STORAGE AND ACCESS SCHEMES FOR AGGREGATE QUERY PROCESSING ON ROAD NETWORKS

A DISSERTATION SUBMITTED TO

THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Engin Demir

January, 2009

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Cevdet Aykanat (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Enis Çetin

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. Nihan Kesim Çiçekli

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

STORAGE AND ACCESS SCHEMES FOR AGGREGATE QUERY PROCESSING ON ROAD NETWORKS

Engin Demir
Ph.D. in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
January, 2009

A well-known example of spatial networks is road networks, which form an integral part of many geographic information system applications, such as location-based services, intelligent traveling systems, vehicle telematics, and location-aware advertising. In practice, road network data is too large to fit into the volatile memory. A considerable portion of the data must be stored on the secondary storage since several spatial and non-spatial attributes as well as points-of-interest data are associated with junctions and links. In network query processing, the spatial coherency that exists in accessing data leads to a temporal coherency; in this way, connected junctions are accessed almost concurrently. Taking this fact into consideration, it seems reasonable to place the data associated with connected junctions in the same disk pages so that the data can be fetched to the memory with fewer disk accesses. We show that the state-of-the-art clustering graph model for allocation of data to disk pages is not able to correctly capture the disk access cost of successor retrieval operations. We propose clustering models based on hypergraph partitioning, which correctly encapsulate the spatial and temporal coherency in query processing via the utilization of query logs in order to minimize the number of disk accesses during aggregate query processing. We introduce the link-based storage scheme for road networks as an alternative to the widely used junction-based storage scheme. We present *Get-Unevaluated-Successors* (*GUS*) as a new successor retrieval operation for network queries, where the candidate successors to be retrieved are pruned during processing a query. We investigate two different instances of *GUS* operation: the *Get-unProcessed-Successors* operation typically arises in Dijkstra's single source shortest path algorithm, and the *Get-unVisited-Successors* operation typically arises in the incremental network expansion framework. The simulation results

show that our storage and access schemes utilizing the proposed clustering hypergraph models are quite effective in reducing the number of disk accesses during aggregate query processing.

Keywords: Spatial network databases, road networks, storage management, query processing, clustering, hypergraph partitioning.

ÖZET

YOL AĞLARI ÜZERİNDEKİ TOPAK SORGU İŞLEME İÇİN DEPOLAMA VE ERİŞİM PLANLARI

Engin Demir
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Ocak, 2009

İyi bilinen örnek bir uzamsal ağ olan yol ağları, bölge bazlı servisler, akıllı yolculuk sistemleri, araç telematikleri, ve bölgeye duyarlı reklamcılık gibi coğrafi bilgi sistemi uygulamalarının temel bir parçasını oluşturmaktadır. Uygulamada, yol ağ verileri geçici belleğe sığamayacak kadar büyüktür. Hem çeşitli uzamsal ve uzamsal olmayan özellikler, hem de ilgi çekici noktalar kavşak ve yollarla ilişkilendirildiğinden, verinin oldukça büyük bir parçası ikincil bellekte saklanmalıdır. Ağ sorgusu işlemede, veri erişimi sırasındaki uzamsal tutarlılık zamansal tutarlılığa neden olmaktadır; böylece, bağlı kavşaklara neredeyse eşzamanlı erişilmektedirler. Bu gerçeği gözönünde bulundurarak, bağlı kavşakların verilerini aynı disk bölümleri içerine yerleştirilmesi mantıklı görünmektedir ki veriler daha az sayıda disk erişimi ile getirilebilsin. Şu andaki verileri disk bölümlerine kümeleyen çizge modelinin ardıl getirme operasyonlarının disk erişim maliyetini doğru yakalamadığını gösteriyoruz. Topak sorgu işleme sırasındaki disk erişimini en aza indirmek için sorgu işlemede uzamsal ve zamansal tutarlılığı sorgu kayıtlarını kullanarak doğru kapsayan hiperçizge bölümlemeye dayalı kümeleme modelleri öneriyoruz. Yol ağları için yola dayalı depolama planınını yaygın kullanılan kavşağa dayalı depolama planına alternatif olarak tanıtıyoruz. Ağ sorgularında getirilecek aday ardılları sorgu işleme sırasında azaltacak yeni bir ardıl getirme işlemi olarak *Değerlendirilmemiş-Ardılları-Getir*'i sunuyoruz. İki değişik *Değerlendirilmemiş-Ardılları-Getir* işlem örneğini inceliyoruz: *İşlenmemiş-Ardılları-Getir* işlemi tipik olarak Dijkstra'nın tek başlangıç kısayol çözüm yolunda ve *Görülmemiş-Ardılları-Getir* işlemi tipik olarak atrımlı ağ genişleme taslağında ortaya çıkıyor. Benzetim sonuçları kümeleyen hiperçizge modellerini kullanan depolama ve erişim planlarımızın topak sorgu işleme sırasındaki disk erişim sayısını azaltmakta oldukça etkili olduğunu göstermektedir.

Anahtar sözcükler: Uzamsal ağ veritabanları, yol ağları, bellek yönetimi, sorgu işleme, sınıflandırma, hiperçizge bölümleme.

Acknowledgement

I would like to express my gratitude to Prof. Dr. Cevdet Aykanat, from whom I have learned a lot, due to his supervision, suggestions, and support during this research. I thank Berkant Barla Cambazoğlu for his contributions in the thesis.

I am also indebted to Prof. Dr. Enis Çetin, Assoc. Prof. Dr. Nihan Kesim Çiçekli, Assoc. Prof. Dr. Uğur Güdükbay, and Prof. Dr. Özgür Ulusoy for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to thank to the former and current members of Computer Engineering Department, who turned the department into a lovely place. I owe my warmest thanks to my colleagues İsmail Sengör Altıngövde, Berkant Barla Cambazoğlu, Kamer Kaya, Ata Türk, and Funda Durupınar.

Above all, I am deeply thankful to my family members for their everlasting encouragement and support.

To the memory of my dad...

Contents

1 Introduction **2**

1.1 Motivation 2

1.2 Contributions 4

2 Background and Related Work **7**

2.1 Road Networks 7

2.2 Query Processing on Road Networks 7

2.2.1 Nearest Neighbor Search 10

2.2.2 Advanced Nearest Neighbor Search 12

2.3 Kernel Set of Database Operations 13

2.4 Processing Aggregate Network Queries 14

2.5 Data Allocation Problem 15

2.6 Graph and Hypergraph Partitioning 16

2.7 Disk-Based Network Representations 18

3 Junction-Based Storage Scheme **21**

3.1 Definition 22
3.2 Clustering Graph Model 22
3.2.1 Graph Representation 22
3.2.2 Graph Model 23
3.2.3 Deficiencies of Clustering Graph Model 23
3.3 Clustering Hypergraph Model 25
3.3.1 Hypergraph Construction 25
3.3.2 Hypergraph Model 27
3.4 Recursive Graph/Hypergraph Bipartitioning Schemes 30
3.4.1 Determining Number of Pages 31
3.4.2 Packing Lightly Loaded Parts 32
3.5 Summary 32

4 Link-Based Storage Scheme **34**

4.1 Definition 35
4.2 Comparison of Storage Schemes 35
4.3 Auxiliary Index Structures 41
4.4 Clustering Hypergraph Model 42
4.4.1 Hypergraph Construction 42
4.4.2 Hypergraph Model 44
4.4.3 Comparison of Clustering Hypergraph Models 46

4.5 Summary 50

5 Efficient Successor Retrieval Operations 52

5.1 *Get-Unevaluated-Successors* (*GUS*) 53

5.2 Clustering Hypergraph Model for *GUS* Operations 57

5.2.1 Hypergraph Construction 57

5.2.2 Hypergraph Model 60

5.3 Summary 64

6 Experimental Results 66

6.1 Experimental setup 66

6.2 Implementation Details 69

6.3 Evaluation of Junction-Based Storage Scheme 70

6.3.1 Query Generation 70

6.3.2 Properties of Graphs and Hypergraphs 71

6.3.3 Comparison of RB1 and RB2 schemes 72

6.3.4 Comparison of Clustering Graph and Hypergraph Models . 75

6.4 Evaluation of Link-Based Storage Scheme 78

6.4.1 Query Generation 78

6.4.2 Storage Size of Junction- and Link-Based Schemes 79

6.4.3 Properties of Hypergraphs 81

6.4.4 Partitioning Quality 82

6.4.5 Disk Access Simulations 85

6.5 Evaluation of Efficient Successor Retrieval Operations 90

6.5.1 Query Generation 90

6.5.2 Properties of Hypergraphs 92

6.5.3 Partitioning Quality 93

6.5.4 Disk Access Simulations 95

7 Conclusions and Future Work **100**

Bibliography **103**

List of Figures

2.1 A sample road network. 15

3.1 A 3-way vertex partition, which models disk access costs of (a) GaS and (b) GS operations for the clustering graph $\mathcal{G}$ of the sample network given in Fig. 2.1. 24

3.2 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 $GS(t_1)$ operations. 26

3.3 The clustering hypergraph $\mathcal{H}$ for the network given in Fig. 2.1 and a 3-way vertex partition separately shown on net-induced subhypergraphs (a) $(\mathcal{V}, \mathcal{N}^{GaS})$ and (b) $(\mathcal{V}, \mathcal{N}^{GS})$ respectively modeling disk access costs of GaS and GS operations. 29

4.1 Storage of records in a bidirectional sub-network using (a) the junction-based and (b) the link-based storage schemes. 40

4.2 The clustering hypergraph construction: (a) two-pin net n_{123} for the $GaS(\ell_{12}, \ell_{23})$ operations, (b) coalescence of two two-pin nets incurred by $GaS(\ell_{12}, \ell_{21})$ and $GaS(\ell_{21}, \ell_{12})$ into net n_{121}, (c) multi-pin net n_{12} for the $GS(\ell_{12})$ operations. 43

4.3 The clustering hypergraph $\mathcal{H}_{\mathrm{L}}$ for the network given in Fig. 2.1 and a 4-way vertex partition separately shown on net-induced subhypergraphs (a) $(\mathcal{V}_{\mathrm{L}}, \mathcal{N}_{\mathrm{L}}^{GaS})$ and (b) $(\mathcal{V}_{\mathrm{L}}, \mathcal{N}_{\mathrm{L}}^{GS})$ respectively modeling the disk access cost of GaS and GS operations. 45

4.4 (a) A sub-network with $GS(t_3)$, (b) $\mathcal{H}_{\mathrm{T}}$: a four-pin net n_3 for the $GS(t_3)$ operations with $f(t_3) = 10$, (c) $\mathcal{H}_{\mathrm{L}}$: two four-pin nets n_{13} for the $GS(\ell_{13})$ operations with $f(\ell_{13}) = 5$ and n_{23} for the $GS(\ell_{23})$ operations with $f(\ell_{13}) = 5$. 48

4.5 (a) A bidirectional sub-network with $GS(t_1)$, (b) $\mathcal{H}_{\mathrm{T}}$: a five-pin net n_1 for the $GS(t_1)$ operations with $c(n_1) = f(t_1)$, (c) $\mathcal{H}_{\mathrm{L}}$: four identical four-pin nets n_{12}, n_{13}, n_{14}, and n_{15} for $GS(\ell_{12})$, $GS(\ell_{13})$, $GS(\ell_{14})$, and $GS(\ell_{15})$, respectively, (d) $\mathcal{H}_{\mathrm{L}}$: identical nets n_{12}, n_{13}, n_{14}, and n_{15} coalesced into net n'_1 with cost $c(n'_1) = c(n_1)$. 49

5.1 The clustering hypergraph construction: $GUS(t_1, \{t_3, t_5\})$ incurs a net with pins $\{v_1, v_3, v_5\}$. 58

5.2 A sample road network with a query set. 61

5.3 Clustering hypergraphs (a) $\mathcal{H}_{\mathrm{GS}}$, (b) $\mathcal{H}_{\mathrm{GuPS}}$, and (c) $\mathcal{H}_{\mathrm{GuVS}}$ for the sample road network in Fig. 5.2 and 3-way vertex partitions of these hypergraphs. 61

6.1 Comparison of RB1 and RB2 schemes for D0–D7 datasets in terms of the number of allocated pages 73

6.2 Comparison of RB1 and RB2 schemes for D0–D7 datasets in terms of the cutsize 74

6.3 Percent cutsize improvement of the clustering hypergraph model over the clustering graph model 75

6.4 The total disk access costs of aggregate network queries for each dataset in the clustering graph and hypergraph models with increasing page size P in KB and page buffer size B in number of pages. 77

6.5 Partitioning quality of the clustering hypergraph models for the junction- and link-based storage schemes with $C_{\mathrm{T}}=0$. 83

6.6 Disk access comparisons of the two storage schemes in aggregate network query simulations for the Q_{short} query set and $C_{\mathrm{T}}=0$. . . 86

6.7 Disk access comparisons of the two storage schemes in aggregate network query simulations for the Q_{long} query set and $C_{\mathrm{T}}=0$. . . . 87

6.8 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. 94

6.9 Total disk access cost of $(GS, \mathcal{H}_{\mathrm{GS}})$, $(GuPS, \mathcal{H}_{\mathrm{GuPS}})$, and $(GuVS, \mathcal{H}_{\mathrm{GuVS}})$ in query simulations using different page size P in KB and buffer size B in number of pages for Q_{short} query set. . 96

6.10 Total disk access cost of $(GS, \mathcal{H}_{\mathrm{GS}})$, $(GuPS, \mathcal{H}_{\mathrm{GuPS}})$, and $(GuVS, \mathcal{H}_{\mathrm{GuVS}})$ in query simulations using different page size P in KB and buffer size B in number of pages for Q_{long} query set. . 97

List of Tables

6.1 Properties of road network datasets 67

6.2 Average number of junctions accessed in queries 71

6.3 Properties of generated clustering graphs and hypergraphs 72

6.4 Properties of query sets 79

6.5 Storage requirements of junction and link-based storage schemes (in bytes) 80

6.6 Difference between theoretical and actual storage requirements of the link-based storage scheme (in bytes) 81

6.7 Properties of the clustering hypergraphs for the junction- and link-based storage schemes 82

6.8 Averages for percent cutsize improvement of the link-based storage scheme over the junction-based storage scheme 84

6.9 Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme in terms of total number of disk accesses for $C_{\mathrm{T}}=0$ 88

6.10 Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme in terms of total number of disk accesses for $C_{\mathrm{L}}=28$ 89

6.11 Properties of query sets 91

6.12 Properties of generated hypergraphs 92

6.13 Number of pages (in ranges) for all allocation instances 93

6.14 Averages for percent cutsize improvements of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ 95

6.15 Averages for percent performance improvements of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ in terms of total number of disk accesses. 95

6.16 Averages for percent performance improvements of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ in terms of the number of disk accesses incurred only by the successor retrieval operations. 98

List of Symbols and Abbreviations

$\mathcal{T}$: Road junctions
$\mathcal{L}$: Road segments, links
t_i : A junction $t_i \in \mathcal{T}$
ℓ_{ij} : A link $\ell_{ij} \in \mathcal{L}$ connecting junction t_i to neighbor junction t_j
GS : *Get-Successors* operation
GaS : *Get-a-Successor* operation
GUS : *Get-Unevaluated-Successors* operation
$GuPS$: *Get-unProcessed-Successors* operation
$GuVS$: *Get-unVisited-Successors* operation
$\mathcal{V}$: Vertex set
$\mathcal{E}$: Edge set
$\mathcal{G}$: Graph
$\mathcal{N}$: Net set
$\mathcal{H}$: Hypergraph
RB : Recursive Bipartitioning
C_{id} : Storage size of junction coordinates
C_{T} : Storage size of junction attributes
C_{L} : Storage size of link attributes
d_{avg} : Average number of incoming and outgoing links
$d_{\mathrm{in}}(t_i)$: Number of predecessors of t_i
$d_{\mathrm{out}}(t_i)$: Number of successors of t_i
S_{T} : Total storage size of the junction-based storage scheme
S_{L} : Total storage size of the link-based storage scheme
s_{T} : Average record size of the junction-based storage scheme
s_{L} : Average record size of the link-based storage scheme
P : Page size
B : Buffer size

Chapter 1

Introduction

1.1 Motivation

In the last three decades, numerous conceptual models, spatial access methods, and query processing techniques are proposed [60, 71, 69, 64, 76] 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.

