected, than it is more probable that evaluation of some of the junctions are already performed and they do not need to be retrieved and evaluated again during query processing. Thus, *GUS* is an efficient implementation of the *GS* operation, where the candidate successors to be retrieved are pruned during query processing.

Second, a clustering hypergraph model that captures the disk access cost of *GUS* operations correctly for the junction-based storage scheme is proposed. The proposed model utilizes query logs to minimize the number of disk page accesses to be incurred by the network queries using *GUS* operation as the underlying successor retrieval operation. Furthermore, the proposed model tries to guarantee full space utilization and hence keeps the number of allocated disk pages at a reasonable amount.

The proposed *GUS* operation and associated hypergraph-based clustering model are evaluated for two different instances of *GUS* operations: *Get-Unprocessed-Successors* and *Get-Unvisited-Successors*. The former operation typically arises in Dijkstra's single source shortest path algorithms, and the latter operation typically arises in incremental network expansion framework.

The rest of the paper is organized as follows. Some background material is discussed in Section 2. The proposed *GUS* operation is introduced and discussed in Section 3. The clustering hypergraph model proposed for the *GUS* operations is discussed in Section 4. Experimental results are presented and discussed in Section 5. Finally, the paper is concluded in Section 6.

2. Preliminaries

2.1. Junction-based storage scheme

The adjacency list data structure is frequently used for storing the connectivity information of a road network in the secondary storage. In the junction-based storage scheme, each junction of the network is stored in a data record. 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 $\mathrm{Suc}(t_i)$ denotes the list of outgoing links of t_i. Each element in $Pre(t_i)$ stores the disk address 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 in $\mathrm{Suc}(t_i)$ stores the disk address 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.

2.2. 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 query logs, allocate a set of data records $\mathcal{R} = \{r_1, r_2, \ldots\}$ to a set of disk pages $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots\}$ 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 successively 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.

2.3. Graph and hypergraph partitioning

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Every edge $e_{ij} \in \mathcal{E}$ connects a pair of distinct vertices v_i and v_j. Each vertex v_i has a weight $w(v_i)$, and each edge e_{ij} has a cost $c(e_{ij})$. $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition of $\mathcal{G}$ if each part $\mathcal{V}_k$ is non-empty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$.

In a given K-way vertex partition Π of $\mathcal{G}$, 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 $\mathcal{E}_{\mathrm{cut}}$, that is,

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

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

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$ [5]. Each net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$, which are referred to as the pins of n_j and denoted as $\text{Pins}(n_j)$. 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 $\mathcal{H}$ is defined as the total number of its pins, i.e., $|\mathcal{H}| = \sum_{n_j \in \mathcal{N}} |n_j|$. Each vertex v_i has a weight $w(v_i)$, and each net n_j has a cost $c(n_j)$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition if each part $\mathcal{V}_k$ is non-empty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. 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.

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 the connectivity-1 metric, which is widely used in VLSI layout design [2,13] and in scientific computing [4,9,10,25,28,34–37], each net n_j incurs the cost $c(n_j)(\lambda(n_j) - 1)$ to the cutsize of a partition Π. That is,

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

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 hypergraph partitioning problem is known to be NP-hard [27]. However, successful hypergraph partitioning tools such as hMeTiS [23] and PaToH [11] exist in the literature. These tools utilize the multi-level framework [8] to provide high quality partitions at reasonable execution time. Although direct K-way hypergraph partitioning [3] is feasible, the recursive bipartitioning (RB) paradigm is widely used in K-way hypergraph partitioning and known to be amenable to produce high quality solutions. 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 the RB-based hypergraph partitioning, the cut net splitting scheme in [10] is adopted after each bipartitioning step to capture the connectivity-1 cutsize metric given in (2).

2.4. Clustering graph and hypergraph models

The clustering graph model is proposed by Shekhar and Liu [32], whereas the clustering hypergraph model is proposed in our earlier work [15]. Given a road network $(\mathcal{T}, \mathcal{L})$ and the frequencies of successor retrieval operations extracted from the query logs, the clustering graph and hypergraph models are constructed as follows. Let $f(t_i)$ denote the frequency of $GS(t_i)$ operations invoked on junction t_i and $f(t_i, t_j)$ denote the frequency of $GaS(t_i, t_j)$ operations invoked on the link from junction t_i to junction t_j. $(\mathcal{T}, \mathcal{L})$ is represented as a clustering graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a clustering hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ on the same vertex set. That is, there exist a vertex $v_i \in \mathcal{V}$ for each record $r_i \in \mathcal{R}$ storing the data associated with junction $t_i \in \mathcal{T}$. The size of a record r_i is assigned as the weight $w(v_i)$ of vertex v_i.

In the clustering graph model, edges represent the disk accesses of both GaS and GS operations. There exists an edge e_{ij} between vertices v_i and v_j due to $GS(t_i)$, $GS(t_j)$, $GaS(t_i, t_j)$, and $GaS(t_j, t_i)$ operations if junctions t_i and t_j are connected by at least one link. The cost associated with e_{ij} is $c(e_{ij}) = f(t_i) + f(t_j) + f(t_i, t_j) + f(t_j, t_i)$.

In the clustering hypergraph model, nets represent the disk accesses of both GaS and GS operations. There exists a two-pin net n_{ij} with $\text{Pins}(n_{ij}) = \{v_i, v_j\}$ due to $GaS(t_i, t_j)$ and $GaS(t_j, t_i)$ operations. The cost associated with n_{ij} is $c(n_{ij}) = f(t_i, t_j) + f(t_j, t_i)$. Furthermore, there exists a multi-pin net n_i with $\text{Pins}(n_i) = \{v_i\} \cup \{v_j : t_j \in \text{Suc}(t_i)\}$ due to $GS(t_i)$ operations. The cost associated with n_i is $c(n_i) = f(t_i)$.

After modeling the network $(\mathcal{T}, \mathcal{L})$ as a clustering graph/hypergraph, the graph/hypergraph is partitioned into a number of parts with the disk page size P being the upper bound on part weights. The resulting K-way partition 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. Since K is not known in advance, recursive bipartitioning framework is used in partitioning both clustering graph and hypergraph. The partitioning objective in the clustering graph model 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 (1). In the clustering hypergraph model, the partitioning objective is to minimize the cutsize according to (2). As shown in [15], the WCRR metric is exactly decodes the disk access cost of GaS operations under the single-page buffer assumption. However, the WCRR metric has deficiencies in capturing the disk access cost of GS operations. In the clustering hypergraph model, the cutsize exactly decodes the disk access costs of both GaS and GS operations under the single-page buffer assumption. That is, minimizing the cutsize according to (2) corresponds to minimizing the total number of disk accesses due to GaS and GS operations. Note that the two-pin nets due to GaS operations in the clustering hypergraph are equivalent to the edges in the clustering graph. Thus, the clustering graph and hypergraph models are effectively equivalent and display the same the performance in terms of encapsulating the disk access cost of GaS operations.

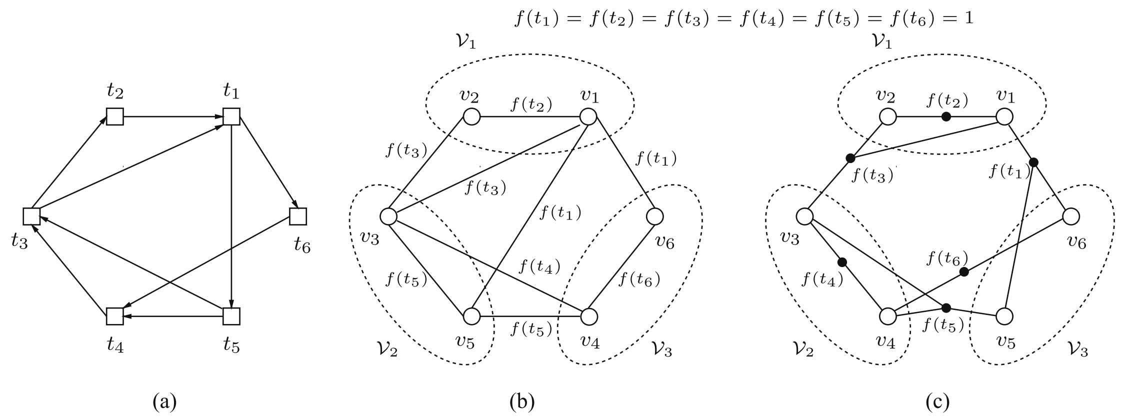


Fig. 1. (a) A sample road network, (b) the clustering graph and its 3-way optimum partition, and (c) the clustering hypergraph and its 3-way optimum partition.

Fig. 1 shows a sample network with 6 junctions and 9 links to compare the clustering graphs and hypergraphs. This figure also illustrates the deficiency of the clustering graph model in capturing the disk access cost of *GS* operations. Assuming only one *GS* operation is invoked on each junction, unit cost is assigned to all edges and nets as seen in Fig. 1b and c. The clustering graph and hypergraph models, respectively, achieve their optimum partitions in Fig. 1b and c under the partitioning constraint of two records per page. As shown in Fig. 1b, the cutsize is equal to 6 due to 6 cut edges, whereas the actual cost is 4. This difference is due to the overestimation of the costs of the $GS(t_1)$ and $GS(t_3)$ operations by the clustering graph model. For example, the disk access cost of $GS(t_1)$ operation, where the set of successors of t_1 is $\text{Suc}(t_1) = \{v_5, v_6\}$, is estimated as $2 \times 1 = 2$ due to cut edges e_{15}, e_{16}, each with a cost of 1. However, the actual cost is $f(t_1) = 1$ since page $\mathcal{P}_3$, which contains records r_5 and r_6, is accessed and placed into the page buffer only once to retrieve both r_5 and r_6 at each $GS(t_1)$ operation. As shown in Fig. 1c, the clustering hypergraph model correctly captures the cost of *GS* operations, since the cutsize is equal to 4 due to 4 cut nets, each with a connectivity of 2.

As reported in [15], the clustering hypergraph model achieves significantly better than the clustering graph model in record-to-page allocations for a wide range of road networks with query sets involving both *GaS* and *GS* operations. The reader is referred to [15] for a detailed theoretical and experimental comparison of the clustering graph and hypergraph models. Based on these findings, the focus in this paper is to develop clustering hypergraph models for encapsulating the disk access cost of *GUS* operations. The basic ideas proposed here for clustering hypergraph models can be easily extended to develop clustering graph models for *GUS* operations.

3. *Get-Unevaluated-Successors* (*GUS*) operation

For a given query, during the execution of the underlying search algorithm, those junctions, whose records are retrieved and the computation related with these records are completed, are said to be "evaluated". The remaining junctions are said to be "unevaluated". That is, a *GUS* operation is defined as retrieving the unevaluated successors of a given junction. The sequence of *GUS* operations to be performed for a given query can be efficiently implemented by maintaining a set of either evaluated or unevaluated junctions in-memory. That is, checking whether a given junction is evaluated or unevaluated can be simply achieved without retrieving the record of the junction. This way, only the records of the unevaluated successors of t are retrieved for a $GUS(t, \text{Suc}(t, U))$ operation, where U denotes the set of unevaluated junctions just before the invocation of the *GUS* operation for the current query. The set

$$\text{Suc}(t_i, U) = \{t_j : t_j \in \text{Suc}(t_i) \wedge t_j \in U\} \tag{3}$$

denotes the set of unevaluated successors of t_i. Note that in this notation $\text{Suc}(t_i)$ corresponds to $\text{Suc}(t_i, \mathcal{T})$. Two examples of *GUS* operations: Get-unProcessed-Successors (*GuPS*) and Get-unVisited-Successors (*GuVS*) are introduced.