A well-known example of spatial networks is road networks, which form an integral part of many GIS applications, such as location-based services, intelligent traveling systems, vehicle telematics, and location-aware advertising. A road network is represented as a finite collection of junctions and the road segments (links) between pairs of junctions. The distance between two objects in the network is determined by the length of the shortest path connecting these two

objects. Several spatial and non-spatial attributes are associated with junctions (e.g., locations, turn restrictions) and links (e.g., length, average speed limit, capacity, type, location related information). Additionally, points-of-interest data is also associated with junctions and links.

In practice, road 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 road networks, the spatial coherency that exists in accessing data leads to a temporal coherency, that is, connected junctions are accessed almost concurrently. Taking this fact into consideration, it seems reasonable to place the data associated with connected junctions in the same disk pages so that the data can be fetched to the memory with fewer disk accesses. In this thesis, we formulate the allocation of data to disk pages as a clustering problem. 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. Based on this observation, this thesis focuses on storage and access schemes for efficient aggregate query processing on road networks. We propose clustering models based on hypergraph-partitioning, which encapsulates the spatial and temporal coherency in query processing via the utilization of query logs in order to minimize the number of disk accesses during query processing.

1.2 Contributions

The contributions of this thesis can be categorized into two concepts: data storage schemes and kernel set of data retrieval operations. In Chapter 2, we provide the background information and the related work on disk-based storage schemes. Chapters 3 and 4 respectively present the junction- and link-based storage schemes for road networks and our clustering hypergraph models. Chapters 5 introduces a new kernel data retrieval operation and a clustering hypergraph model that encapsulate the cost of this kernel operation. In Chapter 6, we present and discuss the simulation results of our models in detail. In the following paragraphs, we briefly overview our particular contributions together with the organization of the thesis.

In Chapter 3 (based on [26]), first, we show that the state-of-the-art clustering graph model for the junction-based storage scheme is not able to correctly capture the disk access cost of successor retrieval operations. Second, 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 successor retrieval 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 [75] in order to support network operations on clustered network data. Note that our model can also benefit from the dynamic clustering strategies presented in [75]. Third, we introduce two adaptive partitioning schemes based on recursive bipartitioning. 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.

In Chapter 4 (based on [27]), first, we introduce the link-based storage scheme. In this storage scheme, each record stores the data associated with a link together with the link's connectivity information. Second, we introduce a clustering hypergraph model for the link-based storage scheme to partition the network data

to disk pages. Third, we present a detailed comparative analysis on the properties of the junction- and link-based storage schemes and show that the link-based storage scheme is more amenable to clustering. Fourth, we introduce storage enhancements for bidirectional networks. We show that the link-based storage scheme is more amenable to our enhancements than the junction-based storage scheme and results in better data allocation for processing aggregate network queries.

In Chapter 5 (based on [25]), first, we introduce *Get-Unevaluated-Successors* (*GUS*) 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 *Get-Successors GS* operations. During processing a query, the successors of a junction can be classified as evaluated and unevaluated according to the properties and state of the algorithm in which the *GUS* operation is invoked on that junction. The *GUS* operation is defined as retrieving the unevaluated successors of a given junction. It is an efficient implementation of the *Get-Successors*(*GS*) operation, where the candidate successors to be retrieved are pruned accordingly. We introduce 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 algorithm, and the latter operation typically arises in the incremental network expansion framework. Second, we propose a clustering hypergraph model that captures the disk access cost of *GUS* operations correctly for the junction-based storage scheme. 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.

In Chapter 6, we present and discuss the simulation results of our models in detail. First, we evaluate our clustering hypergraph model for the junction-based storage scheme and recursive bipartitioning schemes on a wide range of real-life road network datasets with synthetic queries. The results of the conducted experiments show that the proposed clustering hypergraph model is quite effective in reducing the number of disk accesses incurred by the network operations. Second, extensive experimental comparisons are carried out on the effects

of page size, buffer size, path length, record size, and dataset size for the junction- and link-based storage schemes. According to the experimental results, the link-based storage scheme can be a good alternative to the widely used junction-based storage scheme. Third, the proposed *GUS* operation and associated hypergraph-based clustering model for the junction-based storage scheme are evaluated for two different instances of *GUS* operations: *Get-unProcessed-Successors* and *Get-unVisited-Successors*. The results of our experimental simulations 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.

Finally, Chapter 7 concludes the thesis with a summary of the contributions and propose some future research directions.

Chapter 2

Background and Related Work

2.1 Road Networks

A well-known example of spatial networks is the road networks. We represent a road network as a two tuple $(\mathcal{T}, \mathcal{L})$, where $\mathcal{T}$ denotes the spatial locations (junctions) and $\mathcal{L}$ denotes the road segments (links) between pairs of junctions. That is, $\ell_{ij} \in \mathcal{L}$ denotes a link from a junction $t_i \in \mathcal{T}$ to a neighbor junction $t_j \in \mathcal{T}$. 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 [11, 40, 47].

2.2 Query Processing on Road Networks

With the increasing interest in intelligent transportation and modern spatial database management systems, complex and advanced query types need to be supported. 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. Every conventional spatial query type such as nearest neighbors, range search, closest pairs, and spatial join has a counterpart in spatial network databases as discussed in [59, 28]. In network query processing, computation of shortest paths between all pairs of objects constitutes a major problem since the distance between two objects in road networks is determined by the length of the shortest path connecting these objects.

Algorithms and approaches proposed for the computation of shortest paths in networks are based on three main strategies.

- Network distances are computed on-the-fly [59, 84, 23, 28] using the state-of-the-art shortest path algorithms such as Dijkstra's algorithm [31] and A* algorithm [24]. Disk-based shortest path algorithms are also proposed [17, 16] for the case where the size of the network is larger than the available memory.
- Materialization techniques are proposed to precompute the network distances and store partial/full list of shortest path distances between all pairs of objects to support efficient distance computations [48, 49, 53, 66, 16, 65].
- The underlying network is transformed into another representation in which a network distance between two objects can be found in almost constant time [70, 35, 66].

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.

Recent frameworks for query processing on spatial networks [59, 66, 28] tend to support efficient computation of path distances and provide efficient algorithms for the computation of nearest neighbors, range search, closest pairs, and spatial join queries. Papadias et al. [59] introduced a disk-based network representation that integrates connectivity and location information, while spatial entities are

indexed by respective spatial access methods for supporting Euclidean queries and dynamic updates. They proposed Incremental Eucledian Restriction (IER) and Incremental Network Expansion (INE) frameworks. In IER framework, after the selection of candidate region in Eucledian space, network distances are computed and used to determine the actual candidates by utilizing the Eucledian lower-bound property of the network. Network distances are computed on-the-fly using the A* algorithm when the source and destination are known, otherwise Dijkstra's algorithm is employed in queries such as range search. The IER framework assumes Eucledian lower-bound property, which may not hold in practice (e.g., when the search criteria is defined as the expected travel time). In INE, network expansion strategy of the Dijkstra's algorithm is used in query processing. The INE framework assumes monotonically increasing path distances as in the Dijkstra's algorithm. In a recent work, Deng et al. [28] showed that the candidates in IER and INE can be further pruned by utilizing an incremental lower bound property to achieve instance optimal query processing in spatial networks. They introduced the path distance lower bound which is defined as the minimum of the sum of the actual shortest distance traveled from the source node to the current node and the estimated (or "heuristic") distance from the current node to the destination node. Based on this lower bound, Deng et al. [28] presented instance optimal algorithms for nearest neighbor, range search, closest pairs and multi-source skyline queries.

Sankaranarayanan et al. [66] introduced the Spatially Induced Linkage Cognizance (SILC) framework using the path coherence between shortest paths and the spatial positions of objects on the spatial network. In the SILC framework, distances between all pairs of objects in a spatial network is precomputed and stored to efficiently process spatial network queries. Even though they argue on the feasibility of their approach, it may not be a practical approach for all applications when the storage and the complexity of updates are considered. Samet et al. [65] recently proposed a scalable approach for network distance browsing based on the precomputation of the shortest paths between all possible vertices in the network. They make use of an encoding that takes advantage of the fact that the shortest paths from a source node to all of the remaining nodes can be

decomposed into subsets based on the first links on the shortest paths between source and destination nodes. In [65], the amount of storage required to keep track of the subsets is reduced by taking advantage of their spatial coherence using a shortest path quadtree.

Range queries [59, 68, 28], closest pairs [59, 28], spatial join [67], and skyline queries [29, 45] on road networks took the attention of many researchers. Here, we give a detailed survey for the most popular search algorithm on road networks namely nearest neighbor search and its extensions.

2.2.1 Nearest Neighbor Search

Studies on nearest neighbor search algorithms constitute a considerable portion of the previous works due to its applications in many disciplines. A k-nearest neighbor query returns the k points of interest that have the minimum network distance from a query point. First, Jensen et al. [46] introduced a general framework for nearest neighbor queries in client-server architectures. A best-first search algorithm is used to compute the nearest neighbor candidate set in the server side. Actual distances from the query point to objects in the candidate set is computed and the nearest neighbor list is maintained by sorting the distances in the client side. Papadias et al. [59] proposed nearest neighbor search algorithms using the IER and INE frameworks. They showed that INE is more efficient and robust than IER, which suffers by the excessive network distance computations due to false hits. However, IER could perform better in denser, more regular networks (e.g., city blocks), where the Euclidean distance gives a better approximation of the travel cost. The main disadvantage of these approaches is they perform poorly when objects are not densely distributed in the network. In order to overcome this problem, Kolahdouzan and Shahabi [53] proposed a novel approach using first order Voronoi diagram. In their approach, the network is partitioned into Voronoi regions, where each cell is centered by one object and contains the nodes that are closest to that object in network distance. Distances between nodes within the region and the center object as well as the distances between the border nodes of the adjacent cells are precomputed to reduce the computation time

of distances and k-nearest neighbors. However, for increasing values of k, it gets computationally more expensive, because many paths can be traversed between the Voronoi regions. For sparse data sets, the number of border points is large since the number of Voronoi partitions is small and consequently its polygon is complex. This makes the computation in the refinement step more complex since the distances form the query point to all elements in the candidate set need to be computed. On the other hand, if the data set is dense, the number of Voronoi partitions and the size of the candidate set is large. Hence, there will be several possibilities in the refinement step.

Huang et al. [41] proposed the Islands approach that precomputes and stores the nearest nodes of the points of interest with a maximum distance of the radius of the island. This approach has the advantage of precomputation approach together with the network expansion approach. Cho and Chung [19] presented an approach which is very similar to the Islands approach. They introduced the concept of condensing points and precomputed the nearest points of interest of the condensing points. In [19], the authors fixed the number of nearest neighbors to be precomputed for each condensing point. Almeida and Güting [23] proposed an incremental nearest neighbor search algorithm using the Dijkstra's algorithm. They used an R-tree [36] for indexing the links, which are stored as polylines and a B-tree for indexing relative positions of points of interest lying on that link. Recently, Samet et al. [65] proposed a best-first k-nearest neighbor search algorithm in static networks utilizing their scalable approach for network distance browsing based on precomputation of the shortest paths between all possible vertices in the network. As expected, in [65], precomputation-based approaches are shown to perform better than on-the-fly computation when many queries are made on a particular spatial network. However, the methods based on Dijkstra's algorithm may be preferable especially if the desired neighbors are quite close to the query object as the entire spatial network need not be explored.

Ku et al. [55] introduced the travel time network, whose link costs are dynamic and reflect real time traffic functions. Based on this foundation, they proposed a local-based greedy nearest neighbor algorithm that exploits peer-to-peer communication among clients to compute results and a global-based adaptive nearest

neighbor algorithm that relies on a centralized server for retrieving results. In order to continuously monitor the nearest neighbors, Mouratidis et al. [58] proposed two approaches. In their first approach, they maintained the query results by processing only updates that may invalidate the current nearest neighbor sets. In the latter approach, the queries that fall in the path between two consecutive intersections in the network are grouped together, and nearest neighbors are computed by monitoring the nearest neighbor sets of these intersections.

2.2.2 Advanced Nearest Neighbor Search

A continuous nearest neighbor query is defined as finding the nearest points of interest along an entire path and the result is a set of intervals and their nearest neighbors. Kolahdouzan and Shahabi [54] extended the Voronoi-based approach to compute results for continuous nearest neighbor queries. Cho and Chung [19] combined network expansion and precomputed nearest neighbor lists of network nodes in order to derive query results.

Yiu et al. [82] proposed solutions for the reverse nearest neighbor queries in large graphs, where a reverse nearest neighbor query returns data objects that have a query point as their nearest neighbor. Yoo and Shekhar [85] introduced in-route nearest neighbor queries on spatial networks. An in-route nearest neighbor query searches for all points of interest closest to a prespecified route, hence the detour from the route on the way to the destination is smallest via these points of interest.

An aggregate nearest neighbor query is defined as finding the point of interest that minimizes an aggregate distance function (e.g., sum of distances, maximum travel time) with respect to a set of query points. Yiu et al. [84] proposed solutions for aggregate nearest neighbor queries using alternative aggregate functions. Their approaches, which utilize Euclidean distance bounds, spatial access methods, and network distance materialization structures, takes advantage of IER and INE frameworks proposed in [59]. IER approach is shown to be the best when the link costs are proportional to their lengths so that Eucledian distance is a tight

lower bound on actual network distance. However, their IER algorithm requires a large number of network traversals for the distance computation of candidates. Hu et. al [38] proposed a method based on precomputation of distance signatures for each node in the network. Their method is efficient in sparse networks but the precomputation cost will be extensive in dense networks. Alternatively, Ioup et al. [44] proposed an algorithm based on M-tree [20] index structure and the road network embedding method in [70]. The road network embedding method computes approximate network distances and hence it is not suitable for all applications.

2.3 Kernel Set of Database Operations

In road network databases, *Create*, *Find*, *Insert*, *Delete*, *Get-a-Successor*(*GaS*), and *Get-Successors* (*GS*) operations should be performed efficiently as discussed in [75]. 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 Hilbert-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 successor retrieval operations *GaS* and *GS* 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 *GS* 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 recursively repeated until all records of successors

are retrieved.

2.4 Processing Aggregate Network Queries

In road networks, frequently observed aggregate network queries include route evaluation and path computation queries [74], in which an aggregate property is defined as a function of the attributes of junctions and links. Here, we will focus on the problem of determining the aggregate properties on-the-fly by using the network expansion strategy. 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 since an efficient implementation of *GaS* operations reduces the total disk access cost for route evaluation queries. Path computation queries deploy iterative search algorithms such as the breadth-first search, best-first search, A* search, and Dijkstra's algorithm [31] on the network to derive the aggregate properties. The Dijkstra's algorithm processes an unprocessed 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 *GS* operation pairs (i.e., *Find*, *GS*, ..., *Find*, *GS*), where the *Find* operations are selectively performed only if the record is not found in the current page buffer. The quantitative models for the disk access cost of several path computations are summarized in [74] and validated in [73]. These models show that efficient implementation of the *GS* operation leads to reduced disk access cost for many path computations.

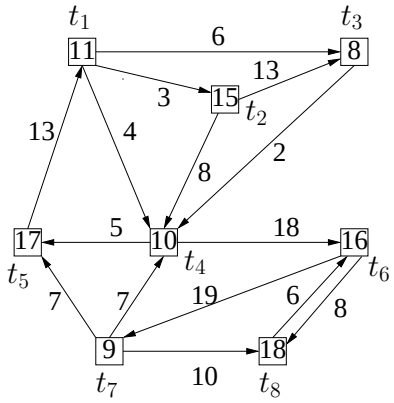


Figure 2.1: A sample road network.

2.5 Data Allocation Problem

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 recursively 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.

Typically, distribution of queries over network elements is not uniform, and individual access frequencies of the network elements are different. Hence, if the previous query logs are available, they can be utilized to estimate the access frequencies of the network elements that will be retrieved by the future queries. Fig. 2.1 illustrates a sample network with 8 junctions and 14 links, where the squares represent the junctions and directed lines represent the links. In the figure, access frequencies for *GS* and *GaS* operations are respectively given in squares and on directed lines.

In this thesis, we mainly focus on the record-to-page allocation problem in road networks. Given a road network $(\mathcal{T}, \mathcal{L})$ and the access frequencies extracted from a query log, the record-to-page allocation problem can be stated as the problem of allocating 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 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 successor retrieval 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.

2.6 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. A weight $w(v_i)$ is associated with each vertex $v_i\in\mathcal{V}$ and a cost $c(e_{ij})$ is assigned with each edge $e_{ij}\in\mathcal{E}$. $\Pi=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K\}$ 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 $\mathcal{E}_{\text{cut}}$, that is,

$$\text{Cutsize}(\Pi)=\sum_{e_{ij}\in\mathcal{E}_{\text{cut}}}c(e_{ij}). \tag{2.1}$$

The partitioning constraint is to maintain an upper bound on the part weights, i.e., $W_k\leq W_{\max}$, 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 $\mathcal{N}$ [7]. Each net $n_j\in\mathcal{N}$ connects a subset of vertices in $\mathcal{V}$, which are called as the pins of n_j and denoted as $\text{Pins}(n_j)$. Hence, graph is a special instance of a hypergraph where each net has exactly two pins. 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 $c(n_j)$.

In a given K-way vertex partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}$, 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 a cut net. The partitioning objective is to minimize a cutsize metric defined over the cut nets. In the literature, a number of cutsize metrics are employed [2]. In connectivity−1 $(\lambda - 1)$ metric, which is widely used in VLSI [22] and in scientific computing [4, 13, 77] communities, each net n_j contributes $c(n_j)(\lambda(n_j) - 1)$ to the cutsize of a partition Π. That is,

$$\text{Cutsize}(\Pi) = \sum_{n_j \in \mathcal{N}} c(n_j)(\lambda(n_j) - 1). \tag{2.2}$$

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

The multi-level framework [10] has been successfully adopted in hypergraph partitioning leading to successful hypergraph partitioning tools hMeTiS [50] and PaToH [14]. In multi-level hypergraph partitioning, the original hypergraph is coarsened into a smaller hypergraph after a series of coarsening levels. At each coarsening level, highly coherent vertices are grouped into supervertices by using various matching heuristics. After the partitioning of the coarsest hypergraph, the generated coarse hypergraphs are uncoarsened back to the original, flat hypergraph. At each uncoarsening level, a refinement heuristic (e.g., Fiduccia-Mattheyses [32] or Kernighan-Lin [52]) is applied to minimize the cutsize while maintaining the partitioning constraint.

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 good solution qualities. This paradigm is especially suitable for partitioning hypergraphs when K is not known in advance. In the RB paradigm, first, a two-way partition of the hypergraph is obtained. Then, each part of the bipartition is further bipartitioned in a recursive manner

until the desired number K of parts is obtained or part weights drop below a given maximum allowed part weight, $W_{\max}$.

2.7 Disk-Based Network Representations

A considerable number of studies have been carried out on spatial databases to design effective storage schemes [33, 42, 59, 61, 62, 63, 75, 81] for efficient query processing. These efforts can be categorized into two groups as proximity- and connectivity-based approaches. In the proximity-based approaches [33, 61, 62, 63], interrelation of data is dependent on spatial proximity. However, query processing in spatial networks mostly involves route evaluation and path computation queries [34, 57, 72], 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 [73]. In the connectivity-based approaches [43, 59, 75, 81, 84], 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 [5, 21, 39, 56] and with limited cycles [1]. 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, only a few approaches which fully utilize the connectivity information are proposed [75, 81, 43]. These works use graph partitioning for clustering the spatial network data. These approaches represent the spatial network as an undirected clustering graph, where partitioning the clustering graph induces a clustering of the spatial network data into disk pages and the partitioning objective relates to storing concurrently accessed data in the same pages.

Shekhar and Liu [75] proposed Connectivity-Clustered Access Method (CCAM) to cluster the spatial network data into disk pages based on network connectivity using graph partitioning. Basic database operations (i.e., *Create*,

Find, *Insert*, and *Delete*) and successor retrieval operations (i.e., *GaS* and *GS*) are evaluated in their clustering graph model. CCAM focuses on *Get-a-Successor* (*GaS*) operations to retrieve a successor of a junction in route evaluation queries and *Get-Successors* (*GS*) operations to retrieve all successors of a junction in path computation queries. CCAM correctly captures the disk access cost of (*GaS*) operations and tries to minimize the disk access cost of successor retrieval operations. In [75], the authors also evaluate dynamic reclustering strategies. Experimental results show that CCAM performs better than the previous proximity- and connectivity-based methods in reducing the number of disk page accesses. The clustering graph model is used in the recent spatial query processing and clustering works [49, 83, 84, 23].

Woo and Yang [81] proposed the Network-Traversal Clustering (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 assumption is not appropriate for spatial networks since, in most cases, the record sizes vary depending on the connectivity of the network.

Both CCAM [75] and NTC [81] methods fail to correctly capture the number of disk accesses incurred by the successor retrieval operations on spatial networks. As mentioned in [75], although the clustering graph model accurately captures the disk access cost of *GaS* operations, it cannot correctly capture the disk access cost of *GS* operations. When *GaS* and *GS* operations are not uniformly distributed over the network and the *GS* operations are more frequent than the *GaS* operations, the performance of the clustering graph model degrades. We will describe in detail and discuss more on the clustering graph model in Section 3.2.

Huang et al. [42] described a general scheme, where links of the network are stored in a separate link table. They proposed a clustering technique, called spatial partition clustering (SPC), for optimization of path computations. 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 [43], Huang et al. proposed alternative

approaches based on SPC and the two-way partitioning clustering [18], which try to utilize both the spatial locality and connectivity information. They used the ratio-cut [18] and Fiduccia-Mattheyses [32] heuristics to reduce the number of cross-page links. Papadias et al. [59] proposed 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. Yiu et al. [84] partitioned the network graph into grids using spatial coordinates of the nodes. The data records are packed into pages in a breath-first manner starting from a randomly selected node in a grid and selecting nodes from that grid using the connectivity information. These approaches do not model the disk access cost of successor retrieval operations in path computation queries. None of the above approaches but CCAM capture spatial and temporal coherency in queries via utilizing query logs.

Chapter 3

Junction-Based Storage Scheme

In this chapter, fist, we describe the widely used junction-based storage scheme. We show that the state-of-the-art clustering graph model for allocation of data of disk pages is not able to correctly capture the disk access costs of successor retrieval operations. 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 due to successor retrieval 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 in Section 6.3. 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.

The organization of this chapter is as follows: In Section 3.2, the clustering graph model and its deficiencies are discussed. Section 3.3 presents the clustering hypergraph model for the junction-based storage scheme. We present our adaptive partitioning schemes in Section 3.4. Finally, we conclude the chapter with a summary in Section 3.5.

3.1 Definition

A frequently used approach for storing a road network in the secondary storage is to use the adjacency list data structure, where a record is allocated for each junction of the network [75, 59, 69, 84, 76]. 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 $Suc(t_i)$ denotes the list of outgoing links of t_i. Each element in the predecessor list stores the coordinates of the source junction t_h of an incoming link ℓ_{hi}. The predecessor lists are used in maintenance operations to update the successor lists. In the successor list, each element stores the coordinates of the destination junction t_j of an outgoing link ℓ_{ij} as well as the attributes of ℓ_{ij}. The record sizes are not fixed because of the variation in the predecessor and successor list sizes. If all links of a junction t_i are bidirectional, a storage saving can be achieved since the predecessor and successor lists of t_i contain exactly the same set of junctions. Hence, it suffices to store only the successor list of t_i.

3.2 Clustering Graph Model

In this section, we briefly describe the clustering graph model proposed by Shekhar and Liu [75] and deficiencies of their model.

3.2.1 Graph Representation

An undirected clustering graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is created to model the network $(\mathcal{T}, \mathcal{L})$. In $\mathcal{G}$, a vertex $v_i \in \mathcal{V}$ exists 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. 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, $e_{ij} \in \mathcal{E}$ if either $\ell_{ij} \in \mathcal{L}$ or $\ell_{ji} \in \mathcal{L}$

or both. The cost $c(e_{ij})$ associated with e_{ij} is

$$c(e_{ij}) = \begin{cases} f(t_i)+f(t_i,t_j), & \text{if } \ell_{ij} \in \mathcal{L}, \ell_{ji} \notin \mathcal{L}; \\ f(t_j)+f(t_j,t_i), & \text{if } \ell_{ji} \in \mathcal{L}, \ell_{ij} \notin \mathcal{L}; \\ f(t_i)+f(t_j)+f(t_i,t_j)+f(t_j,t_i), & \text{if } \ell_{ij}, \ell_{ji} \in \mathcal{L}. \end{cases} \tag{3.1}$$

3.2.2 Graph Model

Shekhar and Liu [75] formulate the record-to-page allocation problem as the problem of partitioning the clustering graph $\mathcal{G}$ with the disk page size being the upper bound on part weights. Shekhar and Liu partition $\mathcal{G}$ into a number of parts $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$, where each part $\mathcal{V}_k \in \Pi$ corresponds to the subset of records to be assigned to disk page $\mathcal{P}_k \in \mathcal{P}$. 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 (2.1). This cutsize relates to the total disk access cost of successor retrieval operations. Note that, in the original problem definition given in Section 2.5, the number K of parts is not known in advance. Thus, they use a partitioning algorithm based on recursive bipartitioning 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. 3.1 shows the clustering graph $\mathcal{G}$ for the sample network given in Fig. 2.1. In the figure, circles denote vertices, and lines denote edges. For the sake of clarity, $\mathcal{G}$ is displayed in two parts, where edge costs represent the access frequencies of *GaS* operations in Fig. 3.1(a) and *GS* operations in Fig. 3.1(b). Fig. 3.1 also shows a 3-way partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\}$ of $\mathcal{G}$, where the shaded regions represent the vertex parts.

3.2.3 Deficiencies 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

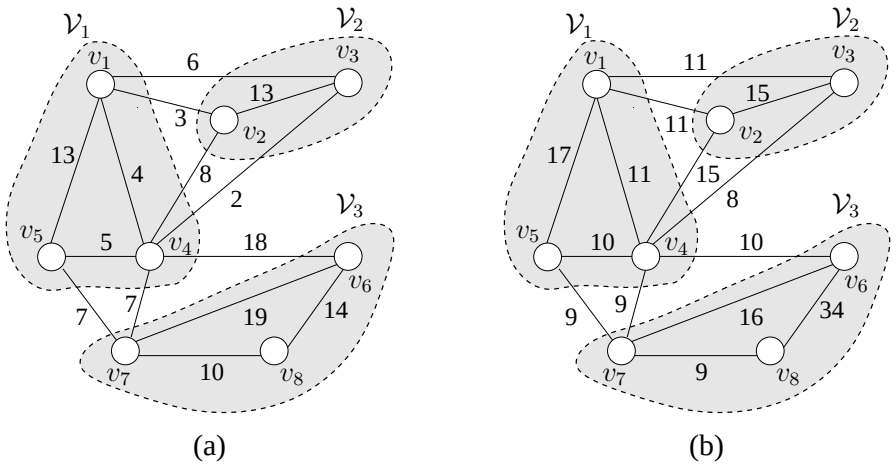


Figure 3.1: A 3-way vertex partition, which models disk access costs of (a) *GaS* and (b) *GS* operations for the clustering graph $\mathcal{G}$ of the sample network given in Fig. 2.1.

the cost of *GS* 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 deficiency 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. 3.1, we illustrate the deficiency of the clustering graph model in estimating the total disk access cost of *GS* operations using the sample partition $\Pi = \{\mathcal{V}_1 = \{v_1, v_4, v_5\}, \mathcal{V}_2 = \{v_2, v_3\}, \mathcal{V}_3 = \{v_6, v_7, v_8\}\}$. According to partition Π, the total costs of *GaS* and *GS* operations, due to the cut edges in $\mathcal{E}_{\text{cut}} = \{e_{12}, e_{13}, e_{24}, e_{34}, e_{46}, e_{47}, e_{57}\}$, 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 GS 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 $GS(t_1)$ and $GS(t_7)$ operations by the clustering graph model. For example, the disk access cost of $GS(t_1)$ operations, where the set of successors of t_1 is $\text{Suc}(t_1)=\{t_2,t_3,t_4\}$, 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 $GS(t_1)$ operation.

3.3 Clustering Hypergraph Model

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

3.3.1 Hypergraph Construction

A clustering hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ is created to model the network $(\mathcal{T},\mathcal{L})$. In $\mathcal{H}$, a vertex $v_i\in\mathcal{V}$ exists 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. The net set $\mathcal{N}$ of $\mathcal{H}$ is the union of two disjoint sets of nets, $\mathcal{N}^{GaS}$ and $\mathcal{N}^{GS}$, which respectively encapsulate the disk access costs of GaS and GS operations, i.e., $\mathcal{N}=\mathcal{N}^{GaS}\cup\mathcal{N}^{GS}$.