The *GuPS* operation typically arises in Dijkstra's single source shortest path algorithm [19]. Dijkstra's algorithm repeatedly extracts an unprocessed junction from a priority queue and processes it, where processing a junction means scanning its successor list to compute an aggregate property. Thus, in the *GuPS* operation, evaluated junctions correspond to the processed junctions whose records will not be reevaluated during the execution of the search algorithm for a given query. Hence there is no need to retrieve the records of such junctions more than once. In order to clarify the usage of this operation, the pseudocode of the Dijkstra's single source shortest path algorithm [19] is shown in two parts: Algorithm 1 shows the main body of the algorithm, whereas Algorithm 2 shows an I/O efficient implementation of the *GuPS* operation. In Algorithms 1 and 2, Q represents an in-memory priority queue, which contains unprocessed junctions keyed with respect to their distance values from the source junction. So, Q effectively corresponds to the set U of unevaluated junctions as in the definition of *GUS*.

Recall that, in the algorithms using the same strategy presented in Dijkstra's algorithm [19], the *GuPS* operation is invoked while processing the elements extracted from the priority queue as in line 8 of Algorithm 1. As seen in Algorithm 2, the for-loop in lines 1–4 computes the set *PageSet* of pages that contain only unprocessed successor junctions and finally retrieves the pages in *PageSet*. In Algorithm 1, the doubly-nested for-loop in lines 9–14 shows the processing of junction t_i. In this *for* loop, the retrieved pages in *PageSet* are processed one by one to relax the distance values of unprocessed successors of junction t_i. Note that the pages that already reside in the page buffer are handled before the other pages in *PageSet*, and while handling a page, all unprocessed junctions in that page are processed before retrieving a new page.

Algorithm 1. Dijsktra's Single Source Shortest Path Algorithm

```
Require: $(\mathcal{T}, \mathcal{L})$, source junction $s$
1: for each junction $t_i$ in $\mathcal{T}$ do
2:     $dist[t_i] \leftarrow \infty$
3:     $previous[t_i] \leftarrow$ null
4: $dist[s] \leftarrow 0$
5: $Q \leftarrow \mathcal{T}$
6: while $Q$ is not empty
7:     $t_i \leftarrow$ EXTRACT_MIN$(Q)$[1]
8:     $GuPS(t_i, \mathrm{Suc}(t_i, Q))$
9:     for each retrieved page $P_i \in PageSet$ do
10:        for each successor $t_j \in P_i$ of $t_i$ do
11:            if $dist[t_i] + length(t_i, t_j) < dist[t_j]$ then
12:                $dist[t_j] \leftarrow dist[t_i] + length(t_i, t_j)$
13:                DECREASE_KEY$(Q, t_j, dist[t_j])$[2]
14:                $previous[t_j] \leftarrow t_i$
15: return $previous[]$
```

[1] EXTRACT_MIN removes and returns the element of the priority queue with the minimum key value.
[2] DECREASE_KEY decreases the key value of an element of the priority queue.

Algorithm 2. Get-unProcessed-Successors $GuPS(t_i, \mathrm{Suc}(t_i, Q))$

```
1: for each successor $t_j$ of $t_i$ do
2:     if $t_j \in Q$
3:         $PageSet \leftarrow PageSet \cup page[t_j]$
4: retrieve $PageSet$
```

The *GuVS* operation typically arises in algorithms using the network expansion framework. Algorithms using network expansion framework repeatedly extract an unvisited junction from a priority queue and scan its successor list. Thus, in the *GuVS* operation, evaluated junctions correspond to the already visited junctions whose records will not be re-visited during the execution of the search algorithm for a given query. Similar to the *GuPS* operation, there is no need to retrieve the records of these junctions more than once. In order to clarify the usage of the *GuVS* operation, the pseudocode of the k-nearest neighbor search using the incremental network expansion framework [29] is shown in two parts: Algorithm 3 shows the main body of the algorithm, whereas Algorithm 4 shows an I/O efficient implementation of the *GuVS* operation. Point-of-interests are discovered in such a way that the junctions are explored in the order of their network distance from the query point. In order to satisfy this property, a priority queue Q, which contains candidate unprocessed junctions keyed with respect to their network distance values from the query point, is stored in-memory. The set V contains unvisited junctions and effectively corresponds to the set U of unevaluated junctions as in the definition of *GUS*.

Recall that, in the algorithms using the network expansion framework, *GuVS* operation is invoked while processing the elements extracted from the priority queue in the expansion of the network (line 10, Algorithm 3). As seen in Algorithm 4, the for-loop in lines 1–4 computes the set *PageSet* of pages that contain only unvisited junctions and finally retrieves the pages in *PageSet*. In the doubly-nested for-loop in lines 11–18 of Algorithm 3, the retrieved pages in *PageSet* are processed one by one to update the nearest neighbor list by expanding the network search through the unvisited successors of junction t_i. Page handling strategy mentioned for the *GuPS* operation is also valid in this case. That is, the pages that already reside in the page buffer are handled before the other pages in *PageSet*, and while handling a page, all unvisited junctions in that page are visited before retrieving a new page.

Algorithm 3. *k*-Nearest Neighbor Search Using Incremental Network Expansion Framework

```
Require $(\mathcal{T},\mathcal{L})$, query point $q$, $Q$ is a min-heap keyed on $d_N(q,t)$
1: $V \leftarrow \mathcal{T}$
2: $t_it_j \leftarrow find\_segment(q)$
3: $S_{cover} \leftarrow find\_entities(t_it_j)$
4: $\{p_1,\ldots,p_k\} = k$ nearest entities in $S_{cover}$ sorted in ascending order of their network distance
5: $d_{Nmax} \leftarrow d_N(q,p_k)$ // if $p_k = \emptyset,\ d_{Nmax} = \infty$
6: INSERT$(Q, \langle(t_i, d_N(q,t_i)), (t_j, d_N(q,t_j))\rangle)$[1]
7: $t_i \leftarrow$ EXTRACT_MIN$(Q)$
8: $V \leftarrow V - \{t_i\}$
9: while $(d_N(q,t_i) < d_{Nmax})$
10:     $GuVS(t_i, \mathrm{Suc}(t_i,V))$
11:     for each retrieved page $P_i \in PageSet$
12:        for each successor $t_j \in P_i$ of $t_i$ do
13:           $V \leftarrow V - \{t_j\}$
14:           $S_{cover} \leftarrow find\_entities(t_it_j)$
15:           update $\{p_1,\ldots,p_k\}$ from $\{p_1,\ldots,p_k\} \cup S_{cover}$
16:           $d_{Nmax} \leftarrow d_N(q,p_k)$
17:           INSERT$(Q,(t_j,d_N(q,t_j)))$
18:     $t_i \leftarrow$ EXTRACT_MIN$(Q)$
19: return $\{p_1,\ldots,p_k\}$
```

[1] INSERT inserts a new element to the priority queue.

Algorithm 4. Get-unVisited-Successors $GuVS(t_i, \mathrm{Suc}(t_i,V))$

```
1: for each successor $t_j$ of $t_i$ do
2:     if $t_j \in V$ then
3:        $PageSet \leftarrow PageSet \cup page[t_j]$
4: retrieve $PageSet$
```

4. Clustering hypergraph model for *GUS* operations

In this section, our clustering hypergraph model, which correctly captures the cost of *GUS* operations for the junction-based storage scheme, is presented.

4.1. Clustering hypergraph representation

A clustering hypergraph $\mathcal{H}_{\mathrm{GUS}} = (\mathcal{V},\mathcal{N}_{\mathrm{GUS}})$ is created to model the network $(\mathcal{T},\mathcal{L})$. The vertices of $\mathcal{H}_{\mathrm{GUS}}$ represent the records storing the data associated with the junctions as in $\mathcal{H}_{\mathrm{GS}}$. That is, there exists a vertex $v_i \in \mathcal{V}$ for each junction $t_i \in \mathcal{T}$. The size of a record r_i is assigned as the weight $w(v_i)$ of vertex v_i. The set $\mathcal{N}_{\mathrm{GUS}}$ is composed of nets that represent the record access patterns of *GUS* operations. That is, each distinct *GUS* operation incurs a net in $\mathcal{N}_{\mathrm{GUS}}$. The set $GUS(t_i, \mathrm{Suc}(t_i,U))$ of *GUS* operations invoked on junction t_i with the same set $\mathrm{Suc}(t_i,U)$ of unevaluated successors incur a net $n_{\mathrm{Suc}(t_i,U)}$ with a cost

$$c(n_{\mathrm{Suc}(t_i,U)}) = f(t_i, \mathrm{Suc}(t_i,U)). \tag{4}$$

Here, $f(t_i, \mathrm{Suc}(t_i,U))$ denotes the frequency of the $GUS(t_i, \mathrm{Suc}(t_i,U))$ operations obtained from the query log. The net $n_{\mathrm{Suc}(t_i,U)}$ captures the record access pattern of such *GUS* operations by connecting vertex v_i and the vertices corresponding to $\mathrm{Suc}(t_i,U)$. That is,

$$\mathrm{Pins}(n_{\mathrm{Suc}(t_i,U)}) = \{v_i\} \cup \{v_j : t_j \in \mathrm{Suc}(t_i,U)\}. \tag{5}$$

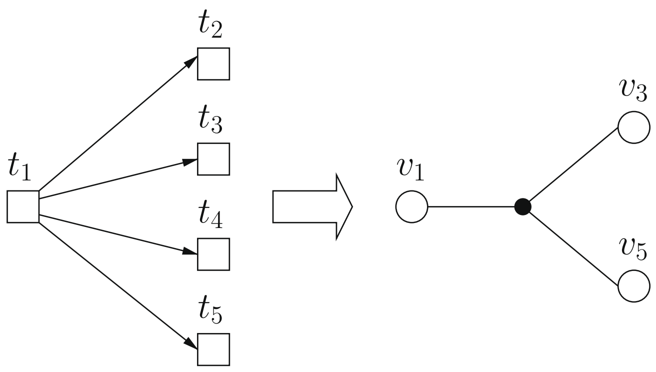


Fig. 2. The clustering hypergraph construction: $GUS(t_1, \{t_3, t_5\})$ incurs a net with pins $\{v_1, v_3, v_5\}$.