We employ two-pin nets in $\mathcal{H}$ to represent the disk access cost of GaS operations. In $\mathcal{N}^{GaS}$, 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, $n_{ij}\in\mathcal{N}^{GaS}$ if either $\ell_{ij}\in\mathcal{L}$ or $\ell_{ji}\in\mathcal{L}$ 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

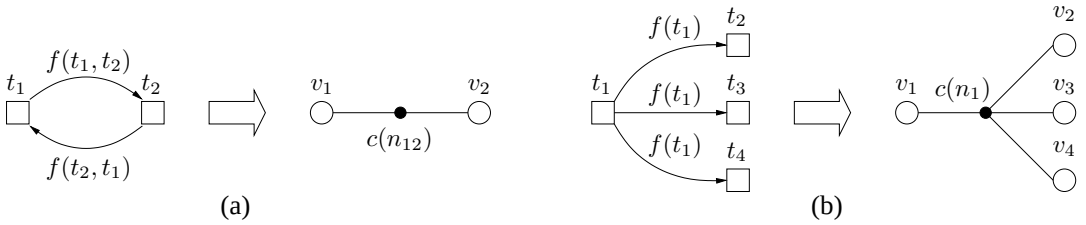


Figure 3.2: The clustering hypergraph: (a) two-pin net n_{12} for the $GaS(t_1, t_2)$ and $GaS(t_2, t_1)$ operations (b) multi-pin net n_1 for the $GS(t_1)$ operations.

$$c(n_{ij}) = \begin{cases} f(t_i, t_j), & \text{if } \ell_{ij} \in \mathcal{L}, \ell_{ji} \notin \mathcal{L}; \\ f(t_j, t_i), & \text{if } \ell_{ji} \in \mathcal{L}, \ell_{ij} \notin \mathcal{L}; \\ f(t_i, t_j) + f(t_j, t_i), & \text{if } \ell_{ij}, \ell_{ji} \in \mathcal{L}. \end{cases} \tag{3.2}$$

Note that these two-pin nets correspond to the edges employed in the clustering graph described in Section 3.2.1 with the difference that their costs do not include the costs of GS operations (i.e., the access frequency $f(t_i)$ of junction t_i). Fig. 3.2(a) displays the two-pin net construction for a pair of neighbor junctions t_1 and t_2.

We employ multi-pin nets in $\mathcal{H}$ to represent the disk access cost of GS operations. In $\mathcal{N}^{GS}$, 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,

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

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

$$c(n_i) = f(t_i) \tag{3.3}$$

to capture the cost of $GS(t_i)$ operations. Fig. 3.2(b) displays the multi-pin net construction for junction t_1 with $\text{Suc}(t_1) = \{t_2, t_3, t_4\}$.

The size of the clustering hypergraph $\mathcal{H}$ in terms of the number of pins depends on the topological properties of the network. The number $|\mathcal{N}^{GaS}|$ of two-pin nets

varies between $\lceil |\mathcal{L}|/2 \rceil$ and $|\mathcal{L}|$. The number $|\mathcal{N}^{GS}|$ of multi-pin nets equals $|\mathcal{T}|-\alpha$, 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 $|\mathcal{L}|+|\mathcal{T}|-\alpha$. Hence, the total number of pins in $\mathcal{H}$ is between $2\lceil |\mathcal{L}|/2 \rceil + |\mathcal{L}| + |\mathcal{T}| - \alpha$ and $3|\mathcal{L}| + |\mathcal{T}| - \alpha$.

3.3.2 Hypergraph Model

After modeling the network $(\mathcal{T}, \mathcal{L})$ as a clustering hypergraph $\mathcal{H}$, we formulate the record-to-page allocation problem as the problem of partitioning $\mathcal{H}$ 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 $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$. Here, partition Π induces a record-to-page allocation, where each 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, $v_i \in V_k$ means that record r_i is allocated to page $\mathcal{P}_k$.

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

$$\begin{aligned} \text{Cutsize}(\Pi) &= \sum_{n_i \in \mathcal{N}^{GaS}} c(n_i)(\lambda(n_i) - 1) + \sum_{n_i \in \mathcal{N}^{GS}} c(n_i)(\lambda(n_i) - 1) \\ &= \sum_{n_i \in \mathcal{N}} c(n_i)(\lambda(n_i) - 1). \end{aligned} \tag{3.4}$$

In fact, under the assumption that a single-page buffer is available, the $\lambda - 1$ cost incurred to the partition by the 2-pin cut nets in $\mathcal{N}^{GaS}$ exactly corresponds to the disk access cost incurred by the *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 *GS* operations in the path computation queries. Our model correctly captures the aggregate disk access costs of *GaS* and *GS* operations. Consequently, in our model, minimizing Cutsize(Π) given in (3.4) 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 2-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 $\mathcal{P}_k$. Since both r_i and r_j can be found in the memory when $\mathcal{P}_k$ 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 $\mathcal{P}_k$ and $\mathcal{P}_m$. Without loss of generality, assume that $r_i \in \mathcal{P}_k$ and $r_j \in \mathcal{P}_m$. In this case, $GaS(t_i, t_j)$ operations incur $f(t_i, t_j)$ disk accesses in order to replace the current page $\mathcal{P}_k$ in the buffer with $\mathcal{P}_m$ 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 $\mathcal{P}_m$ in the buffer with $\mathcal{P}_k$ 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 $n_i \in \mathcal{N}^{GS}$. 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 $\mathcal{P}_k$. Consequently, $GS(t_i)$ operations do not incur any disk access since page $\mathcal{P}_k$ 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 $\mathcal{P}_k$, where $\mathcal{V}_k$ must be in $\Lambda(n_i)$. In this case, each $GS(t_i)$ 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. 3.3, we illustrate the correctness of the clustering hypergraph $\mathcal{H}$ for the network given in Fig. 2.1 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, $\mathcal{H}$ is given in two parts which separately show the net sets $\mathcal{N}^{GaS}$ and $\mathcal{N}^{GS}$ with the associated costs of GaS and GS 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 GS 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

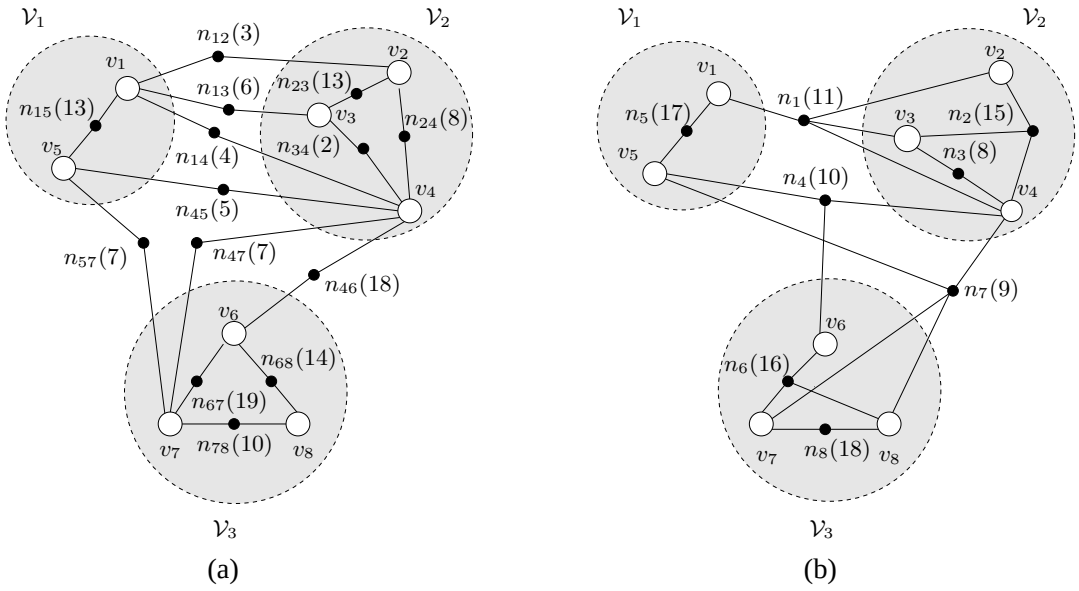


Figure 3.3: The clustering hypergraph $\mathcal{H}$ for the network given in Fig. 2.1 and a 3-way vertex partition separately shown on net-induced subhypergraphs (a) $(\mathcal{V},\mathcal{N}^{GaS})$ and (b) $(\mathcal{V},\mathcal{N}^{GS})$ respectively modeling disk access costs of *GaS* and *GS* operations.

the connectivities of these nets are 2, 3, and 3, respectively. Note that internal nets do not incur any cost for neither *GaS* nor *GS* operations since they have a connectivity of 1. In Fig. 3.3(b), consider cut net n_1 with $\text{Pins}(n_1)=\{v_1,v_2,v_3,v_4\}$ and $\Lambda(n_1)=\{\mathcal{V}_1,\mathcal{V}_2\}$. Since v_1 is in vertex part $\mathcal{V}_1$, page $\mathcal{P}_1$ must be the single page in the buffer when $GS(t_1)$ operations are invoked. Since v_2, v_3, and v_4 are all in $\mathcal{V}_2$, each of the 11 $GS(t_1)$ operations will incur only one disk access for page $\mathcal{P}_2$ 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. 2.1. In Fig. 3.1 and Fig. 3.3, 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 $\mathcal{V}_1$ in the clustering graph model, whereas it is in $\mathcal{V}_2$ 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 $GS(t_1)$ operation prevents vertex v_4 from moving to $\mathcal{V}_2$ although introducing this edge to the cut actually incurs no additional cost.

3.4 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 [37]) even without the main objective of minimizing the number of disk accesses.

The Recursive Bipartitioning (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 [13] is adopted to capture the $\lambda-1$ cutsize metric given in (3.4). In both graph and hypergraph partitioning, balancing the part weights of the bipartition is enforced as the bipartitioning constraint.

3.4.1 Determining Number of Pages

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 $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$, a part $\mathcal{V}_k$ is said to be lightly loaded if $(W_k/W_{\max}) \leq 0.5$ 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,

$$W_1, W_2 \leq \frac{W_1+W_2}{2}(1+\epsilon), \tag{3.5}$$

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 (3.5) 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

$$W_1, W_2 \leq P \left\lceil \frac{W_1+W_2}{2P} \right\rceil (1+\epsilon) \tag{3.6}$$

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.

3.4.2 Packing Lightly Loaded Parts

Elimination of lightly loaded parts can be formulated as an instance of the bin-packing problem [37], 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.

3.5 Summary

We investigated the record-to-page allocation problem for the junction-based storage scheme 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* 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 deficiency, we proposed a clustering hypergraph model. Our model correctly captures the costs of disk accesses for both *Get-a-Successor* operations incurred by route evaluation queries and *Get-Successors* operations incurred by 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.

In order to evaluate our clustering hypergraph model for the junction-based storage scheme, we have conducted extensive simulation tests. In Chapter 6, we present and discuss the simulation results of the proposed model in detail. Experimental results obtained on a wide range of road networks verify the validity of our hypergraph model.

Chapter 4

Link-Based Storage Scheme

In this chapter, we introduce the link-based storage scheme and compare it with the previously proposed junction-based storage scheme. We extend our clustering hypergraph model in Chapter 3 from junction-based storage to link-based storage. We propose techniques for additional storage savings in bidirectional networks that make the link-based storage scheme even more preferable in terms of the storage efficiency. We evaluate the performance of our link-based storage scheme against the junction-based storage scheme both theoretically and empirically. The results of the experiments conducted on a wide range of road network datasets and presented in Section 6.4 show that the link-based storage scheme is preferable in terms of both storage and query processing efficiency.

The organization of this chapter is as follows: In Sections 4.1, the link-based storage scheme is introduced. Section 4.2 gives a detailed comparison between the junction-based storage scheme and the link-based storage scheme. Auxiliary index structures to support query processing on these storage schemes are discussed in Section 4.3. Section 4.4 presents our clustering hypergraph model for the link-based storage scheme. Finally, we summarize the chapter in Section 4.5.

4.1 Definition

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

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

4.2 Comparison of Storage Schemes

In practice, the storage size of the link attributes is greater than that of the junction attributes, and the number of links is greater than the number of junctions. Depending on these network-specific parameters, one of the two storage schemes may be favorable in terms of the total storage size and/or the average record size. The role of average record size in the disk access cost of network queries can be explained as follows. For a given query distribution, the sum of the frequencies of the *GS* operations to be invoked from the outgoing links of junction t_j in the

link-based storage scheme is equal to the frequency of the GS operations to be invoked from t_j in the junction-based storage scheme. Hence, in processing a query, the number of records to be retrieved in both storage schemes is the same. Since smaller average record size enables clustering more records to a page, the query overhead is expected to decrease with decreasing average record size. Below, we provide a detailed comparative analysis of the storage schemes in terms of both the total storage size and average record size.

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

$$\begin{aligned} S_{\mathrm{T}} &= \sum_{t\in\mathcal{T}}(C_{\mathrm{id}} + C_{\mathrm{T}} + |Pre(t)|C_{\mathrm{id}} + |Suc(t)|(C_{\mathrm{id}} + C_{\mathrm{L}})) \\ & c(t)|(C_{\mathrm{id}} + C_{\mathrm{L}})) \\ &= |\mathcal{T}|(C_{\mathrm{id}} + C_{\mathrm{T}}) + |\mathcal{L}|(2C_{\mathrm{id}} + C_{\mathrm{L}}) \end{aligned} \tag{4.1}$$

and

$$\begin{aligned} S_{\mathrm{L}} &= \sum_{\ell\in\mathcal{L}}(2C_{\mathrm{id}} + C_{\mathrm{L}} + C_{\mathrm{T}} + |Pre(\ell)|C_{\mathrm{id}} + |Suc(\ell)|C_{\mathrm{id}}) \\ &= |\mathcal{L}|(2C_{\mathrm{id}} + C_{\mathrm{L}} + C_{\mathrm{T}}) + C_{\mathrm{id}}\sum_{\ell\in\mathcal{L}}(|Pre(\ell)| + |Suc(\ell)|), \end{aligned} \tag{4.2}$$

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

$$\begin{aligned} S_{\mathrm{L}} - S_{\mathrm{T}} &= C_{\mathrm{id}}\sum_{\ell\in\mathcal{L}}(|Pre(\ell)| + |Suc(\ell)|) + |\mathcal{L}|C_{\mathrm{T}} - |\mathcal{T}|(C_{\mathrm{id}} + C_{\mathrm{T}}) \\ &= C_{\mathrm{T}}(|\mathcal{L}| - |\mathcal{T}|) + C_{\mathrm{id}}\left(\sum_{\ell\in\mathcal{L}}(|Pre(\ell)| + |Suc(\ell)|) - |\mathcal{T}|\right). \end{aligned} \tag{4.3}$$

In a typical road network, the number of links is greater than the number of junctions (i.e., $|\mathcal{L}| > |\mathcal{T}|$), and each link has at least one predecessor or successor (i.e., $|Pre(\ell)| + |Suc(\ell)| \geq 1$ for each ℓ). Hence, both terms in (4.3) are always

positive. As a result, the link-based storage scheme requires more disk space than the junction-based storage scheme.

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

$$S_{\mathrm{L}} = |\mathcal{L}|(2C_{\mathrm{id}} + C_{\mathrm{L}} + C_{\mathrm{T}}) + 2C_{\mathrm{id}}|\mathcal{L}|d_{\mathrm{avg}} \tag{4.4}$$

and

$$S_{\mathrm{L}} - S_{\mathrm{T}} = C_{\mathrm{T}}(|\mathcal{L}| - |\mathcal{T}|) + C_{\mathrm{id}}(2|\mathcal{L}|d_{\mathrm{avg}} - |\mathcal{T}|). \tag{4.5}$$

Hence, the average record sizes are

$$s_{\mathrm{T}} = \frac{S_{\mathrm{T}}}{|\mathcal{T}|} = C_{\mathrm{id}} + C_{\mathrm{T}} + d_{\mathrm{avg}}(2C_{\mathrm{id}} + C_{\mathrm{L}}) \tag{4.6}$$

and

$$s_{\mathrm{L}} = \frac{S_{\mathrm{L}}}{|\mathcal{L}|} = 2C_{\mathrm{id}} + C_{\mathrm{L}} + C_{\mathrm{T}} + 2C_{\mathrm{id}}d_{\mathrm{avg}}. \tag{4.7}$$

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

$$s_{\mathrm{T}} - s_{\mathrm{L}} = C_{\mathrm{L}}(d_{\mathrm{avg}} - 1) - C_{\mathrm{id}}. \tag{4.8}$$

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

In bidirectional networks, the storage savings described in Sections 3.1 and 4.1 are expected to increase the efficiency of both storage schemes. The link-based

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

$$S_{\mathrm{T}}^{\mathrm{b}} = |\mathcal{T}|(C_{\mathrm{id}} + C_{\mathrm{T}}) + |\mathcal{L}|(C_{\mathrm{id}} + C_{\mathrm{L}}) \tag{4.9}$$

and

$$S_{\mathrm{L}}^{\mathrm{b}} = \frac{|\mathcal{L}|}{2}(2C_{\mathrm{id}} + C_{\mathrm{L}} + 2C_{\mathrm{T}}) + 2C_{\mathrm{id}}|\mathcal{L}|(d_{\mathrm{avg}} - 1). \tag{4.10}$$

Note that (4.10) is derived by using the simplifying assumption mentioned earlier. The difference between the total storage sizes of the two schemes becomes

$$S_{\mathrm{L}}^{\mathrm{b}} - S_{\mathrm{T}}^{\mathrm{b}} = C_{\mathrm{T}}(|\mathcal{L}| - |\mathcal{T}|) + C_{\mathrm{id}}(2|\mathcal{L}|(d_{\mathrm{avg}} - 1) - |\mathcal{T}|) - C_{\mathrm{L}}\frac{|\mathcal{L}|}{2}. \tag{4.11}$$

The comparison of (4.5) and (4.11) shows that the total storage size difference between the two schemes decreases in favor of the link-based scheme by $|\mathcal{L}|(2C_{\mathrm{id}} + C_{\mathrm{L}}/2)$. As seen in (4.11), the link-based scheme may even require less total disk space than the junction-based scheme for large C_{L} values.

In bidirectional networks, the average record sizes become

$$s_{\mathrm{T}}^{\mathrm{b}} = \frac{S_{\mathrm{T}}^{\mathrm{b}}}{|\mathcal{T}|} = C_{\mathrm{id}} + C_{\mathrm{T}} + d_{\mathrm{avg}}(C_{\mathrm{id}} + C_{\mathrm{L}}) \tag{4.12}$$

and

$$s_{\mathrm{L}}^{\mathrm{b}} = \frac{S_{\mathrm{L}}^{\mathrm{b}}}{|\mathcal{L}|/2} = C_{\mathrm{L}} + 2C_{\mathrm{T}} + 2C_{\mathrm{id}}(2d_{\mathrm{avg}} - 1). \tag{4.13}$$

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

$$s_{\mathrm{T}}^{\mathrm{b}} - s_{\mathrm{L}}^{\mathrm{b}} = C_{\mathrm{L}}(d_{\mathrm{avg}} - 1) - 3C_{\mathrm{id}}(d_{\mathrm{avg}} - 1) - C_{\mathrm{T}}. \tag{4.14}$$

The comparison of (4.8) and (4.14) shows that the difference between the average record sizes decreases in bidirectional networks in general. As seen in (4.14), the

average record size of the link-based scheme remains to be less than that of the junction-based scheme for typical networks, where $d_{\text{avg}} > 1$, $C_{\text{L}} > 3C_{\text{id}}$, and C_{T} is quite small.

Even though the average record size difference between the two schemes decreases in bidirectional networks, the link-based storage scheme is still more amenable to record clustering compared to the junction-based scheme. We will explain this advantage of the link-based storage scheme over the junction-based storage scheme for a junction t_j with d links all of which are bidirectional. In the junction-based storage scheme, junction t_j will have d successors. We should cluster record r_j storing t_j together with all the records storing the d successor junctions to the same page to avoid the page access cost for the $GS(t_j)$ operation. That is, these $d+1$ records need to be clustered in the same page. On the other hand, in the link-based storage scheme scheme, each link incident to junction t_j has $d-1$ successors excluding itself. Since r_{ij} stores both ℓ_{ij} and ℓ_{ji}, we should cluster record r_{ij} together with $d-1$ records storing the links incident to t_j other than ℓ_{ji} in the same page to avoid the page access cost for the $GS(\ell_{ij})$ operation. This holds for all records storing the links incident to junction t_j. Hence, it is sufficient to cluster these d records in the same page to avoid the page access cost for the *GS* operations invoked from the links incident to junction t_j. Therefore, in the link-based scheme, each *GS* operation invoked from a junction connected by only bidirectional links can be accomplished by accessing one less record than the junction-based scheme.

Figs. 4.1(a) and (b) respectively show the junction- and link-based storage schemes for a sub-network consisting of a junction t_1 connected by 4 bidirectional links. The data records are shown in the right sides of Fig. 4.1, where the successors are separated by bold lines and additional successors are appended as dotted parts to represent the neighbor junctions/links not shown in the figure. In the junction-based storage scheme, $d = 5$ records (i.e., r_1, r_2, r_3, r_4, and r_5), whereas in the link-based storage scheme $d-1 = 4$ records (i.e., r_{12}, r_{13}, r_{14}, and r_{15}) need to be clustered in a page to avoid the page access cost for the same number of *GS* operations. This explains why the link-based storage scheme will be more amenable to clustering than the junction-based storage scheme even when the

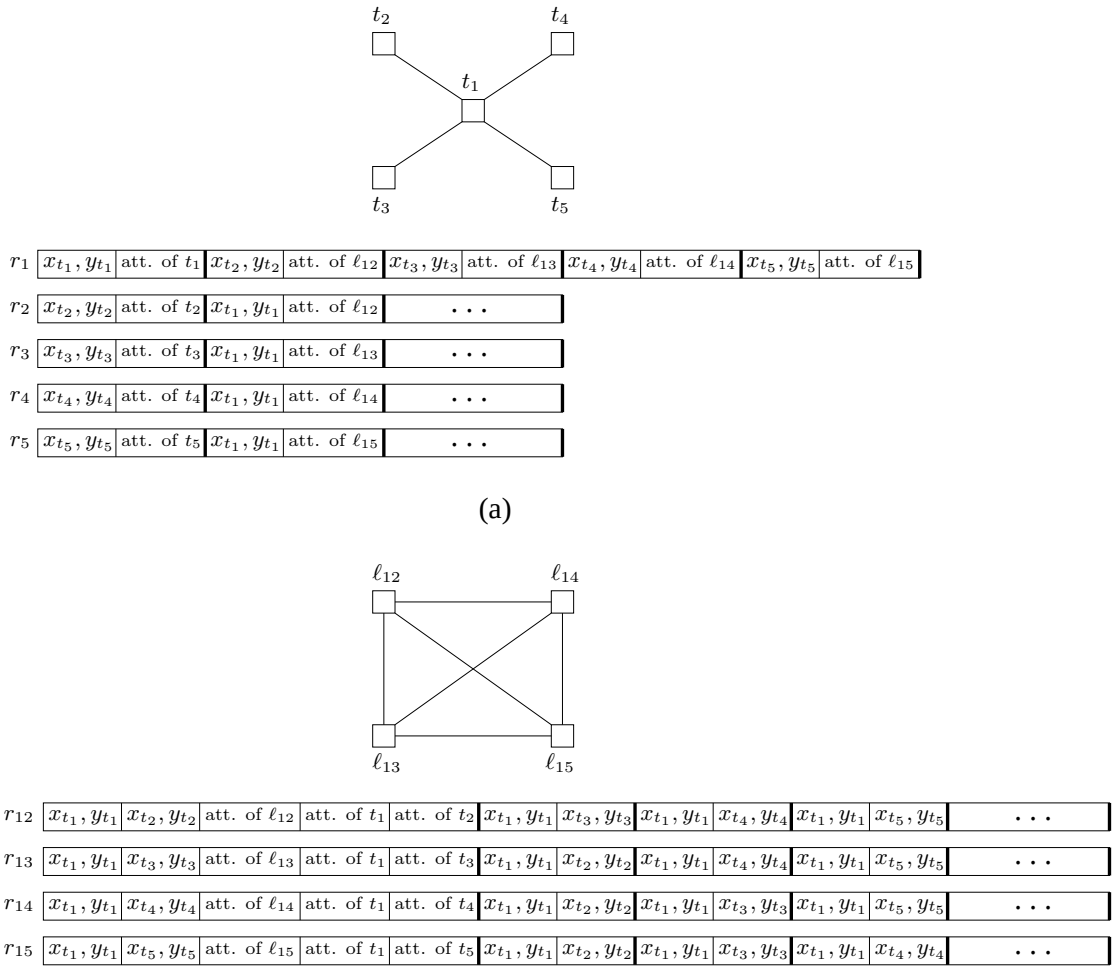


Figure 4.1: Storage of records in a bidirectional sub-network using (a) the junction-based and (b) the link-based storage schemes.

average record sizes are equal in the two storage schemes.

In addition to the above-mentioned advantages in storage size and clustering, the link-based storage scheme, as in the dual network concept, which was originally proposed in [11] and later used in [79] and [80], expresses the relations between consecutive links along paths and is more suitable to capture the restrictions in networks such as turn restrictions.

4.3 Auxiliary Index Structures

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

In general, the route evaluation or path computation queries are submitted to the GIS systems as point queries, which contain the (x, y) coordinates of a source and a destination point. It is more likely that the query points lie on the links rather than junctions. Here, we refer to the link that a source point lies on as the source link. In the link-based storage scheme, route evaluation and path computation start from the source link, whereas, in the junction-based storage scheme, they start from the destination junction of the source link. In both cases, the source link must be identified. In our architecture, an R-tree index on links is used as an additional index in both storage schemes and the sole purpose of this index is to locate the source link. The R-tree has two types of nodes: non-leaf nodes and leaf nodes [36]. Non-leaf nodes contain index record entries of the form <MBR, ptr> where MBR is the minimum bounding rectangle of all rectangles stored in the entries of the lower level child node pointed to by ptr. The only minor difference between the R-tree implementation in the two storage schemes is the data stored in the leaf nodes. Each leaf node stores an <MBR, ptr> pair for a link, where MBR corresponds to the minimum bounding rectangle of the link and ptr is the disk page address of the respective record. This record stores data associated with the respective link in the link-based storage scheme, whereas it stores data associated with the endpoint junction of the respective link in the

junction-based storage scheme. As the leaf nodes determine the overall storage complexity of the index, both storage schemes require an additional storage of size $S_{\text{Rtree}} = |\mathcal{L}|\, C_{\text{Rnode}}$ for indexing the links of the network. Here, C_{Rnode} denotes the size of each leaf node.

4.4 Clustering Hypergraph Model

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

4.4.1 Hypergraph Construction

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

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

$$c(n_{hij}) = \begin{cases} f(\ell_{hi}, \ell_{ij}), & \text{if } \ell_{hi}, \ell_{ij} \in \mathcal{L} \wedge h \neq j; \\ f(\ell_{hi}, \ell_{ij}) + f(\ell_{ij}, \ell_{hi}), & \text{if } \ell_{hi}, \ell_{ij} \in \mathcal{L} \wedge h = j. \end{cases} \tag{4.15}$$

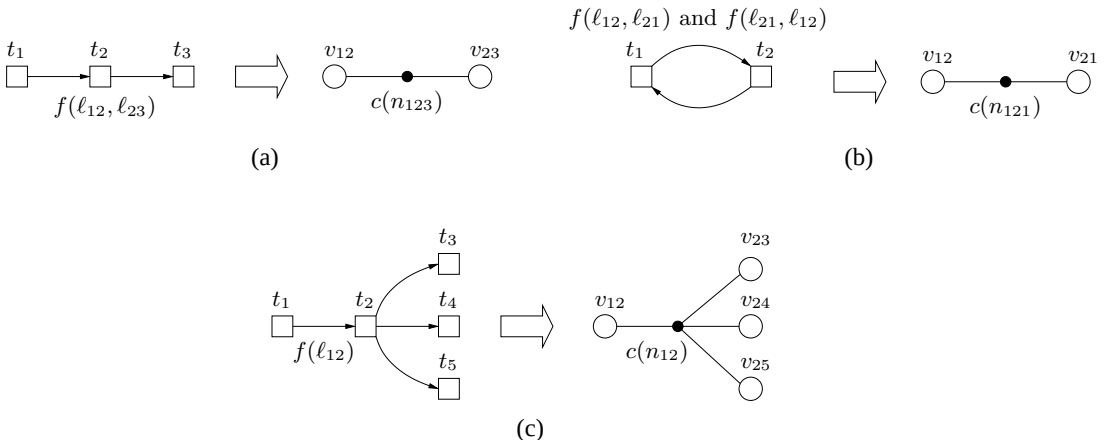


Figure 4.2: The clustering hypergraph construction: (a) two-pin net n_{123} for the $GaS(\ell_{12}, \ell_{23})$ operations, (b) coalescence of two two-pin nets incurred by $GaS(\ell_{12}, \ell_{21})$ and $GaS(\ell_{21}, \ell_{12})$ into net n_{121}, (c) multi-pin net n_{12} for the $GS(\ell_{12})$ operations.

Here, $f(\ell_{hi}, \ell_{ij})$ denotes the total access frequency of path $\langle \ell_{hi}, \ell_{ij} \rangle$ in $GaS(\ell_{hi}, \ell_{ij})$ operations. Fig. 4.2(a) shows the two-pin net construction for a pair of neighbor links ℓ_{12} and ℓ_{23}, and Fig. 4.2(b) shows the two-pin net construction for the cyclic paths $\langle \ell_{12}, \ell_{21} \rangle$ and $\langle \ell_{21}, \ell_{12} \rangle$.

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

$$\mathrm{Pins}(n_{hi}) = \{v_{hi}\} \cup \{v_{ij} : t_j \in \mathrm{Suc}(t_i)\}. \tag{4.16}$$

Each net n_{hi} is associated with a cost

$$c(n_{hi}) = f(\ell_{hi}) \tag{4.17}$$

for capturing the cost of $GS(\ell_{hi})$ operations. Here, $f(\ell_{hi})$ denotes the total access frequency of link ℓ_{hi} in $GS(\ell_{hi})$ operations. Fig. 4.2(c) displays the multi-pin net construction for link ℓ_{12}, which has the successor list $\{\ell_{23}, \ell_{24}, \ell_{25}\}$.