Note that the size of net $n_{\mathrm{Suc}(t_i,U)}$ can be between 2 and $d_{out}(t_i) + 1$ since $|\mathrm{Suc}(t_i, U)| \leqslant d_{out}(t_i)$. Single pin nets are discarded since $GUS(t_i, \mathrm{Suc}(t_i, U))$ operations with $\mathrm{Suc}(t_i, U) = \emptyset$ do not incur any record access. Fig. 2 displays the net construction for a $GUS(t_1, \mathrm{Suc}(t_1, U))$ operation invoked on junction t_1 with $\mathrm{Suc}(t_1) = \{t_2, t_3, t_4, t_5\}$ but $\mathrm{Suc}(t_1, U) = \{t_3, t_5\}$.

The size of hypergraph $\mathcal{H}_{\mathrm{GUS}}$ depends on both the topological properties of the network and the record access patterns in the query log. Each junction t_i with $d_{out}(t_i) > 1$ may incur as many as $2^{d_{out}(t_i)} - 1$ nets in $\mathcal{H}_{\mathrm{GUS}}$. Recall that $GS(t_i)$ operations invoked on junction t_i incur a single net of size $d_{out}(t_i) + 1$ in $\mathcal{H}_{\mathrm{GS}}$ for representing the record access pattern of GS operations. However, our experiments on realistic road networks with synthetic query sets show that the average number of nets generated per junction in $\mathcal{H}_{\mathrm{GUS}}$ remains below 3.6. Furthermore, the possibility of identical nets (those which have the same pin set) incurred by neighbor junctions can be exploited to decrease the number of nets by using the identical net detection and elimination algorithms in [3]. In identical net elimination process, a set of identical nets is collapsed into a single net whose cost is set to be equal to the sum of the costs of its constituent identical nets.

Although generation of $\mathcal{H}_{\mathrm{GS}}$ using the query log is a rather trivial task, generation of $\mathcal{H}_{\mathrm{GUS}}$ may need special attention. As in the GS case, it is assumed that a query log contains a sequence of junctions processed for each query, where the order of the sequence is determined by the order of retrieval of junction records. Let $q_i = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_k}, \cdots \rangle$ denote the sequence of junctions accessed during processing a query q_i of the log. Then, kth junction t_{i_k} in q_i corresponds to the $GUS(t_{i_k}, \mathrm{Suc}(t_{i_k}, U_{ik}))$ operation, where U_{ik} represents the set of unevaluated junctions just before the invocation of $GUS(t_{i_k}, \mathrm{Suc}(t_{i_k}, U_{ik}))$ in query q_i.

For the $GuPS(t_{i_k}, \mathrm{Suc}(t_{i_k}, U_{ik}))$ operations performed on junction t_{i_k},

$$U_{ik} = \mathcal{T} - q_{ik}, \tag{6}$$

whereas for the $GuVS(t_{i_k}, \mathrm{Suc}(t_{i_k}, U_{ik}))$ operations

$$U_{ik} = \mathcal{T} - q_{ik} - \bigcup_{t_j \in q_{ik}} \mathrm{Suc}(t_j). \tag{7}$$

Here, $q_{ik} = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_k} \rangle$ denotes the kth prefix subsequence of q_i. Note that the junction subsequence q_{ik} is also used as a junction subset in (6) and (7). Algorithms 5 and 6 show the pseudocodes for computing the frequencies of the $GuPS$ and $GuVS$ operations, respectively, from a given query log.

Efficient implementation of Algorithms 5 and 6 require efficient maintenance of the ⟨operation, frequency⟩ pairs. For this purpose, a list of GUS operations together with their frequencies for each junction is maintained. Each operation $GUS(t_i, \mathrm{Suc}(t_i, U))$ in the list of a junction t_i is encoded as a bit sequence stored in a byte assuming a junction has at most 8 successors. In this encoding, the positions of 1 bits in a byte determine the junctions in $\mathrm{Suc}(t_i, U)$. In this way, locating a $GUS(t_i, \cdot)$ operation for incrementing its frequency count requires m_i byte comparisons in the list for t_i, where m_i denotes the number of $GUS(t_i, \cdot)$ operations encountered so far in the query log.

Algorithm 5. Frequency computation for net cost determination in $\mathcal{H}_{\mathrm{GuPS}}$

```
Require Query log $Q_{\log} = \langle q_1, q_2, \ldots, q_n \rangle$, where $q_i = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_m} \rangle$
1: for each query $q_i$ in $Q_{\log}$ do
2:    $U \leftarrow \mathcal{T}$ ▷ $U$ denotes the set of unprocessed junctions
3:    for $k = 1$ to $|q_i|$ do
4:       $U \leftarrow U - \{t_{i_k}\}$
5:       for each successor $t_j \in \mathrm{Suc}(t_{i_k})$
6:          if $t_j \in U$ then
7:             $f(t_j, \mathrm{Suc}(t_{i_k}, U)) \leftarrow f(t_j, \mathrm{Suc}(t_{i_k}, U)) + 1$
```

Algorithm 6. Frequency computation for net cost determination in $\mathcal{H}_{\text{GuVS}}$

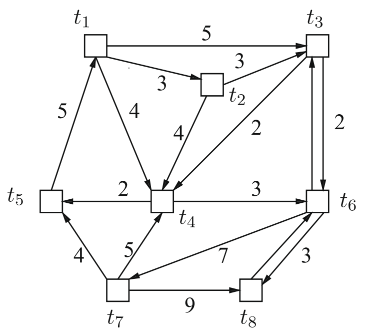
```
Require Query log $Q_{\log} = \langle q_1, q_2, \ldots, q_n \rangle$, where $q_i = \langle t_{i_1}, t_{i_2}, \ldots, t_{i_m} \rangle$
1: for each query $q_i$ in $Q_{\log}$ do
2:    $U \leftarrow \mathcal{T}$ ▷ $U$ denotes the set of unvisited junctions
3:    for $k = 1$ to $|q_i|$ do
4:       $U \leftarrow U - \{t_{i_k}\}$
5:       for each successor $t_j \in \text{Suc}(t_{i_k})$ do
6:          if $t_j \in U$ then
7:             $f(t_j, U) \leftarrow f(t_j, U) + 1$
8:             $U \leftarrow U - \{t_j\}$
```

4.2. Clustering hypergraph model

In the proposed clustering hypergraph model, the constructed hypergraph $\mathcal{H}_{\text{GUS}} = (\mathcal{V}, \mathcal{N}_{\text{GUS}})$ is partitioned into parts $\Pi = \{\mathcal{V}_1, \ldots, \mathcal{V}_k, \ldots\}$ to obtain a record-to-page allocation, where each vertex part $\mathcal{V}_k \in \Pi$ corresponds to the subset of records to be allocated to disk page $\mathcal{P}_k \in \mathcal{P}$. That is, if $v_i \in \mathcal{V}_k$ then record r_i is allocated to page $\mathcal{P}_k$. Hence, the vertex parts of Π correspond to the disk pages of the resulting allocation. The recursive bipartitioning (RB) paradigm is used to obtain Π, where the maximum allowed part weight is set to the disk page size (i.e., $W_{\max} = P$). That is, the partitioning constraint enforces that the disk page size is not exceeded in record-to-page allocation. In each bipartitioning step of the RB scheme, one of the parts is enforced to be nearly a multiple of page size with the intention of generating fully loaded parts (pages) at the end of the partitioning. After obtaining Π, there may be lightly loaded pages (i.e., pages less than half full) in the resulting allocation. These lightly loaded pages can be further packed by formulating the packing problem as an instance of the bin-packing problem, where the parts corresponds to items, pages corresponds to bins, and the disk page size corresponds to bin capacity [15]. In the packing algorithm, parts are assigned to pages in decreasing size order using the best-fit criterion, which corresponds to assigning a part to a page with the minimum space utilization. Alternatives such as adapting the best-fit heuristic to minimize the number of disk accesses due to *Find* and successor retrieval operations are also experimented but the percentage gain in the disk access cost is found to be very small. Thus, the best-fit packing heuristic, which is a fast approximation of the optimal packing algorithm, is used to decrease the total number of pages. The computational cost of packing lightly loaded parts is negligible but the decrease in the total number of parts is 24.8%, on the overall average.

Theorem 4.1. *Let $\mathcal{H}_{\text{GUS}} = (\mathcal{V}, \mathcal{N}_{\text{GUS}})$ denote the clustering hypergraph of a given network $(\mathcal{T}, \mathcal{L})$ for a given query log $Q_{\log}$. In partitioning of $\mathcal{H}_{\text{GUS}}$, the partitioning objective of minimizing the cutsize according to (2) corresponds to minimizing the total number of disk accesses incurred by the GUS operations under the single-page buffer assumption.*

Proof. Consider an internal net n_i of part $\mathcal{V}_k$ in partition Π. As seen in (2), n_i does not incur any cost to the cutsize. Since n_i is internal to part $\mathcal{V}_k$, record r_i and all records of the unevaluated successor junctions of t_i reside in page $\mathcal{P}_k$. Hence, $GUS(t_i, \text{Suc}(t_i, U))$ operations do not incur any disk access as page $\mathcal{P}_k$ is already in the page buffer. In Π, consider a cut net n_i with connectivity set $\Lambda(n_i)$. As seen in (2), n_i incurs a cost of $c(n_i)(\lambda(n_i) - 1)$ to the cutsize. The connectivity set $\Lambda(n_i)$ of n_i means that record r_i and the records of the unevaluated 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 $\mathcal{P}_k$, where $\mathcal{V}_k$ is in $\Lambda(n_i)$. In this case, each $GUS(t_i, \text{Suc}(t_i, U))$ operation incurs $\lambda(n_i) - 1$ page accesses in order to retrieve the records of the unevaluated successors of t_i by fetching the pages corresponding to the vertex parts in $\Lambda(n_i) - \{\mathcal{V}_k\}$ since page $\mathcal{P}_k$ is already in the page buffer when the $GUS(t_i, \text{Suc}(t_i, U))$ operation is invoked. □



QUERY SET	**QUERY LOG**
< Source, Destination >	Processed junction sequence
$< t_2, t_8 >$	$< t_2, t_3, t_4, t_6, t_5 >$
$< t_4, t_3 >$	$< t_4, t_5, t_6 >$
$< t_5, t_6 >$	$< t_5, t_1, t_2, t_4, t_3 >$
$< t_7, t_6 >$	$< t_7, t_5, t_4 >$

Fig. 3. A sample road network with a query set.