4.4.2 Hypergraph Model

After $\mathcal{H}_{\mathrm{L}} = (\mathcal{V}_{\mathrm{L}}, \mathcal{N}_{\mathrm{L}})$ is constructed, it is partitioned into a number of parts $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots\}$ using the recursive bipartitioning paradigm mentioned in Section 3.4. Here, each part $\mathcal{V}_k \in \Pi$ corresponds to the subset of records to be assigned to disk page $\mathcal{P}_k \in \mathcal{P}$. The partitioning constraint is to enforce the page size as the upper bound on the weight of the vertex parts so that the disk page size is not exceeded in record allocation. The partitioning objective is to minimize the cutsize according to the connectivity-1 metric as defined in Section 2.6. Under the single-page buffer assumption, the connectivity-1 cost incurred to the cutsize by the two-pin cut nets in $\mathcal{N}_{\mathrm{L}}^{GaS}$ and multi-pin cut nets in $\mathcal{N}_{\mathrm{L}}^{GS}$ exactly corresponds to the disk access cost incurred by the *GaS* operations in the route evaluation queries and *GS* operations in the path computation queries, respectively. Thus, in our model, minimizing Cutsize(Π) given in (4.18) exactly minimizes the total number of disk accesses. In the following two paragraphs, we show the correctness of our model for the *GaS* and *GS* operations.

$$\begin{aligned} \text{Cutsize}(\Pi) &= \sum_{n_i \in \mathcal{N}_{\mathrm{L}}^{GaS}} c(n_i)(\lambda(n_i) - 1) + \sum_{n_i \in \mathcal{N}_{\mathrm{L}}^{GS}} c(n_i)(\lambda(n_i) - 1) \\ &= \sum_{n_i \in \mathcal{N}_{\mathrm{L}}} c(n_i)(\lambda(n_i) - 1). \end{aligned} \tag{4.18}$$

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

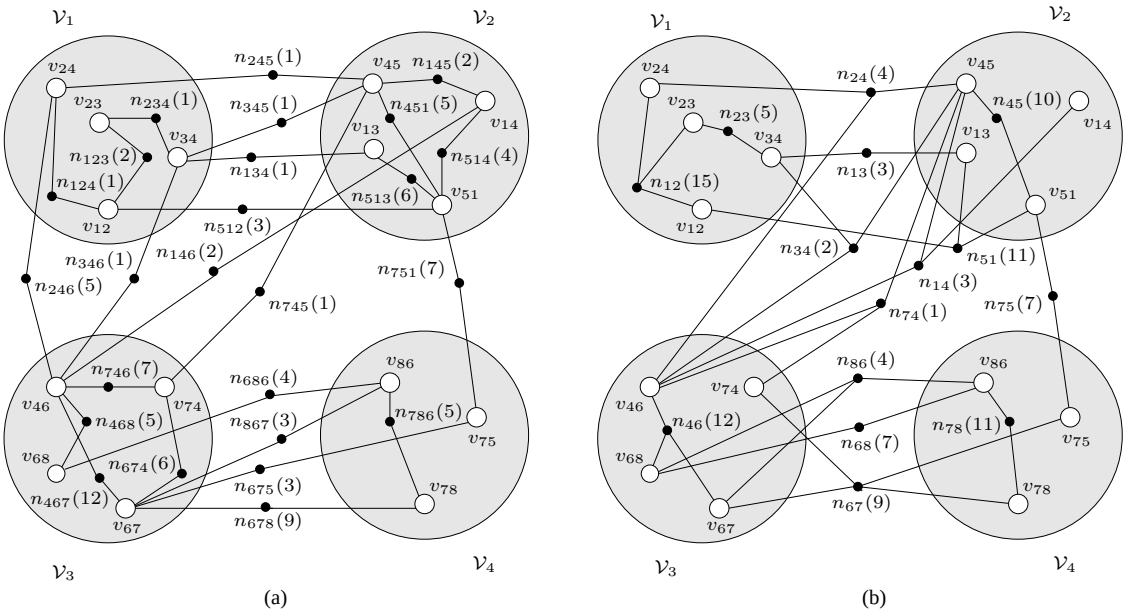


Figure 4.3: The clustering hypergraph $\mathcal{H}_{\mathrm{L}}$ for the network given in Fig. 2.1 and a 4-way vertex partition separately shown on net-induced subhypergraphs (a) $(\mathcal{V}_{\mathrm{L}}, \mathcal{N}_{\mathrm{L}}^{GaS})$ and (b) $(\mathcal{V}_{\mathrm{L}}, \mathcal{N}_{\mathrm{L}}^{GS})$ respectively modeling the disk access cost of *GaS* and *GS* operations.

a cost of $c(n_{hij})$ to the cutsize since $\lambda(n_{hij})-1=1$.

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

Fig. 4.3 shows the clustering hypergraph $\mathcal{H}_{\mathrm{L}}$ for the network given in Fig. 2.1 in two parts, which separately show the net sets $\mathcal{N}_{\mathrm{L}}^{GaS}$ and $\mathcal{N}_{\mathrm{L}}^{GS}$ with the associated costs of *GaS* and *GS* operations shown in parentheses. In Fig. 4.3(a), consider

two-pin cut net n_{246} with $\text{Pins}(n_{246})=\{v_{24}, v_{46}\}$ and $\Lambda(n_{246})=\{\mathcal{V}_1, \mathcal{V}_3\}$. Since v_{24} is in vertex part $\mathcal{V}_1$, page $\mathcal{P}_1$ must be the single page in the buffer when $GS(\ell_{24})$ operations are invoked. Since v_{46} is in part $\mathcal{V}_2$, $\lambda(n_{246})-1=2-1=1$ disk access is required to retrieve record r_{46} into the buffer. Similarly, in Fig. 4.3(b), consider multi-pin cut net n_{24} with $\text{Pins}(n_{24}) = \{v_{24}, v_{45}, v_{46}\}$ and $\Lambda(n_{24}) = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\}$. Since v_{24} is in vertex part $\mathcal{V}_1$, page $\mathcal{P}_1$ must be the single page in the buffer when $GS(\ell_{24})$ operations are invoked. Since v_{45} and v_{46} are respectively in parts $\mathcal{V}_2$ and $\mathcal{V}_3$, each of the four $GS(\ell_{24})$ operations will incur $\lambda(n_{24})-1=3-1=2$ disk accesses for pages $\mathcal{P}_2$ and $\mathcal{P}_3$ to bring them into the buffer for processing records r_{45} and r_{46}. Note that internal nets do not incur any cost for neither *GaS* nor *GS* operations since they have a connectivity of 1. The total cost of *GaS* operations, due to the cut nets $\{n_{134}, n_{146}, n_{245}, n_{246}, n_{345}, n_{346}, n_{512}, n_{675}, n_{678}, n_{686}, n_{745}, n_{751}, n_{867}\}$, is $(1+2+1+5+1+1+3+3+9+4+1+7+3)\times(2-1)=41$ and the total cost of *GS* operations, due to the cut nets $\{n_{13}, n_{14}, n_{24}, n_{34}, n_{51}, n_{67}, n_{68}, n_{74}, n_{75}, n_{86}\}$, is $3\times(2-1)+3\times(2-1)+4\times(3-1)+2\times(3-1)+11\times(2-1)+9\times(2-1)+7\times(2-1)+1\times(2-1)+7\times(2-1)+4\times(2-1)=57$.

The clustering hypergraph models for the junction- and link-based storage schemes are accurate as long as the queries in the previous query log tend to reappear in the current time window. Disk pages can be periodically reorganized to capture the characteristics of query logs in different time windows. Furthermore, incremental clustering approaches can be adapted to reflect the changes in time.

4.4.3 Comparison of Clustering Hypergraph Models

The clustering hypergraph models for the junction- and link-based storage schemes are closely related in representing a given road network for solving the record-to-page allocation problem under the respective storage scheme. In both clustering hypergraphs, vertices represents the records, whereas nets represent the successor retrieval operations. The set of vertices connected by a net correspond to the set of records concurrently accessed by the respective operation. Vertex weights correspond to records sizes, whereas net costs correspond to

the frequency of the respective network operation. In both models, records are clustered into disk pages by partitioning the respective hypergraph, where the partitioning objective corresponds to minimizing the disk access cost of successor retrieval operations in network queries. The topological difference between these two hypergraph models stems from the difference between the two storage schemes. Topologically, vertices correspond to junctions and links in the former and latter hypergraph models, respectively.

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

$$\begin{aligned} |\mathcal{V}_{\mathrm{T}}| &= |\mathcal{T}|, \\ \lceil |\mathcal{L}|/2 \rceil + |\mathcal{T}| - \alpha \leq |\mathcal{N}_{\mathrm{T}}| &\leq |\mathcal{L}| + |\mathcal{T}| - \alpha, \\ 2\lceil 1.5\,|\mathcal{L}| \rceil + |\mathcal{T}| - \alpha \leq |\mathcal{H}_{\mathrm{T}}| &\leq 3|\mathcal{L}| + |\mathcal{T}| - \alpha. \end{aligned} \tag{4.19}$$

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

$$\begin{aligned} |\mathcal{V}_{\mathrm{L}}| &= |\mathcal{L}|, \\ |\mathcal{N}_{\mathrm{L}}| &= \sum_{t_i \in \mathcal{T}} (d_{\mathrm{in}}(t_i) \times d_{\mathrm{out}}(t_i)) - \beta + |\mathcal{L}| - \sum_{t_i \in \mathcal{T}, d_{\mathrm{out}}(t_i)=0} d_{\mathrm{in}}(t_i), \\ |\mathcal{H}_{\mathrm{L}}| &= 3 \sum_{t_i \in \mathcal{T}} (d_{\mathrm{in}}(t_i) \times d_{\mathrm{out}}(t_i)) + \sum_{t_i \in \mathcal{T}, d_{\mathrm{out}}(t_i)>0} d_{\mathrm{in}}(t_i) - 2\beta. \end{aligned} \tag{4.20}$$

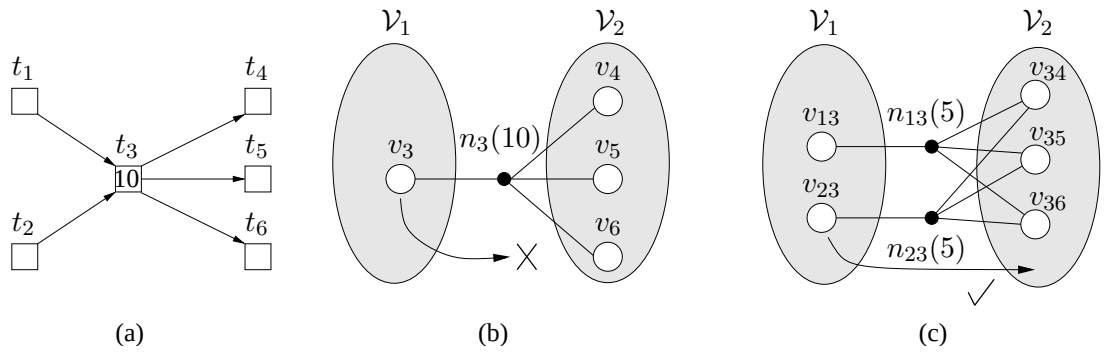


Figure 4.4: (a) A sub-network with $GS(t_3)$, (b) $\mathcal{H}_{\mathrm{T}}$: a four-pin net n_3 for the $GS(t_3)$ operations with $f(t_3)=10$, (c) $\mathcal{H}_{\mathrm{L}}$: two four-pin nets n_{13} for the $GS(\ell_{13})$ operations with $f(\ell_{13})=5$ and n_{23} for the $GS(\ell_{23})$ operations with $f(\ell_{13})=5$.

In this work, we claim that the clustering hypergraph model provides more flexibility in partitioning for the link-based storage scheme compared to the junction-based storage scheme. We illustrate this by the following example. Fig. 4.4(a) shows a sample sub-network $(\mathcal{T},\mathcal{L})$ with a junction t_3 having two incoming and three outgoing links. Figs. 4.4(b) and 4.4(c) show the net-induced subhypergraphs $(\mathcal{V}_{\mathrm{T}},\mathcal{N}_{\mathrm{T}}^{GS})$ and $(\mathcal{V}_{\mathrm{L}},\mathcal{N}_{\mathrm{L}}^{GS})$ corresponding to the sub-network given in Fig. 4.4(a) for the junction- and link-based storage schemes, respectively. Ten *GS* operations are assumed to be performed on junction t_3, five *GS* operations for each incoming link of t_3. As seen in the figure, junction t_3 induces only one net n_3 in $\mathcal{H}_T$, whereas the two incoming links ℓ_{13} and ℓ_{23} of t_3 induce nets n_{13} and n_{23} in $\mathcal{H}_L$. Figs. 4.4(b) and 4.4(c) also show 2-way partitions for $\mathcal{H}_T$ and $\mathcal{H}_L$. In this example, if there were no part size constraints, moving vertex v_3 from $\mathcal{V}_1$ to $\mathcal{V}_2$ would remove net n_3 from the cut, thus reducing the cutsize by 10. However, this move may not be feasible due to the maximum part size constraint on $\mathcal{V}_2$. Since the record sizes in the link-based storage scheme are less than those in the junction-based storage scheme as shown in Section 4.2, either v_{13} or v_{23} can move to $\mathcal{V}_2$ without violating the maximum part size constraint, respectively removing n_{13} or n_{23} from the cut with a saving of 5 on the cutsize. In general, the partitioning of the clustering hypergraph for the link-based storage scheme has a better solution space as there is greater flexibility in moving vertices between parts.

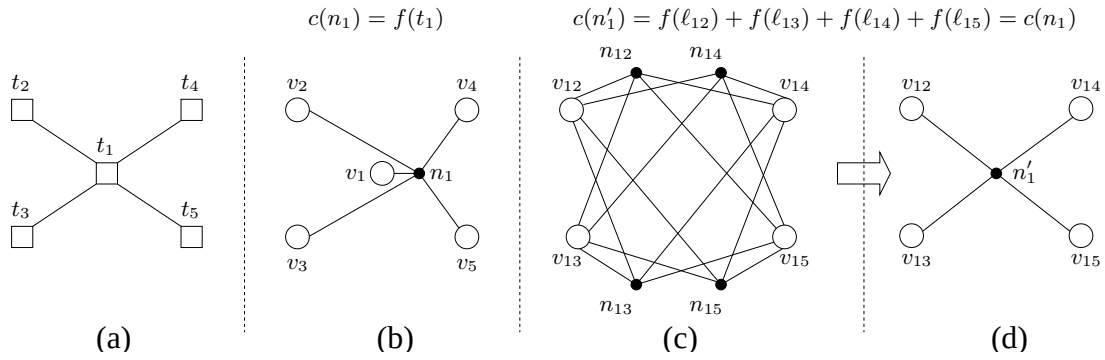


Figure 4.5: (a) A bidirectional sub-network with $GS(t_1)$, (b) $\mathcal{H}_{\mathrm{T}}$: a five-pin net n_1 for the $GS(t_1)$ operations with $c(n_1) = f(t_1)$, (c) $\mathcal{H}_{\mathrm{L}}$: four identical four-pin nets n_{12}, n_{13}, n_{14}, and n_{15} for $GS(\ell_{12})$, $GS(\ell_{13})$, $GS(\ell_{14})$, and $GS(\ell_{15})$, respectively, (d) $\mathcal{H}_{\mathrm{L}}$: identical nets n_{12}, n_{13}, n_{14}, and n_{15} coalesced into net n'_1 with cost $c(n'_1) = c(n_1)$.

In bidirectional networks, the storage saving in the link-based scheme results in higher improvements in query processing performance compared to the junction-based scheme. We provide Fig. 4.5 to validate this claim. Fig. 4.5(a) shows a sample sub-network $(\mathcal{T}, \mathcal{L})$ with a junction t_1 having four bidirectional incoming/outgoing links. Figs. 4.5(b) and 4.5(c) show the net-induced subhypergraphs $(\mathcal{V}_{\mathrm{T}}, \mathcal{N}_{\mathrm{T}}^{GS})$ and $(\mathcal{V}_{\mathrm{L}}, \mathcal{N}_{\mathrm{L}}^{GS})$ corresponding to the sub-network for the junction- and link-based storage schemes, respectively. Note that the sum of the number of GS operations performed on the incoming links of junction t_1 in the link-based storage scheme is equal to the number of GS operations performed on junction t_1. That is, $f(\ell_{21}) + f(\ell_{31}) + f(\ell_{41}) + f(\ell_{51}) = f(t_1)$.