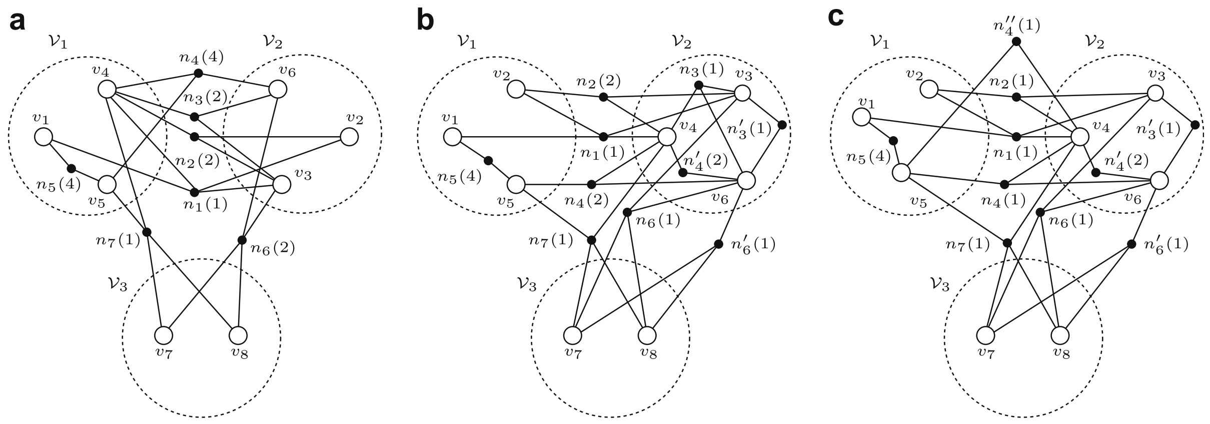


Fig. 4. Clustering hypergraphs (a) $\mathcal{H}_{\text{GS}}$, (b) $\mathcal{H}_{\text{GuPS}}$, and (c) $\mathcal{H}_{\text{GuVS}}$ for the sample road network in Fig. 3 and 3-way vertex partitions of these hypergraphs.

Fig. 3 shows a sample road network with 8 junctions and 17 links. In the figure, squares represent junctions, directed edges represent links, and the values on the links represent the length of these links. Fig. 3 also illustrates a sample query set composed of 4 queries, where each query is shown as a ⟨source, destination⟩ junction pair together with the sequence of processed junctions (query log). For the sake of presentation, in each query, it is assumed that the sequence of processed junctions are the same in the three clustering hypergraph models.

In Fig. 4, the clustering hypergraphs $\mathcal{H}_{\text{GS}}$, $\mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$ are illustrated for the sample road network given in Fig. 3. Fig. 4 also shows sample 3-way vertex partitions of these hypergraphs, where each part can store at most 3 vertices. Each net is named with the id of the junction on which *GS* or *GUS* operations are invoked and net costs are shown in parentheses. If multiple nets are generated for a junction t_i due to $GUS(t_i, \text{Suc}(t_i, U))$ operations with different $\text{Suc}(t_i, U)$, they are marked with apostrophes (e.g., n_4, n'_4, n''_4).

Consider the 3-way partition $\Pi = \{\mathcal{V}_1 = \{v_1, v_4, v_5\},\ \mathcal{V}_2 = \{v_2, v_3, v_6\},\ \mathcal{V}_3 = \{v_7, v_8\}\}$ of $\mathcal{H}_{\text{GS}}$ shown in Fig. 4a. The cut net n_4 with Pins $(n_4) = \{v_4, v_5, v_6\}$ and $\Lambda(n_4) = \{\mathcal{V}_1, \mathcal{V}_2\}$ incurs the cost $c(n_4)(\lambda(n_4) - 1) = 4(2 - 1) = 4$ to the cutsize. Each of the four $GS(t_4)$ operations represented by net n_4 incurs one disk access under the single-page buffer assumption. Since v_4 is in part $\mathcal{V}_1$, $\mathcal{P}_1$ must be the page in the single-page buffer when $GS(t_4)$ operations are invoked. The records r_5 and r_6 corresponding to the successors t_5 and t_6 of t_4 will be accessed in the following order. Since v_5 is also in part $\mathcal{V}_1$, firstly the record r_5 in $\mathcal{P}_1$ will be accessed. Then, since v_6 is in part $\mathcal{V}_2$, page $\mathcal{P}_2$ will be retrieved to replace $\mathcal{P}_1$ in the buffer in order to access record r_6 in $\mathcal{P}_2$. The disk access cost of *GS* operations due to the set $\{n_1, n_2, n_3, n_4, n_6, n_7\}$ of cut nets is $(1 + 2 + 2 + 4 + 2 + 1)(2 - 1) = 12$ since each of these nets has a connectivity of 2.

Consider $\mathcal{H}_{\text{GuPS}}$ shown in Fig. 4b. As seen in Fig. 4b, *GuPS* operations invoked on junction t_4 incur two nets n_4 and n'_4. The net n_4 is generated with Pins= $\{v_4, v_5, v_6\}$ and a cost of 2 since $\text{Suc}(t_4, U) = \{t_5, t_6\}$ in queries $\langle t_2, t_8\rangle$ and $\langle t_4, t_3\rangle$. The net n'_4 is generated with Pins $= \{v_4, v_6\}$ and a cost of 2 since $\text{Suc}(t_4, U) = \{t_6\}$ in queries $\langle t_5, t_6\rangle$ and $\langle t_7, t_6\rangle$.

Consider the 3-way partition $\Pi = \{\mathcal{V}_1 = \{v_1, v_2, v_5\},\ \mathcal{V}_2 = \{v_3, v_4, v_6\},\ \mathcal{V}_3 = \{v_7, v_8\}\}$ of $\mathcal{H}_{\text{GuPS}}$ shown in Fig. 4b. In this partition, n_4 is a cut net with $\lambda(n_4) = 2$ thus incurring the cost of $2(2 - 1) = 2$ to the cutsize, whereas net n'_4 is an internal net of $\mathcal{V}_2$ and hence does not incur any cost to the cutsize. It is clear that $GuPS(t_4, \{t_6\})$ operations represented by net n'_4 do not incur any disk access. Each of the two $GuPS(t_4, \{t_5, t_6\})$ operations represented by net n_4 incurs one disk access under the single-page buffer assumption. Since v_4 is in part $\mathcal{V}_2$, $\mathcal{P}_2$ must be the page in the single-page buffer when $GuPS(t_4, \{t_5, t_6\})$ operations are invoked. The records r_5 and r_6 corresponding to the successors t_5 and t_6 of t_4 will be accessed in the following order. Since v_6 is also in part $\mathcal{V}_2$, firstly the record r_6 in $\mathcal{P}_2$ will be accessed. Then, since v_5 is in part $\mathcal{V}_1$, page $\mathcal{P}_1$ will be retrieved to replace $\mathcal{P}_2$ in the buffer in order to access record r_5 in $\mathcal{P}_1$. In this way, the proposed clustering hypergraph model correctly encapsulates the disk access cost of the *GuPS* operations invoked on junction t_4. Note that if the record-to-page allocation induced by the partition in Fig. 4a is used instead of the one induced by the partition in Fig. 4b, *GuPS* operations invoked on junction t_4 will incur two more disk accesses due to the disposition of records r_2 and r_4 in different pages. The disk access cost of *GuPS* operations due to the set $\{n_1, n_2, n_4, n_6, n'_6, n_7\}$ of cut nets is $(1 + 2 + 2 + > 1 + 1)(2 - 1) + 1(3 - 1) = 9$.

Consider $\mathcal{H}_{\text{GuVS}}$ shown in Fig. 4c. Note that some of the *GuVS* operations do not incur a net since all successors of the respective junctions are already visited during processing a query. For example, in query $\langle t_5, t_6\rangle$, *GuVS* operations invoked on junctions t_2 and t_3 do not incur any net. As seen in Fig. 4c, *GuVS* operations invoked on junction t_4 incur three nets n_4, n'_4, and n''_4. The net n_4 is generated with Pins= $\{v_4, v_5, v_6\}$ and a cost of 1 since $\text{Suc}(t_4, U) = \{t_5, t_6\}$ in query $\langle t_4, t_3\rangle$. The net n'_4 is generated with Pins $= \{v_4, v_6\}$ and a cost of 2 since $\text{Suc}(t_4, U) = \{t_6\}$ in queries $\langle t_5, t_6\rangle$ and $\langle t_7, t_6\rangle$. The net n''_4 is generated with Pins $= \{v_4, v_5\}$ and a cost of 1 since $\text{Suc}(t_4, U) = \{t_5\}$ in query $\langle t_2, t_8\rangle$.

Consider the 3-way partition $\Pi = \{\mathcal{V}_1 = \{v_1, v_2, v_5\},\ \mathcal{V}_2 = \{v_3, v_4, v_6\},\ \mathcal{V}_3 = \{v_7, v_8\}\}$ of $\mathcal{H}_{\text{GuVS}}$ shown in Fig. 4(c). In this partition, n_4 and n''_4 are cut nets with $\lambda(n_4) = \lambda(n''_4) = 2$ thus both incurring the cost of $1(2 - 1) = 1$ to the cutsize, whereas net

n'_4 is an internal net of $\mathcal{V}_1$ and does not incur any cost to the cutsize. It is clear that the two $GuVS(t_4, \{t_6\})$ operations represented by net n'_4 do not incur any disk access. Each $GuVS(t_4, \{t_5, t_6\})$ operation represented by net n_4 incurs one disk access under the single-page buffer assumption as discussed for the $GuPS(t_4, \{t_5, t_6\})$ operation since the record-to-page allocation is the same in Fig. 4b and c. Each $GuVS(t_4, \{t_5\})$ operation represented by net n''_4 incurs one disk access under the single-page buffer assumption. Since v_4 is in part $\mathcal{V}_2$, $\mathcal{P}_2$ must be the page in the single-page buffer when $GuVS(t_4, \{t_5\})$ operations are invoked. Since v_5 is in part $\mathcal{V}_1$, page $\mathcal{P}_1$ will be retrieved to replace $\mathcal{P}_2$ in the buffer in order to access record r_5 in $\mathcal{P}_1$. In this way, the proposed clustering hypergraph model correctly encapsulates the disk access cost of the *GuVS* operations invoked on junction t_4. The disk access cost of *GuVS* operations due to the set $\{n_1, n_2, n_4, n''_4, n_6, n'_6, n_7\}$ of cut nets is $(1+1+1+1+1+1)(2-1)+1(3-1)=8$. Note that the total number of disk accesses is smaller both in $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$ models when compared with $\mathcal{H}_{\text{GS}}$ model since the number of records to be accessed are pruned by the *GuPS* and *GuVS* operations according to the properties of queries.

The performance of the clustering models depends on the assumption that a set of queries in the log can be used to predict the access patterns of upcoming queries. Disk pages can be periodically reorganized to capture the disk access cost of queries using logs from different time windows because of the possible changes in the query patterns for long period of time. Incremental clustering and adaptive reorganization of disk pages according to the new coming queries can be integrated into our model. However, the changes in the query patterns for a short period of time may degrade the overall performance due to the reorganization costs. The scale of the time window in the selection of the queries has a major effect on the performance of the system. Similar to clustered indexes used in the database management systems to improve the performance of the search queries, database tuning via reorganization for better performance is a selective choice in our model. Since a hypergraph for a given query set can be constructed and partitioned in a reasonable time to propose a new allocation, the difference between the expected I/O cost of operations in the current and the new allocations can be computed efficiently. If this performance difference is more than the reorganization cost, then the reorganization is realized.

5. Experimental results

In order to confirm the validity of the proposed successor retrieval operations and associated clustering models, the performance of the proposed *GuPS* operations modeled by $\mathcal{H}_{\text{GuPS}}$ $(GuPS, \mathcal{H}_{\text{GuPS}})$ and *GuVS* operations modeled by $\mathcal{H}_{\text{GuVS}}(GuVS, \mathcal{H}_{\text{GuVS}})$ are compared against *GS* operations modeled by $\mathcal{H}_{\text{GS}}(GS, \mathcal{H}_{\text{GS}})$. The experimental setup is described in Section 5.1. Section 5.2 evaluates the partitioning quality in terms of cutsize, which corresponds to the total number of disk accesses incurred by the successor retrieval operations under the single-page buffer assumption. In Section 5.3, the total number of disk accesses is estimated in query processing through simulations.

5.1. Experimental setup

A wide range of experiments are conducted on four real-life road network datasets. These datasets are collected from US Tiger/Line [33] (Minnesota7 including 7 counties Anoka, Carver, Dakota, Hennepin, Ramsey, Scott, Washington; Sanfrancisco), US Department of Transportation [38] (California Highway Planning Network), and Brinkhoff's data files [7] (SanJoaquin). The self-loops and multi-links in the datasets are eliminated through a preprocessing step. The properties of the preprocessed datasets are given in Table 1. Note that datasets are listed in the order of increasing network size (number of junctions and links).

In the experiments, 8, 16, and 32 bytes are reserved for the coordinates of a junction, junction attributes, and link attributes, respectively. The storage sizes assigned for these parameters are selected in accordance with the earlier proposals and characteristics of the datasets. Note that, as all links in each dataset are bidirectional, the storage saving mentioned in Section 2.1 is utilized (i.e., only the successor list of each junction is stored). The last column of Table 1 shows the total storage size of the network data excluding size of the point-of-interests and index structures. In the table, d_{avg} refers to average junction degree which is equal to the number of bidirectional links per junction.

For query generation, a modified version of the network-based node selection option of Brinkhoff's Network Generator for Moving Objects [6] is used. For each dataset, three synthetic query sets Q_{short}, Q_{medium}, and Q_{long} are generated depending on the shortest path length of the queries. In order to attain a high level network coverage, a different path length and a query count for each dataset and query set (D, Q) pair are determined. Here, the network coverage for a given (D, Q) pair is defined as the ratio of the number of processed links to the total number of links in the network. The path length is set to 1/18, 1/6,

Table 1
Properties of road network datasets (storage size includes only network data).

D	Road network	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{L}\rvert$	d_{avg}	Size (KB)
D1	California HPN	10,141	28,370	2.80	1378
D2	SanJoaquin	17,444	45,974	2.64	2258
D3	Minnesota7	34,222	92,206	2.69	4510
D4	Sanfrancisco	166,558	426,742	2.56	21,067

Table 2
Properties of query sets.

D	Path length	# of operations		# of operations that may incur disk access			
		Queries	*GS*/*GuPS*/*GuVS*	*GuPS*	*GuVS*	% Improvement	
						GuPS	*GuVS*
Q_{short}							
D1	8	7096	713,540	649,990	564,474	8.9	22.9
D2	8	11,701	994,296	814,044	745,016	18.1	30.6
D3	26	18,011	14,510,159	11,801,657	10,371,337	18.7	35.1
D4	27	86,167	125,939,189	95,580,202	84,914,536	24.1	42.9
					Averages	17.5	32.9
Q_{medium}							
D1	25	3909	3,104,899	2,748,033	2,286,093	11.5	29.8
D2	25	5899	4,692,252	3,749,669	3,286,196	20.1	37.5
D3	78	9964	56,685,642	45,116,531	39,096,176	20.4	39.0
D4	81	49,074	581,893,328	433,966,062	381,245,888	25.4	46.2
					Averages	19.4	38.1
Q_{long}							
D1	75	1153	3,310,447	2,880,172	2,389,886	13.0	32.0
D2	76	1759	9,745,309	7,655,299	6,618,883	21.4	40.8
D3	233	3458	66,055,205	51,384,653	44,490,684	22.2	42.0
D4	242	16,505	976,443,708	723,602,253	635,910,853	25.9	47.1
					Averages	20.6	40.5

and 1/2 of the diameter of each network for Q_{short}, Q_{medium}, and Q_{long}, respectively. The number of queries for each (D, Q) pair is selected as follows: Initially, the number of queries is set to 0.5%, 0.3%, and 0.1% of the number of junctions in the network for Q_{short}, Q_{medium}, and Q_{long}, respectively. Each of these queries is repeated 100 times on the average (between 50 and 150 times) to simulate a more realistic case with frequent queries. If the network coverage of these queries remains below 90%, then additional queries are added to have a coverage higher than 90%. These query sets are used both in the construction of the clustering hypergraphs and in the simulations. Table 2 displays the properties of these synthetic query sets.

In Table 2, the number of queries and operations columns refer to the total number of queries and successor retrieval operations invoked for each (D, Q) pair. For a fair comparison among different query processing strategies, the numbers of *GS*, *GuPS*, and *GuVS* operations are enforced to be the same for each (D, Q) pair. As shown in Table 2, for a given query type (i.e., Q_{short}, Q_{medium}, or Q_{long}), the total number of operations increases with increasing network size due to the increase in the path length and the number of queries. Similarly, for a given dataset, the total number of operations increases with increasing path length in Q_{short}, Q_{medium}, and Q_{long} even though the number of queries decreases. This is explained by the properties of the network traversal algorithm used in the Brinkhoff's Network Generator for Moving Objects. In Table 2, the 5th and 6th columns show the number of *GuPS* and *GuVS* operations that may incur disk access (es). The remaining *GuPS* and *GuVS* operations do not incur any disk access, because the set of unevaluated successors for these operations is found to be empty in query processing (i.e., $\text{Suc}(t, U) = \emptyset$). Note that each *GS* operation may incur disk access (es). The last two columns of Table 2 show the percent decrease in the total number of *GuPS* and *GuVS* operations that may incur disk access (es) when compared with the total number of *GS* operations. Each "Averages" row shows the percent improvements for the *GuPS* and *GuVS* operations averaged over all datasets for the respective query type.

As seen in Table 2, the percent improvements of the *GuPS*/*GuVS* operations over the *GS* operations vary significantly between the data and query sets. The query path length and topological properties of road networks such as connectivity and average junction degree are the main factors, which affect the number of *GuPS*/*GuVS* operations that may incur disk access in query processing. During processing a query, the evaluated junctions are expected to appear more frequently in the successor lists of junctions to be evaluated for higher network connectivity and longer query path length, thus decreasing the number of *GuPS*/*GuVS* operations that may incur disk accesses. During processing a query in a highly connected network, the unevaluated successor list of a junction with a smaller degree is more likely to become empty when compared to a junction with a larger degree. It is relatively easier to assess a trend and validate these factors for a fixed dataset. As seen in Table 2, the percent improvement increases considerably with increasing query path length for each dataset. For example, for dataset D1, the percent improvement increases from 8.9% to 11.5% and 13.0% for the query sets Q_{short}, Q_{medium}, and Q_{long}, respectively. However, it is relatively harder to assess such regular trends between different datasets because of the difficulty in the comparison of topological properties of different datasets. For example, although the query path lengths are almost equal for datasets D1 and D2 (due to the very close diameters), the percent improvement in D2 is significantly higher than that in D1. This difference can be attributed to the smaller junction degree 2.64 of D2 compared to 2.80 of D1. A similar argument can be stated for the considerable performance difference between datasets D3 and D4.

Table 3
Properties of generated hypergraphs.

D	$\lvert\mathcal{V}\rvert$	Q_{short}			Q_{medium}			Q_{long}		
		$\lvert\mathcal{N}_{\text{GS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$
D1	10,141	10,134	38,495	3.8	10,136	38,500	3.8	10,137	38,502	3.8
D2	17,444	17,366	63,236	3.6	17,351	63,181	3.6	17,279	62,926	3.6
D3	34,222	33,723	125,103	3.7	33,383	124,082	3.7	33,451	124,288	3.7
D4	166,558	166,152	592,183	3.6	166,212	592,327	3.6	165,850	591,150	3.6
		$\lvert\mathcal{N}_{\text{GuPS}}\rvert$	$\lvert\mathcal{H}_{\text{GuPS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuPS}}\rvert$	$\lvert\mathcal{H}_{\text{GuPS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuPS}}\rvert$	$\lvert\mathcal{H}_{\text{GuPS}}\rvert$	$\lvert n\rvert_{\text{avg}}$
D1	10,141	35,682	103,912	2.9	36,287	102,855	2.8	32,242	89,230	2.8
D2	17,444	51,118	150,450	2.9	50,712	144,007	2.8	43,497	120,550	2.8
D3	34,222	92,806	265,192	2.9	79,731	222,837	2.8	63,047	176,108	2.8
D4	166,558	408,021	1,139,808	2.8	369,094	1,015,144	2.8	332,092	910,504	2.7
		$\lvert\mathcal{N}_{\text{GuVS}}\rvert$	$\lvert\mathcal{H}_{\text{GuVS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuVS}}\rvert$	$\lvert\mathcal{H}_{\text{GuVS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuVS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$
D1	10,141	35,544	99,437	2.8	34,288	91,279	2.7	28,754	72,789	2.5
D2	17,444	49,697	141,767	2.9	45,695	123,424	2.7	37,155	95,684	2.6
D3	34,222	77,829	207,891	2.7	64,546	165,548	2.6	51,067	128,760	2.5
D4	166,558	365,759	964,027	2.6	319,967	819,010	2.6	287,675	730,615	2.5

As seen in Table 2, the percent improvement in the number of operations that may incur disk access (es) is significantly greater for queries utilizing *GuVS* operations compared to those utilizing *GuPS* operations. This is due to the fact that, in the incremental network expansion framework, the size of the visited junction set grows much faster when compared to the size of the processed junction set in queries utilizing Dijkstra's single source shortest path algorithm.

Table 3 shows the properties of the generated clustering hypergraphs for *GS*, *GuPS*, and *GuVS* operations. In the table, $|\mathcal{V}|,\ |\mathcal{N}|,\ |\mathcal{H}|$, and $|n|_{\text{avg}}$ denote the number of vertices, nets, pins and the average net degree of hypergraphs, respectively. Recall that, for a given dataset, the numbers of vertices of the three clustering hypergraphs $\mathcal{H}_{\text{GS}},\ \mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$ are the same for all three query sets. As mentioned in Section 2.4, in $\mathcal{H}_{\text{GS}}$, there exists a single net for each junction on which a *GS* operation is invoked. However, in Table 3, for each (D,Q) pair, the number of nets is slightly less than the number of junctions since the network coverage of queries can be less than 100% (between 90% and 100%). In $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$, there might be multiple nets for each junction on which a *GuPS* and a *GuVS* operation is invoked with distinct set of unevaluated successors, respectively. For each (D,Q) pair, the average number of nets per junction remains below 3.50 and 3.58, and on the overall average it is 2.40 and 2.68 for $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$, respectively. On the overall average, the size (total number of pins) of $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$ is 2.06 and 1.74 times that of $\mathcal{H}_{\text{GS}}$, respectively. Thus, the additional complexity of the hypergraph due to the increase in the number of nets is moderate.