As seen in Fig. 4.5(b), in $\mathcal{H}_{\mathrm{T}}$, for the $GS(t_1)$ operation, there is a five-pin net with $\mathrm{Pins}(n_1) = \{v_1, v_2, v_3, v_4, v_5\}$ and $c(n_1) = f(t_1)$. In the construction of the clustering hypergraph for the link-based storage scheme, two directional links between the same junctions (i.e., ℓ_{ij} and ℓ_{ji}) are represented with a bidirectional link ℓ_{ij}, where $i < j$. Hence, a vertex v_{ij} exists for each record r_{ij} storing link ℓ_{ij}. As seen in Fig. 4.5(c), $\mathcal{H}_{\mathrm{L}}$ has four four-pin nets n_{12}, n_{13}, n_{14}, and n_{15} to capture the costs of the $GS(\ell_{21})$, $GS(\ell_{31})$, $GS(\ell_{41})$, and $GS(\ell_{51})$ operations, respectively. Note that these four four-pin nets connect the same set of pins, i.e., $\mathrm{Pins}(n_{12}) = \mathrm{Pins}(n_{13}) = \mathrm{Pins}(n_{14}) = \mathrm{Pins}(n_{15}) = \{v_{12}, v_{13}, v_{14}, v_{15}\}$. Such nets, which connect exactly the same set of pins, are called identical nets. Identical nets can be

coalesced into a single representative net. The representative net's cost is set to the total cost of all constituting nets. Here, n_{12}, n_{13}, n_{14}, and n_{15} can be coalesced into a representative net n'_1 with $\text{Pins}(n'_1) = \{v_{12}, v_{13}, v_{14}, v_{15}\}$ and $c(n'_1) = c(n_{12}) + c(n_{13}) + c(n_{14}) + c(n_{15})$ as shown in Fig. 4.5(d). Comparison of Figs. 4.5(b) and 4.5(d) shows that, for *GS* operations, the clustering hypergraphs for the two storage schemes have the same set of nets with equal costs. However, the size of each net in $\mathcal{H}_{\text{L}}$ is one less than the size of the respective net in $\mathcal{H}_{\text{T}}$. This finding conforms with the fact that, in query processing, each *GS* operation in the link-based storage scheme accesses one record less compared to the junction-based storage scheme. Thus, the partitioning of $\mathcal{H}_{\text{L}}$ is expected to lead to smaller cutsize compared to that of $\mathcal{H}_{\text{T}}$ because of smaller net sizes in the link-based storage scheme.

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

$$\begin{aligned} |\mathcal{V}_{\text{T}}| &= |\mathcal{T}|, \\ |\mathcal{N}_{\text{T}}| &= |\mathcal{L}|/2 + |\mathcal{T}|, \\ |\mathcal{H}_{\text{T}}| &= 2|\mathcal{L}| + |\mathcal{T}|, \end{aligned} \tag{4.21}$$

and

$$\begin{aligned} |\mathcal{V}_{\text{L}}| &= |\mathcal{L}|/2, \\ |\mathcal{N}_{\text{L}}| &= \sum_{t_i \in \mathcal{T}} d(t_i)^2 - |\mathcal{L}| + |\mathcal{T}| - \tau, \\ |\mathcal{H}_{\text{L}}| &= 2 \sum_{t_i \in \mathcal{T}} d(t_i)^2 - |\mathcal{L}| - \tau, \end{aligned} \tag{4.22}$$

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

4.5 Summary

We introduced the link-based storage scheme and presented a detailed comparative analysis on the properties of junction- and link-based storage schemes. In

the link-based storage scheme, each record stores the data associated with a link together with the link's connectivity information. We proposed a clustering hypergraph model for the link-based storage scheme to partition the network data into disk pages where data would be periodically reorganized using the previous query logs. Our detailed comparative analysis on the properties of the junction- and link-based storage schemes showed that the link-based storage scheme is more amenable to clustering. Moreover, we introduced storage enhancements for bidirectional networks. We showed that the link-based storage scheme is more amenable to our enhancements than the junction-based storage scheme and results in better data allocation for processing aggregate network queries.

Extensive experimental comparisons were carried out on the effects of page size, buffer size, path length, record size, and dataset size for the junction- and link-based storage schemes. In Chapter 6, we present and discuss the simulation results of the proposed model in detail. Experimental results showed that the link-based storage scheme outperforms the widely-used junction-based storage scheme in terms of both storage and query processing efficiency.

Chapter 5

Efficient Successor Retrieval Operations

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. However, efficient implementation of the *GS* operation is overlooked in the literature.

In this chapter, we first 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 used in the target application. Second, for the junction-based storage scheme, we propose a hypergraph-based model for clustering concurrently 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 [31] and the incremental

network expansion framework [59]. The results of our experimental simulations in Section 6.5 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.

The organization of this chapter is as follows: We introduce and discuss the proposed *GUS* operation in Section 5.1. Section 5.2 describes the clustering hypergraph model proposed for the *GUS* operations. Finally, we summarize the chapter in Section 5.3.

5.1 *Get-Unevaluated-Successors* (*GUS*)

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 $GUS(t, \mathrm{Suc}(t, U))$ operation, where U denotes the set of unevaluated junctions just before the invocation of the *GUS* operation for the current query. The set

$$\mathrm{Suc}(t_i, U) = \{t_j : t_j \in \mathrm{Suc}(t_i) \wedge t_j \in U\} \tag{5.1}$$

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

The *GuPS* operation typically arises in Dijkstra's single source shortest path algorithm [31]. 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, we briefly show the pseudocode of the Dijkstra's single source shortest path algorithm [31] 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 [31], 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.

The *GuVS* operation typically arises in algorithms using the incremental network expansion (INE) framework [59]. Algorithms using INE 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, we briefly show the pseudocode of the k-nearest

Algorithm 1 Dijkstra'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 **do**
7: $t_i \leftarrow$ EXTRACT_MIN(Q)
8: $GuPS(t_i, \text{Suc}(t_i, Q))$
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($Q, t_j, dist[t_j]$)
14: $previous[t_j] \leftarrow t_i$
15: return $previous[]$

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

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

neighbor search using the incremental network expansion framework [59] 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. Points of interest 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 INE 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

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: $t_i t_j \leftarrow find\_segment(q)$
3: $S_{cover} \leftarrow find\_entities(t_i t_j)$
4: $\{p_1, \cdots, p_k\} = k$ 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$(Q, < (t_i, d_N(q,t_i)), (t_j, d_N(q,t_j)) >)$
7: $t_i \leftarrow$ EXTRACT_MIN(Q)
8: $V \leftarrow V - \{t_i\}$
9: **while** $(d_N(q,t_i) < d_{Nmax})$ **do**
10: $GuVS(t_i, \text{Suc}(t_i, V))$
11: **for** each retrieved page $P_i \in PageSet$ **do**
12: **for** each successor $t_j \in P_i$ of t_i **do**
13: $V \leftarrow V - \{t_j\}$
14: $S_{cover} \leftarrow find\_entities(t_i t_j)$
15: update $\{p_1, \cdots, p_k\}$ from $\{p_1, \cdots, p_k\} \cup S_{cover}$
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 $\{p_1, \cdots, p_k\}$

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

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 $PageSet$

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.

5.2 Clustering Hypergraph Model for *GUS* Operations

Here, we present our clustering hypergraph model, which correctly captures the cost of *GUS* operations for the junction-based storage scheme.

5.2.1 Hypergraph Construction

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

$$c(n_{\text{Suc}(t_i,U)}) = f(t_i, \text{Suc}(t_i, U)). \tag{5.2}$$

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

$$\text{Pins}(n_{\text{Suc}(t_i,U)}) = \{v_i\} \cup \{v_j : t_j \in \text{Suc}(t_i, U)\}. \tag{5.3}$$

Note that the size of net $n_{\text{Suc}(t_i,U)}$ can be between 2 and $d_{out}(t_i) + 1$ since $|\text{Suc}(t_i, U)| \leq d_{out}(t_i)$. Single pin nets are discarded since $GUS(t_i, \text{Suc}(t_i, U))$ operations with $\text{Suc}(t_i, U) = \emptyset$ do not incur any record access. Fig. 5.1 displays the net construction for a $GUS(t_1, \text{Suc}(t_1, U))$ operation invoked on junction t_1 with $Suc(t_1) = \{t_2, t_3, t_4, t_5\}$ but $\text{Suc}(t_1, U) = \{t_3, t_5\}$.

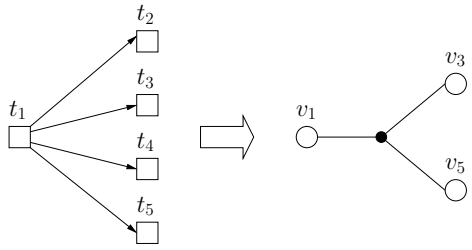


Figure 5.1: The clustering hypergraph construction: $GUS(t_1, \{t_3, t_5\})$ incurs a net with pins $\{v_1, v_3, v_5\}$.

The size of hypergraph $\mathcal{H}_{\text{GUS}}$ 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 $\mathcal{H}_{\text{GUS}}$. Recall that $GS(t_i)$ operations invoked on junction t_i incur a single net of size $d_{out}(t_i) + 1$ in $\mathcal{H}_{\text{GS}}$ 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 $\mathcal{H}_{\text{GUS}}$ 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 [12]. 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 $\mathcal{H}_{\text{GS}}$ using the query log is a rather trivial task, generation of $\mathcal{H}_{\text{GUS}}$ may need special attention. As in the GS case, we assume 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 $q_i =< t_{i_1}, t_{i_2}, \cdots, t_{i_k}, \cdots >$ denote the sequence of junctions accessed during processing a query q_i in the log. Then, k-th junction t_{i_k} in q_i corresponds to the $GUS(t_{i_k}, \text{Suc}(t_{i_k}, U_{ik}))$ operation, where U_{ik} represents the set of unevaluated junctions just before the invocation of $GUS(t_{i_k}, \text{Suc}(t_{i_k}, U_{ik}))$ in query q_i.

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

$$U_{ik} = \mathcal{T} - q_{ik}, \tag{5.4}$$

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

Require: Query log $Q_{\log} =< q_1, q_2, \cdots, q_n >$, where $q_i =< t_{i_1}, t_{i_2}, \cdots, t_{i_m} >$
1: $U \leftarrow \mathcal{T}$ ▷ U denotes the set of unprocessed junctions
2: **for** each query q_i in $Q_{\log}$ **do**
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: $f(t_j, \text{Suc}(t_{i_k}, U)) \leftarrow f(t_j, \text{Suc}(t_{i_k}, U)) + 1$

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

Require: Query log $Q_{\log} =< q_1, q_2, \cdots, q_n >$, where $q_i =< t_{i_1}, t_{i_2}, \cdots, t_{i_m} >$
1: $U \leftarrow \mathcal{T}$ ▷ U denotes the set of unvisited junctions
2: **for** each query q_i in $Q_{\log}$ **do**
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: $U \leftarrow U - \{t_j\}$
7: **if** $t_j \in U$ **then**
8: $f(t_j, U) \leftarrow f(t_j, U) + 1$

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

$$U_{ik} = \mathcal{T} - q_{ik} - \bigcup_{t_j \in q_{ik}} \text{Suc}(t_j). \tag{5.5}$$

Here, $q_{ik} =\!< t_{i_1}, t_{i_2}, \cdots, t_{i_k} >$ denotes the k-th prefix subsequence of q_i. Note that the junction subsequence q_{ik} is also used as a junction subset in (5.4) and (5.5). 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, we maintain a list of *GUS* operations together with their frequencies for each junction. Each operation $GUS(t_i, \text{Suc}(t_i, U))$ 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 $\text{Suc}(t_i, U)$. In this way, locating a $GUS(t_i, \cdot)$ operation for incrementing its frequency count requires m_i

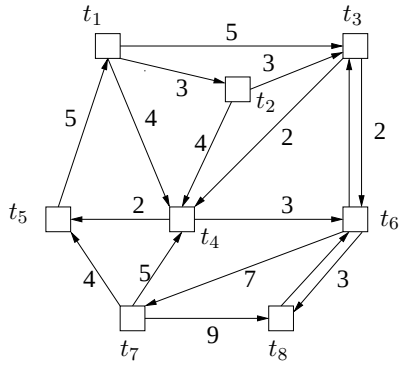
byte comparisons in the list for t_i, where m_i denotes the number of $GUS(t_i, \cdot)$ operations encountered so far in the query log.

5.2.2 Hypergraph Model

The constructed clustering hypergraph $\mathcal{H}_{\text{GUS}} = (\mathcal{V}, \mathcal{N}_{\text{GUS}})$ is partitioned into parts $\Pi = \{\mathcal{V}_1, \cdots, \mathcal{V}_k, \cdots\}$ 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., $W_{max} = P$ (see Section 3.4).

Here, we will show that the partitioning objective of minimizing the cutsize according to (2.2) corresponds to minimizing the total number of disk accesses incurred by the *GUS* operations under the single-page buffer assumption. Consider an internal net n_i of part $\mathcal{V}_k$ in partition Π. As seen in (2.2), 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 (2.2), 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.

Fig. 5.2 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. 5.2 also illustrates a



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

Figure 5.2: A sample road network with a query set.

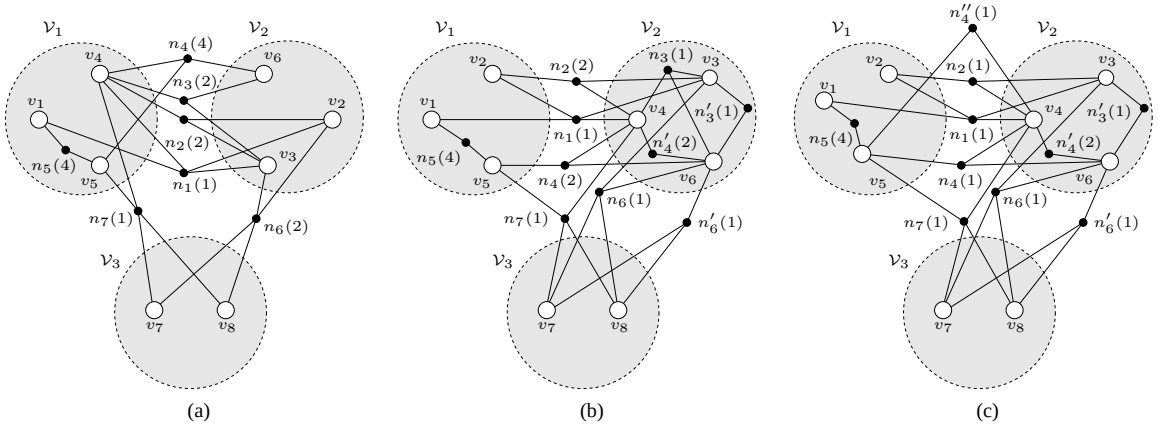


Figure 5.3: Clustering hypergraphs (a) $\mathcal{H}_{\text{GS}}$, (b) $\mathcal{H}_{\text{GuPS}}$, and (c) $\mathcal{H}_{\text{GuVS}}$ for the sample road network in Fig. 5.2 and 3-way vertex partitions of these hypergraphs.

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, we assume that the sequence of processed junctions are the same in the three clustering hypergraph models using $\mathcal{H}_{\text{GS}}$, $\mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$.