The constructed hypergraphs are partitioned using the recursive bipartitioning paradigm discussed in Section 4.2. For this purpose, the state-of-the-art multi-level hypergraph partitioning tool PaToH [11] is used for bipartitioning the hypergraphs [15]. Experimental results are conducted on a PC with a 2.66 GHz Intel Xenon processor and 4 GB of RAM. The average in-memory partitioning times for the largest dataset D4 are 6.3, 10.6, and 8.7 seconds for the $\mathcal{H}_{\text{GS}},\ \mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$ hypergraphs, respectively. The small amount of increase in the partitioning times for the $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$ models compared to that of $\mathcal{H}_{\text{GS}}$ model comply with the moderate increase in the size of $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$ hypergraphs compared to that of the $\mathcal{H}_{\text{GS}}$ hypergraph.

The partitioning of a clustering hypergraph representation for a (D,Q) pair and a given page size is referred to here as a record-to-page allocation instance. Experiments are carried out with four page sizes of $P = 4, 8, 16$, and 32 KB. So, the total number of allocation instances using $Q_{\text{short}},\ Q_{\text{medium}}$ and Q_{long} query sets is equal to $3 \times 4 \times 3 \times 4 = 144$. As PaToH use randomized algorithms, the experiment for each data allocation instance is repeated 10 times and the average partitioning quality results are reported in Section 5.2.

In simulating the query processing, the caching effect is evaluated with a page buffer using the least recently used (LRU) page replacement algorithm. Simulations are carried out with four buffer sizes of B = 1, 2, 4, and 8 pages where only a small portion of a dataset resides in-memory. So, the total number of simulation instances is equal to $4 \times 144 = 576$. As 10 results are generated for each allocation instance, each simulation instance is also performed 10 times and average results are reported in Section 5.3.

Table 4
Number K of pages (in ranges) for all allocation instances.

P	D1	D2	D3	D4
4	[368,370]	[607,609]	[1212,1216]	[5652,5658]
8	[184,186]	[303,305]	[606,608]	[2830,2834]
16	[91,93]	[151,153]	[303,305]	[1414,1418]
32	[46,48]	[75,77]	[151,153]	[705,709]

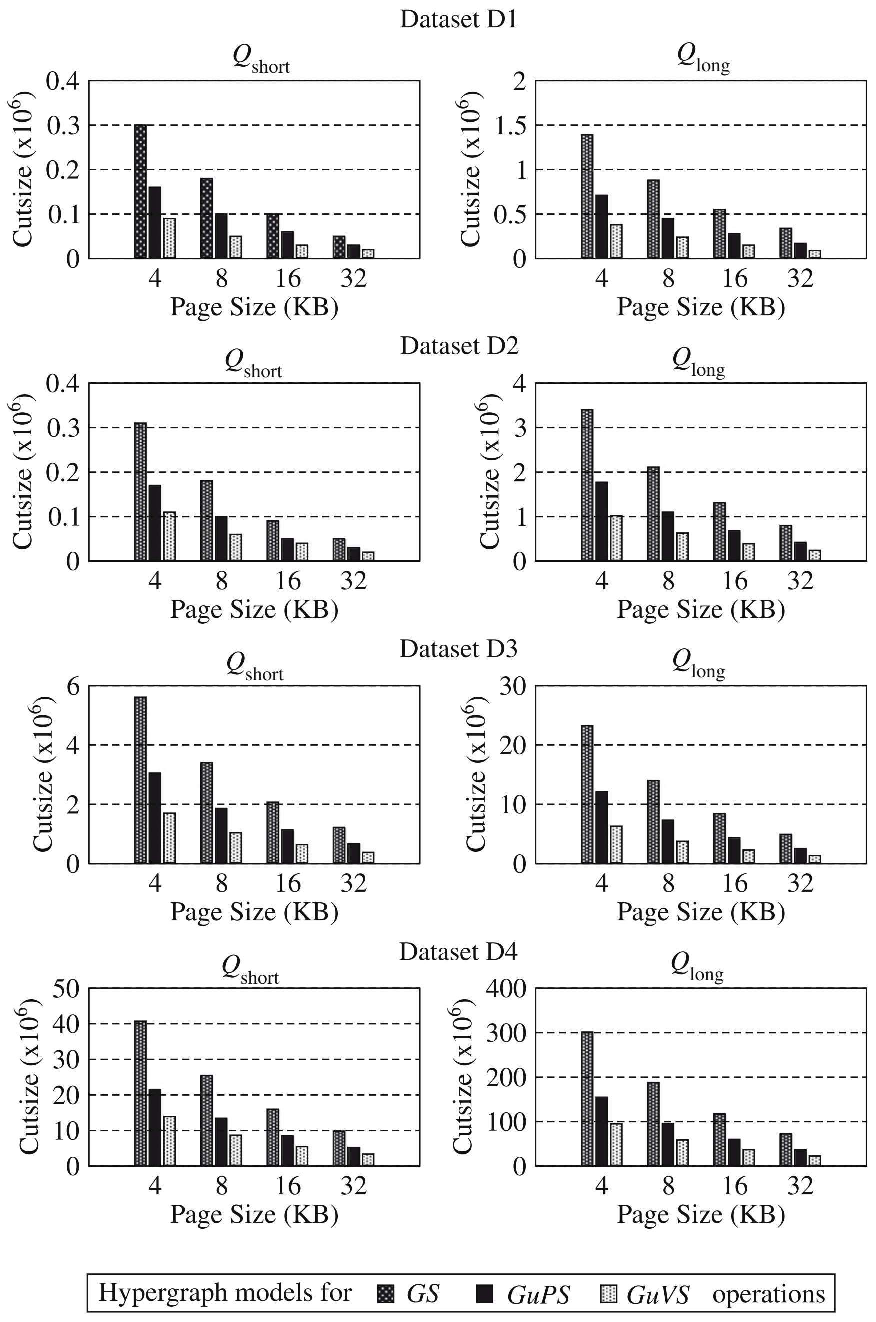


Fig. 5. Partitioning quality of clustering hypergraph models for *GS*, *GuPS*, and *GuVS* operations. Cutsize is equal to the total number of disk accesses due to the respective successor retrieval operation under the single-page buffer assumption.

Table 5
Averages for percent cutsize improvements of $(GuPS, \mathcal{H}_{\mathrm{GuPS}})$ and $(GuVS, \mathcal{H}_{\mathrm{GuVS}})$ over $(GS, \mathcal{H}_{\mathrm{GS}})$.

P	$(GuPS, \mathcal{H}_{\mathrm{GuPS}})$			$(GuVS, \mathcal{H}_{\mathrm{GuVS}})$		
	Q_{short}	Q_{medium}	Q_{long}	Q_{short}	Q_{medium}	Q_{long}
4	46.1	47.2	48.4	67.3	69.1	70.9
8	45.5	47.0	48.2	67.3	69.1	71.0
16	44.7	46.8	48.4	66.3	68.7	70.9
32	45.0	46.8	48.5	65.5	68.3	70.7
Average	45.3	46.9	48.4	66.6	68.8	70.9

In our simulations, for each network query, it is assumed that records are accessed through a sequence of *Find* and successor retrieval operation pairs, i.e., *Find*, *GS*/*GUS*, ..., *Find*, *GS*/*GUS*, ... Here, the *Find* operations are selectively performed only if the record is not found in the current page buffer. A B+ tree with *Z*-ordering is used for efficient support of *Find* operations as discussed in [32]. The lookup cost of this index for *Find* operations is included in our simulation results showing the total number of disk accesses for query processing.

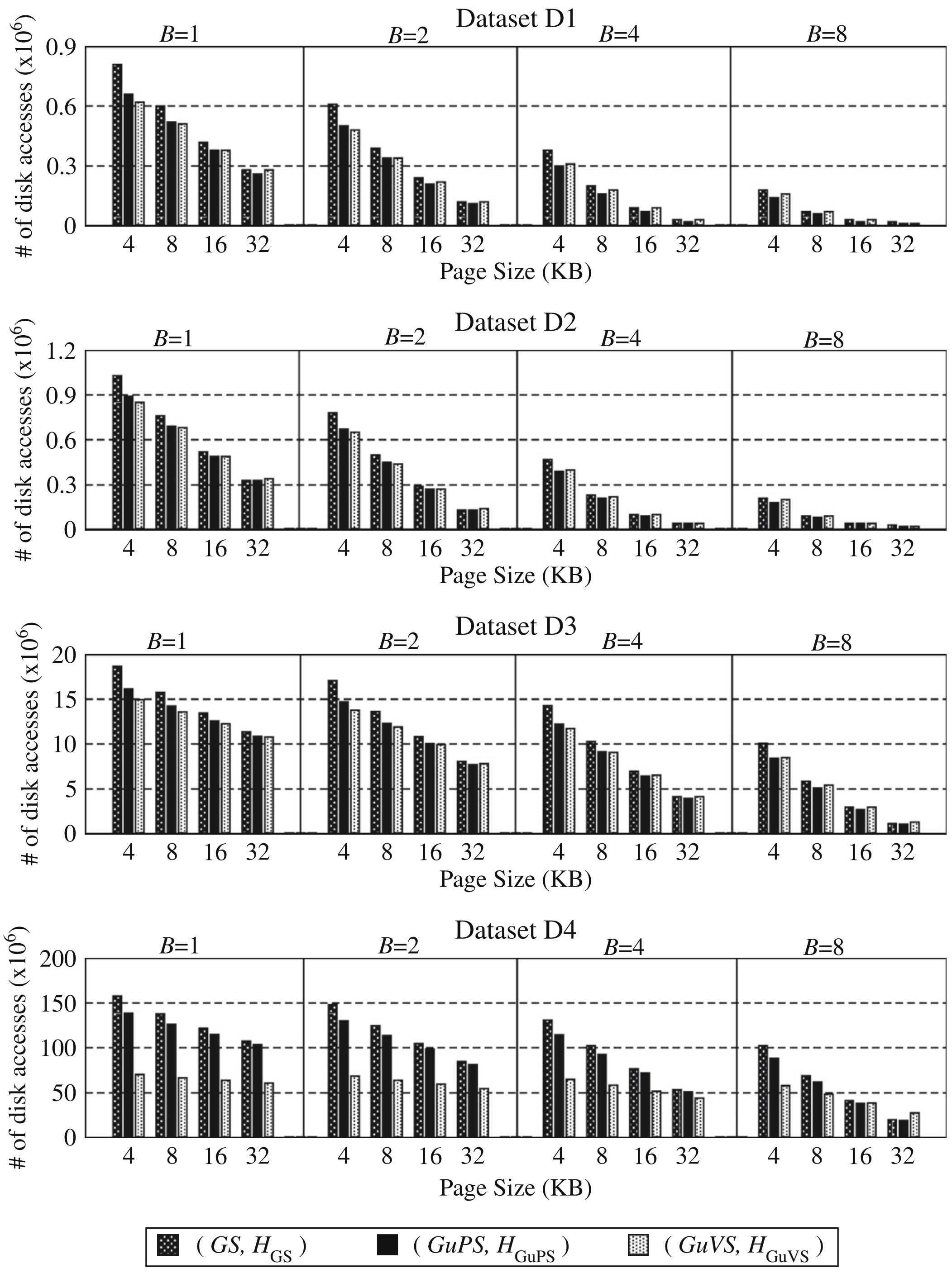


Fig. 6. Total disk access cost of $(GS, \mathcal{H}_{GS})$, $(GuPS, \mathcal{H}_{GuPS})$, and $(GuVS, \mathcal{H}_{GuVS})$ models in query simulations using different page size *P* in KB and buffer size *B* in number of pages for Q_{short} query set.

5.2. Partitioning quality

For a given dataset and a page size, the number K of disk pages allocated either changes very slightly or does not change at all for different query sets and successor retrieval operations. In Table 4, K value ranges are reported for each dataset and page size pairs. As seen in Table 4, for each dataset, the number of allocated pages decreases linearly with increasing page size as expected.

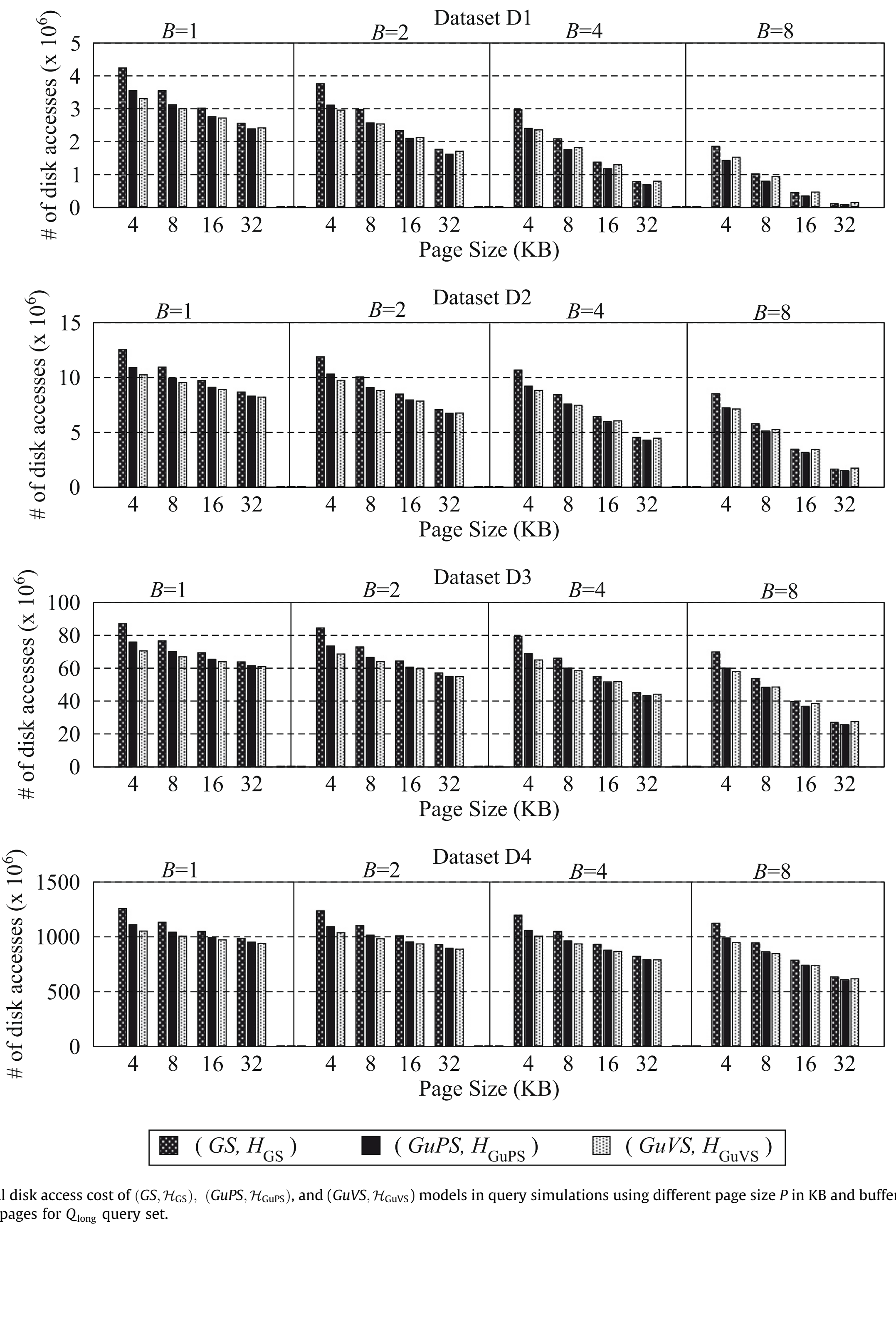


Fig. 7. Total disk access cost of $(GS, \mathcal{H}_{GS})$, $(GuPS, \mathcal{H}_{GuPS})$, and $(GuVS, \mathcal{H}_{GuVS})$ models in query simulations using different page size P in KB and buffer size B in number of pages for Q_{long} query set.

Fig. 5 displays the partitioning quality of clustering hypergraph models for *GS*, *GuPS*, and *GuVS* operations in terms of cutsize for the Q_{short} and Q_{long} query sets and for different page sizes. In all allocation instances, both $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ achieve significantly smaller cutsize values than $(GS, \mathcal{H}_{GS})$. As seen in Fig. 5, the cutsize values decrease with increasing page size since the number of records that can be packed in a page increases.

Table 5 shows the average cutsize improvement of $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ over $(GS, \mathcal{H}_{GS})$ for the Q_{short}, Q_{medium}, and Q_{long} query sets. As seen in the table, for a fixed query set, the performance gaps between $(GS, \mathcal{H}_{GS})$ and the other two models do not vary considerably with increasing page size. On the other hand, for a fixed page size, the performance gaps slightly increase in favor of $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ as the query set changes from Q_{short} to Q_{long}. This can be explained by the expected increase in the number of evaluated junctions in the successor lists of the junctions to be evaluated with increasing query length as discussed in Section 5.1.

Recall that the cutsizes obtained by the clustering hypergraph models $(GS, \mathcal{H}_{GS})$, $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ correspond to the total number of disk accesses incurred by the respective successor retrieval operations under the single-page buffer assumption. As seen in Table 5, $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ achieve 46.9% and 68.8% cutsize improvement over $(GS, \mathcal{H}_{GS})$, on the overall average. A part of these improvements relates to the 19.1% and 37.2% decrease in the number of *GuPS* and *GuVS* operations that may incur disk accesses as shown in Table 2. So, the significant part of these improvements comes from the correct modeling of the *GuPS* and *GuVS* operations by the $\mathcal{H}_{GuPS}$ and $\mathcal{H}_{GuVS}$ models, respectively. Significantly smaller cutsizes obtained by the $\mathcal{H}_{GuPS}$ and $\mathcal{H}_{GuVS}$ over those obtained by $\mathcal{H}_{GS}$ can be explained as follows: multiple nets of size smaller than the junction degree used by the $\mathcal{H}_{GuPS}$ and $\mathcal{H}_{GuVS}$ models for each junction, in contrast to a single net of size equal to the junction degree used by the $\mathcal{H}_{GS}$ model, provide a flexibility to the hypergraph partitioning tool in removing more nets from the cut in the $\mathcal{H}_{GuPS}$ and $\mathcal{H}_{GuVS}$ models compared to the $\mathcal{H}_{GS}$ model.

5.3. Disk access simulations

Figs. 6 and 7 compare the performance of $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ over $(GS, \mathcal{H}_{GS})$ in terms of total number of disk accesses for the Q_{short} and Q_{long} query sets, respectively. The values displayed in Figs. 6 and 7 show the number of disk accesses incurred by the successor retrieval operations as well as those incurred by the *Find* operations in query processing. Table 6 shows the average percent performance improvement of $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ over $(GS, \mathcal{H}_{GS})$ over all datasets.

As seen in Figs. 6 and 7, $(GuPS, \mathcal{H}_{GuPS})$ performs considerably better than $(GS, \mathcal{H}_{GS})$ in all simulation instances, whereas $(GuVS, \mathcal{H}_{GuVS})$ performs better than $(GS, \mathcal{H}_{GS})$ in all but 10 out of 128 simulation instances. This is because of the fact that the disk access cost due to the *Find* operations constitutes a much larger portion of the total disk access cost in the $(GuVS, \mathcal{H}_{GuVS})$ scheme when compared to the other two schemes since the number of disk accesses incurred by the *GuVS* operations are much less than those incurred by the *GS* and *GuPS* operations. Recall that although the clustering hypergraph models $\mathcal{H}_{GS}$, $\mathcal{H}_{GuPS}$, and $\mathcal{H}_{GuVS}$ capture the exact cost of disk accesses to be incurred by the respective successor retrieval operations under the single-page buffer assumption, they do not capture the cost of disk accesses to be incurred by the *Find* operations. The percent performance averages in Table 6 also confirm this finding. As seen in Table 6, $(GuVS, \mathcal{H}_{GuVS})$ performs better than $(GS, \mathcal{H}_{GS})$ in all but 4 out of 48 cases where these performance changes only occur for the large page and buffer size values. Furthermore, comparison of Tables 5 and 6 shows that average percent performance improvements in simulation results are considerably less than average cutsize improvements. In order to further clarify this issue, Table 7 is introduced to display the average percent performance improvements in terms of disk accesses only due to the successor retrieval

Table 6
Averages for percent performance improvements of $(GuPS, \mathcal{H}_{GuPS})$ and $(GuVS, \mathcal{H}_{GuVS})$ over $(GS, \mathcal{H}_{GS})$ in terms of total number of disk accesses.

B	*P*	$(GuPS, \mathcal{H}_{GuPS})$			$(GuVS, \mathcal{H}_{GuVS})$		
		Q_{small}	Q_{medium}	Q_{long}	Q_{small}	Q_{medium}	Q_{long}
1	4	14.5	13.6	13.4	19.4	19.2	18.9
	8	10.4	9.8	9.5	13.0	13.4	13.0
	16	6.9	6.9	6.5	7.7	8.7	8.4
	32	3.6	4.7	4.5	1.8	5.5	5.0
2	4	14.8	14.0	13.8	18.5	18.8	18.5
	8	10.8	10.3	10.0	12.2	12.9	12.6
	16	7.1	7.3	7.1	6.6	8.0	7.7
	32	5.5	4.8	5.0	−0.5	4.5	3.9
4	4	16.0	14.7	14.5	16.6	18.2	18.0
	8	12.1	11.2	10.9	8.7	11.8	11.7
	16	8.2	8.3	8.3	2.0	6.2	6.2
	32	5.5	5.4	6.5	0.7	1.8	1.6
8	4	17.7	16.4	16.1	12.5	16.6	16.6
	8	12.4	13.0	12.9	4.8	8.6	9.1
	16	8.3	10.1	10.9	2.6	1.1	1.4
	32	7.7	7.9	9.7	−0.2	−3.6	−7.7

Table 7
Averages for percent performance improvements of $(GuPS, \mathcal{H}_{\mathrm{GuPS}})$ and $(GuVS, \mathcal{H}_{\mathrm{GuVS}})$ over $(GS, \mathcal{H}_{\mathrm{GS}})$ in terms of the number of disk accesses incurred only by the successor retrieval operations.

B	P	$(GuPS, \mathcal{H}_{\mathrm{GuPS}})$			$(GuVS, \mathcal{H}_{\mathrm{GuVS}})$		
		Q_{small}	Q_{medium}	Q_{long}	Q_{small}	Q_{medium}	Q_{long}
1	4	46.8	47.3	48.4	68.2	69.2	70.9
	8	46.4	47.0	48.2	68.3	69.2	71.0
	16	45.8	46.9	48.4	67.5	68.8	70.9
	32	46.4	46.9	48.5	66.9	68.5	70.8
2	4	44.4	46.7	48.1	65.5	68.3	70.3
	8	42.9	46.1	47.7	64.3	68.0	70.3
	16	40.9	45.4	47.8	61.6	67.0	70.0
	32	39.9	44.4	47.5	58.1	65.8	69.5
4	4	40.7	45.6	47.5	59.7	66.6	69.1
	8	37.7	44.5	46.8	55.9	65.5	68.8
	16	33.4	42.7	46.7	48.8	63.0	68.0
	32	30.8	39.8	45.9	41.8	59.4	66.4
8	4	34.5	43.4	46.2	48.2	62.4	66.5
	8	28.8	40.6	44.9	40.0	58.7	64.9
	16	24.0	36.1	43.7	31.7	51.7	61.9
	32	22.4	29.5	39.7	27.3	42.3	54.9

operations in simulations. Comparison of Tables 5 and 7 shows that percent performance improvements for all simulations are almost the same as in the cutsize improvements for the single-page buffer case, and very close for the larger buffer sizes. These experimental findings confirm the validity of the proposed clustering hypergraph models $\mathcal{H}_{\mathrm{GuPS}}$ and $\mathcal{H}_{\mathrm{GuVS}}$ for the *GuPS* and *GuVS* successor retrieval operations.

According to Figs. 6 and 7, as expected, the number of disk accesses decreases with increasing page size and increasing buffer size in all simulation instances. Comparison of Figs. 6 and 7 show that the decrease in the number of disk accesses is more prominent with the Q_{short} query set compared with the Q_{long} query set. As seen in Table 7, for fixed page and buffer sizes, the performance improvement of both $(GuPS, \mathcal{H}_{\mathrm{GuPS}})$ and $(GuVS, \mathcal{H}_{\mathrm{GuVS}})$ over $(GS, \mathcal{H}_{\mathrm{GS}})$, in terms of the disk access cost due to the successor retrieval operations, slightly increase with increasing query length. However, as seen in Table 6, it is hard to find any such trend for the total disk access cost because of the additional disk accesses incurred by the *Find* operations.

The simulation results show that, during aggregate query processing, *GUS* is a crucial kernel successor retrieval operation in order to decrease the number of disk accesses. In queries utilizing the Dijkstra's single shortest path algorithm, *GuPS* operations still constitute a considerable portion of the total disk accesses. Hence, with the utilization of the clustering hypergraph $\mathcal{H}_{\mathrm{GuPS}}$ model, a significant improvement in the total number of disk accesses can be achieved using the resulting record-to-page allocation. However, in queries utilizing the incremental network expansion framework, the percent of disk accesses due to the *GuVS* operations is considerably less than that due to the *Find* operations because of the significant reduction in the number of disk accesses due to the *GuVS* operations. Thus, even though the clustering hypergraph $\mathcal{H}_{\mathrm{GuVS}}$ model results in significantly better record-to-page allocations for successor retrieval operations, the effectiveness of this model may degrade with the usage of larger page and buffer sizes because the percent improvement in disk accesses due to the *Find* operations becomes prominent due to the buffering effect, where $\mathcal{H}_{\mathrm{GS}}$ model seems to give better clustering for *Find* operations. Further research is needed for encapsulating the disk access cost of *Find* operations especially for algorithms using the incremental network expansion framework.

6. Concluding remarks

We introduced a new successor retrieval operation, *Get-Unevaluated-Successors* (*GUS*), for spatial network databases and focused on the problem of record-to-page data allocation in road networks in order to minimize the disk access cost of *GUS* operations in query processing.

The *GUS* operation is an efficient implementation of the *Get-Successors* (*GS*) operation, where the candidate successors to be retrieved are pruned according to the properties and state of the search algorithm used in the target application. Two examples of *GUS* operation are introduced in network query processing, namely the *Get-unProcessed-Successors* (*GuPS*) operation as used in the Dijsktra's single source shortest path algorithm [19] and the *Get-unVisited-Successors* (*GuVS*) operation as used in the algorithms utilizing the incremental network expansion framework [29].

We proposed a clustering hypergraph model to allocate network data to disk pages, where data would be periodically reorganized using query logs. Our model exactly captures the disk access cost of *GUS* operations in network queries under the single-page buffer assumption. Extensive experiments are conducted to show the effects of dataset, query set, page size, and buffer size through simulations. Experimental results show that both *GuPS* and *GuVS* operations lead to a significant improvement in query processing and the corresponding clustering hypergraph models achieve better results than earlier solutions for the record-to-page allocation problem in road networks.

In spatial network database management systems, the *GUS* operation can be implemented as a kernel successor retrieval operation. Data clustering is already studied in the database literature and most of the enterprise database management systems utilize clustered indexes to improve the I/O performance of the system during query processing. Hence, the proposed clustering hypergraph models can easily be deployed into these database management systems so that aggregate query processing can be performed efficiently in any GIS system using these data sources.

Acknowledgements

Some experiments were carried out on TR-Grid e-Infrastructure. This work is partially supported by the Scientific and Technological Research Council of Turkey under Grant EEEAG-109E019.

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, B.B. Cambazoglu, B. Ucar, Multi-level direct *k*-way hypergraph partitioning with multiple constraints and fixed vertices, Journal of Parallel and Distributed Computing 68 (5) (2008) 609–625.
[4] 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.
[5] C. Berge, Graphs and Hypergraphs, North-Holland Publishing Company, 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 6th SIAM Conference on Parallel Processing for Scientific Computing, 1993, pp. 445–452.
[9] B.B. Cambazoglu, C. Aykanat, Hypergraph-partioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids, IEEE Transactions on Parallel and Distributed Systems 18 (1) (2007) 3–16.
[10] Ü.V. Çatalyürek, C. Aykanat, Hypergraph-partitioning-based decomposition of parallel sparse-matrix vector multiplication, IEEE Transactions on Parallel 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] E.P. Chan, H. Lim, Optimization and evaluation of shortest path queries, The VLDB Journal 16 (3) (2007) 343–369.
[13] A. Dasdan, C. Aykanat, Two novel multiway circuit partitioning algorithms using relaxed locking, IEEE Transactions on Computer-Aided Design Integrated Circuits and Systems 16 (2) (1997) 169–178.
[14] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A^*, Journal of ACM 32 (3) (1985) 505–536.
[15] E. Demir, C. Aykanat, B.B. Cambazoglu, Clustering spatial networks for aggregate query processing: a hypergraph approach, Information Systems 33 (1) (2008) 1–17.
[16] E. Demir, C. Aykanat, B.B. Cambazoglu, A link-based storage scheme for efficient aggregate query processing on clustered road networks, Information Systems 35 (1) (2010) 75–93.
[17] K. Deng, X. Zhau, H.T. Shen, S. Sadiq, X. Li, Instance optimal query processing in spatial networks, The VLDB Journal (2008). doi:10.1007/s00778-008-0115-0.
[18] Y. Deng, Exploiting the performance gains of modern disk drives by enhancing data locality, Information Sciences 179 (14) (2009) 2494–2511.
[19] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1 (1959) 269–271.
[20] 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.
[21] N. Jing, Y.-W. Huang, E.A. Rundensteiner, Hierarchical encoded path views for path query processing: an optimal model and its performance evaluation, IEEE Transaction on Knowledge and Data Engineering 10 (3) (1998) 409–432.
[22] S. Jung, S. Pramanik, An efficient path computation model for hierarchically structured topographical road maps, IEEE Transaction on Knowledge and Data Engineering 14 (5) (2002) 1029–1046.
[23] G. Karypis, R. Aggarwal, V. Kumar, S. Shekhar, Multilevel hypergraph partitioning: applications in VLSI domain, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 7 (1) (1999) 69–79.
[24] M. Kolahdouzan, C. Shahabi, Voronoi-based *K* nearest neighbor search for spatial network databases, in: Proceedings of the 30th International Conference on Very Large Data Bases, 2004, pp. 840–851.
[25] M. Koyuturk, C. Aykanat, Iterative-improvement based declustering heuristics for multi-disk databases, Information Systems 30 (1) (2005) 47–70.
[26] A.J.T. Lee, Y.-A. Chena, W.-C. Ip, Mining frequent trajectory patterns in spatial–temporal databases, Information Sciences 179 (13) (2009) 2218–2231.
[27] T. Lengauer, Combinatorial Algorithms for Integrated Circuit Layout, John Wiley & Sons, Inc., New York, NY, USA, 1990.
[28] M. Ozdal, C. Aykanat, Hypergraph models and algorithms for data-pattern based clustering, Data Mining and Knowledge Discovery 9 (1) (2004) 29–57.
[29] 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.
[30] J. Sankaranarayanan, H. Alborzi, H. Samet, Efficient query processing on spatial networks, in: Proceedings of the 13th ACM International Workshop on Geographic Information Systems, 2005, pp. 200–209.
[31] C. Shahabi, M.R. Kolahdouzan, M. Sharifzadeh, A road network embedding technique for *k*-nearest neighbor search in moving object databases, in: Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems, 2002, pp. 94–100.
[32] S. Shekhar, D.R. Liu, A connectivity-based access method for networks and network computation, IEEE Transaction on Knowledge and Data Engineering 9 (1) (1997) 102–117.
[33] Topologically integrated geographic encoding and referencing system (TIGER), 2002 <http://www.census.gov/geo/www/tiger/>.
[34] B. Ucar, C. Aykanat, Partitioning sparse matrices for parallel preconditioned iterative methods, SIAM Journal on Scientific Computing 29 (4) (2007) 1683–1709.
[35] B. Ucar, C. Aykanat, M. Pınar, T. Malas, Parallel image restoration using surrogate constraint methods, Journal of Parallel and Distributed Computing 67 (2) (2007) 186–204.
[36] B. Uçar, 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.
[37] B. Uçar, C. Aykanat, Revisiting hypergraph models for sparse matrix partitioning, SIAM Review 49 (4) (2007) 595–603.

[38] US department of transportation federal highway administration, the national highway planning network, 2004 <http://www.fhwa.dot.gov/planning/nhpn/index.html>.
[39] S.-H. Woo, S.-B. Yang, An improved network clustering method for I/O-efficient query processing, in: Proceedings of the ACM Symposium on Geographic Information Systems, 2000, pp. 62–68.
[40] 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.
[41] M.L. Yiu, N. Mamoulis, D. Papadias, Aggregate nearest neighbor queries in road networks, IEEE Transaction on Knowledge and Data Engineering 17 (6) (2005) 820–833.