In Fig. 5.3, we illustrate the clustering hypergraphs $\mathcal{H}_{\text{GS}}$, $\mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$ for the sample road network given in Fig. 5.2. Fig. 5.3 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. 5.3(a). The cut net n_4 with $\text{Pins}(n_4) = \{v_4, v_5, v_6\}$ 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. Here, we will show that 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. 5.3(b). As seen in Fig. 5.3(b), *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 $< t_2, t_8 >$ and $< t_4, t_3 >$. 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 $< t_5, t_6 >$ and $< t_7, t_6 >$.

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. 5.3(b). 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. Here, we will show that 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. 5.3(a) is used instead

of the one induced by the partition in Fig. 5.3(b), *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. 5.3(c). 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 $<t_5, t_6>$, *GuVS* operations invoked on junctions t_2 and t_3 do not incur any net. As seen in Fig. 5.3(c), *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 $<t_4, t_3>$. 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 $<t_5, t_6>$ and $<t_7, t_6>$. 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 $<t_2, t_8>$.

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. 5.3(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 Figs. 5.3(b) and (c). Here, we will show that 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 accuracy of the clustering hypergraph models depend on the repetition of the past queries since the record access frequencies in these models are obtained from the query log. Disk pages can be periodically reorganized to capture the disk access cost of queries using logs from different time windows. Moreover, changes in time can be integrated in our models with the usage of incremental clustering approaches.

5.3 Summary

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. We showed two examples of *GUS* operation in network query processing, namely the *Get-unProcessed-Successors* (*GuPS*) operation as used in the Dijkstra's single source shortest path algorithm [31] and the *Get-unVisited-Successors* (*GuVS*) operation as used in the algorithms utilizing the incremental network expansion framework [59].

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. In Chapter 6, we present and discuss the simulation results of the proposed model in detail. Experimental results showed 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 a road network database.

Chapter 6

Experimental Results

6.1 Experimental setup

Experiments are conducted on a wide range of real-life road network datasets collected from U.S. Tiger/Line [15] (Minnesota7 including 7 counties Anoka, Carver, Dakota, Hennepin, Ramsey, Scott, Washington; Sanfrancisco; Oregon; NewMexico; Washington), U.S. Department of Transportation [30] (California Highway Planning Network), and Brinkhoff's data files [9] (Oldenburg; San-Joaquin). These datasets are primarily composed of points and polylines connecting the points. Since there is no embedded direction information in these datasets, 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 datasets 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 datasets, whose properties are given in Table 6.1, where the datasets are listed in the order of increasing network size. In Table 6.1, $|\mathcal{T}|$ and $|\mathcal{L}|$ denote the number of junctions and links, respectively, and d_{avg} denotes the average number of predecessors and successors of a junction.

Table 6.1: Properties of road network datasets

Dataset	Name	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{L}\rvert$	d_{avg}
D0	Oldenburg	4465	10 778	2.41
D1	California HPN	10 141	28 370	2.80
D2	SanJoaquin	17 444	45 974	2.64
D3	Minnesota7	34 222	92 206	2.69
D4	Sanfrancisco	166 558	426 742	2.56
D5	NewMexico	448 959	1 112 230	2.48
D6	Oregon	507 212	1 203 344	2.37
D7	Washington	548 901	1 304 126	2.38

It is important to note that all links in our datasets are bidirectional. This enables the use of the storage savings mentioned in Sections 3.1 and 4.1. In the junction-based storage scheme, we store only the successor list of each junction. In the link-based storage scheme, we combine the records storing the two directional links between two junctions into a single record and hence halve the number of records.

We generated synthetic query set for each dataset in order to be able to obtain a cost distribution over the nets of the constructed hypergraphs. For this purpose, a set of source and destination junction pairs, which have a predetermined shortest path length, is generated by slightly modifying the network-based node selection option of Brinkhoff's Network Generator for Moving Objects [8]. Queries that traverse the junctions on the shortest paths between the source and destination junction pairs are added into the query set as route evaluation queries. Queries that seek the shortest paths (using Dijkstra's algorithm) are added into the query set as path computation queries. The number of queries is set to be the same in both route evaluation and path computation queries in Sections 6.3 and 6.4. In Section 6.5, we conducted experiments with only path computation queries to construct and evaluate our model for efficient successor retrieval operations.

The clustering hypergraphs are constructed as described in Sections 3.3.1, 4.4.1, and 5.2.1. The vertex weights are set to be equal to the size of the respective records. The state-of-the-art tools MeTiS [51] and PaToH [14] are used with default parameters for bipartitioning the clustering graphs and hypergraphs, respectively. The running time of the hypergraph partitioning tool PaToH is $O(\log|\mathcal{V}|\sum_{n_j\in\mathcal{N}}|n_j|^2)$ at each bisection step of the recursive bipartitioning scheme, where $\mathcal{V}$ and $\mathcal{N}$ denote the vertex and net sets of the remaining

hypergraph at that bisection step (see the net splitting process used in the recursive bipartitioning (RB) in [14]). In terms of network parameters, the running time of the first bisection step is $O(d_{\text{avg}}^2 |\mathcal{T}| \log |\mathcal{T}|)$ and $O(d_{\text{avg}}^2 |\mathcal{L}| \log |\mathcal{L}|)$ for the junction- and link-based storage schemes, respectively, under the simplifying assumption that the number of incoming and outgoing links for each junction are both equal to $d_{\text{avg}} = |\mathcal{L}|/|\mathcal{T}|$. Assuming a balanced recursive bisection tree for our RB schemes in Section 3.4 and number K of pages, the overall running time becomes $O(d_{\text{avg}}^2 |\mathcal{T}| \log |\mathcal{T}| \log K)$ and $O(d_{\text{avg}}^2 |\mathcal{L}| \log |\mathcal{L}| \log K)$ for the junction- and link-based storage schemes, respectively. However, these are rather loose upper bounds and the partitioning tool PaToH is quite fast while generating high quality results. For example, the overall RB2-based partitioning times for the D1, D2, D3, and D4 datasets are respectively 3.2, 5.6, 26.7, and 317.1 seconds, on the average, including the read/write operations of input/output files. These timings are reported on a PC that is equipped with an Intel Pentium IV 2.6 GHz processor and 2GB of RAM, and hypergraph representations for all datasets and parameters fit into the main memory.

The deviation ratio ϵ in (3.5) and (3.6) is set to 0.10 throughout the experiments. We conducted experiments with four page sizes of $P = 1$, 2, 4, and 8 KB in Sections 6.3 and 6.4. In Section 6.5, we conducted experiments with page sizes of $P = 4$, 8, 16, and 32 KB in order to show the advantages of efficient successor retrieval operations with larger page sizes. Here, the allocation of the records of a given dataset for a given page size is referred to as an allocation instance. Due to the randomized nature of the partitioning heuristics, the experiment for each allocation instance is repeated 10 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 successor retrieval operations on the total number of disk accesses incurred by the aggregate network queries. We implement a hash-based index on junction/link ids for efficient record retrieval and an R-tree index on links for point queries. The lookup cost of these indices for *Find* operations is included in our simulation results showing the total number of disk accesses for query processing. In order to evaluate the effect of disk caching

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

6.2 Implementation Details

In order to clarify how we handled the data retrieval in our simulations we describe each case as follows: Recall that if the disk page that has the target record is already stored in the memory, then it causes a page hit and the record is used from the cache. Otherwise, the disk page is retrieved from the secondary storage. In this case, there are three possibilities:

- We are given a query point (x, y) to find the network element matching these coordinates. The R-tree is used to find the disk address of the link, i.e., the first *Find* operation in the course of query processing.
- We are given a network element which already resides in the memory due to prior operations, and we are asked to find a successor of that record. The hash-based index is used to retrieve the successor record from the secondary storage, i.e., *Get-a-Successor* ($GaS(t_i, t_j)$), *Get-Successors* ($GS(t_i)$), *Get-unProcessed-Successors* ($GuPS(t_i, \mathrm{Suc}(t_i, U))$), *Get-unVisited-Successors* ($GuVS(t_i, \mathrm{Suc}(t_i, U))$). We should mention here that, for this case, the use of a hash-based index can be avoided by storing the disk addresses of the records in the successor/predecessor lists of the

respective network elements. However, this scheme introduces additional complexity during the reorganization and update operations.

- We are given a network element with its id. The hash-based index is used to locate the disk address of the record, i.e., $Find(t_i)$, $GaS(t_i, t_j)$, $GS(t_i)$, $GuPS(t_i, \mathrm{Suc}(t_i, U))$, $GuVS(t_i, \mathrm{Suc}(t_i, U))$.

For each network query, the R-tree index is used only once to realize the very first *Find* operation to locate the source link. All following data retrieval operations (i.e., *Find*, *Get-a-Successor*, *Get-Successors*, and *Get-Unevaluated-Successors*) needed during query processing are performed by using a hash-based index with an average cost of single disk access for each retrieval request if the network element does not already reside in the memory. Please note that, disk access costs regarding the indices are already included in our simulation results. The way the R-tree structure is constructed may affect only the cost of the very first $Find$ operation. In our simulations, we assume an R*-tree [6] implementation with an average lookup cost of a single disk access.

6.3 Evaluation of Junction-Based Storage Scheme

In this section, we first describe query generation and properties of the constructed graphs and hypergraphs. 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 *GS* operations, and the total number of disk accesses in aggregate network queries.

6.3.1 Query Generation

The clustering graph and hypergraph models are constructed based on the assumption that network usage frequencies can be obtained via utilizing the query

Table 6.2: Average number of junctions accessed in queries

Dataset	Route evaluation	Path computation
D0	11	109
D1	16	217
D2	19	419
D3	31	1207
D4	48	3577
D5	99	8767
D6	105	10 214
D7	107	12 696

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 using a synthetic-query generation approach. Specifically, Brinkhoff's Network Generator for Moving Objects framework [8] 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 used, whereas for path computation queries, Dijkstra's single source shortest path algorithm is executed between the source and destination junctions. For each dataset, 1000 queries are generated. These queries are used for first generating the query sets 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 dataset. The average number of junctions accessed in the queries for each dataset is given in Table 6.2.

6.3.2 Properties of Graphs and Hypergraphs

Table 6.3 displays the properties of generated clustering graphs and hypergraphs used in the experiments. As described in Section 3.3, in our model, vertex weights correspond to record sizes in bytes. Throughout the experiments, we reserve 12 bytes for link attributes (i.e., $C_{\mathrm{L}} = 12$) and 4 bytes for the coordinates of a junction (i.e., $C_{\mathrm{id}} = 4$). The size of coordinates of a junction is highly dependent on the specification of the coordinate system, and it is possible to compress the coordinates to small values. In the experimental section, 4 bytes is found to be sufficient to represent all coordinates in our datasets. No space is reserved for junction attributes (i.e., $C_{\mathrm{T}} = 0$). The average vertex weight, w_{avg}, and the total vertex weight, W_{total}, for each dataset are also given in the table. Recall that,

Table 6.3: Properties of generated clustering graphs and hypergraphs

	$\mathcal{G}$		$\mathcal{H}$				
Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{H}\rvert$	w_{avg}	W_{total}
D0	4465	5389	4465	7886	22 085	42.62	190 308
D1	10 141	14 181	10 141	17 369	52 965	48.76	494 484
D2	17 444	22 801	17 444	26 225	80 785	46.17	805 360
D3	34 222	45 419	34 222	46 056	149 493	47.11	1 612 184
D4	166 558	212 916	166 558	197 630	654 990	44.99	7 494 104
D5	448 959	554 448	448 959	510 477	1 682 429	43.64	19 591 516
D6	507 212	601 340	507 212	574 353	1 844 579	41.96	21 282 352
D7	548 901	650 268	548 901	613 558	1 980 535	42.01	23 051 620

since all of the links in the datasets 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.

6.3.3 Comparison of RB1 and RB2 schemes

In Fig. 6.1, 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. 6.1, 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 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. 6.2, 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

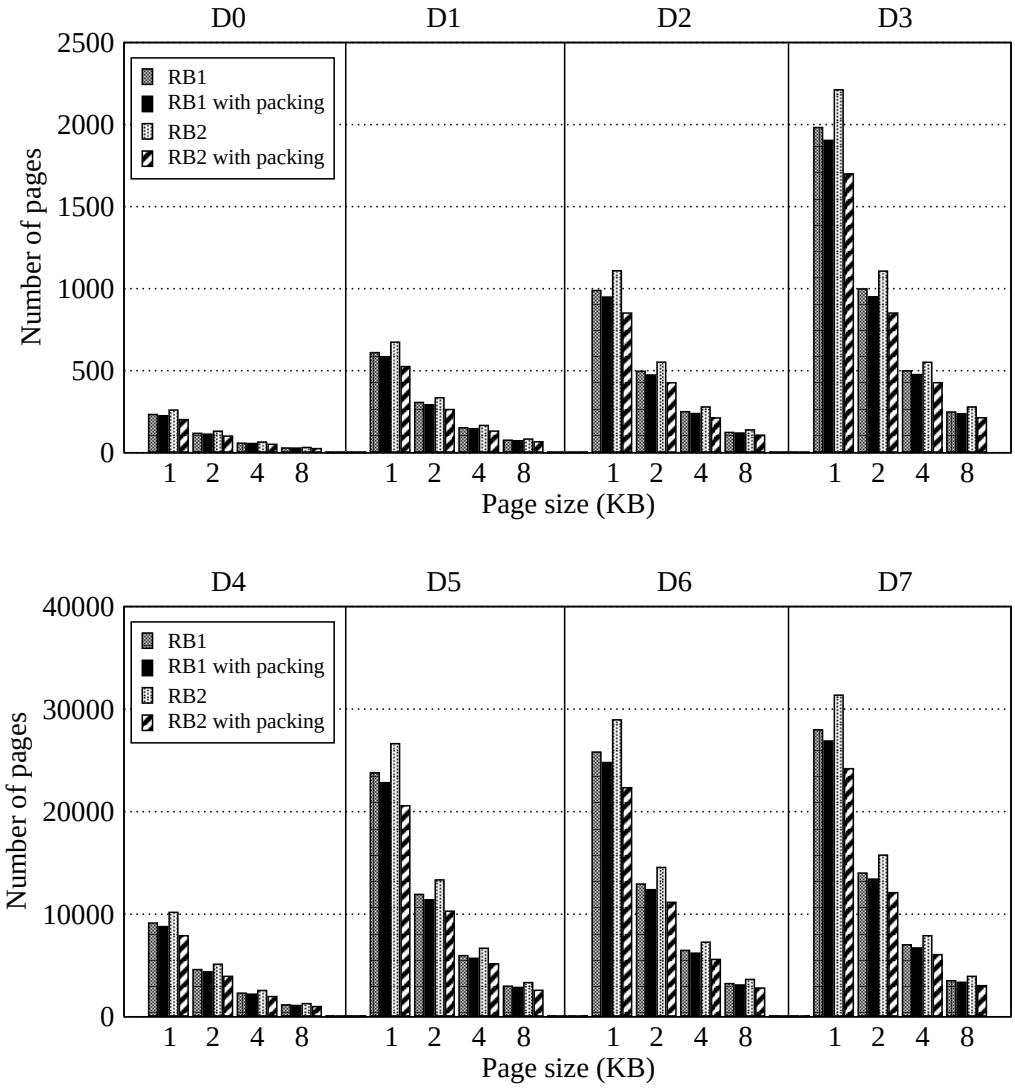


Figure 6.1: Comparison of RB1 and RB2 schemes for D0–D7 datasets in terms of the number of allocated pages

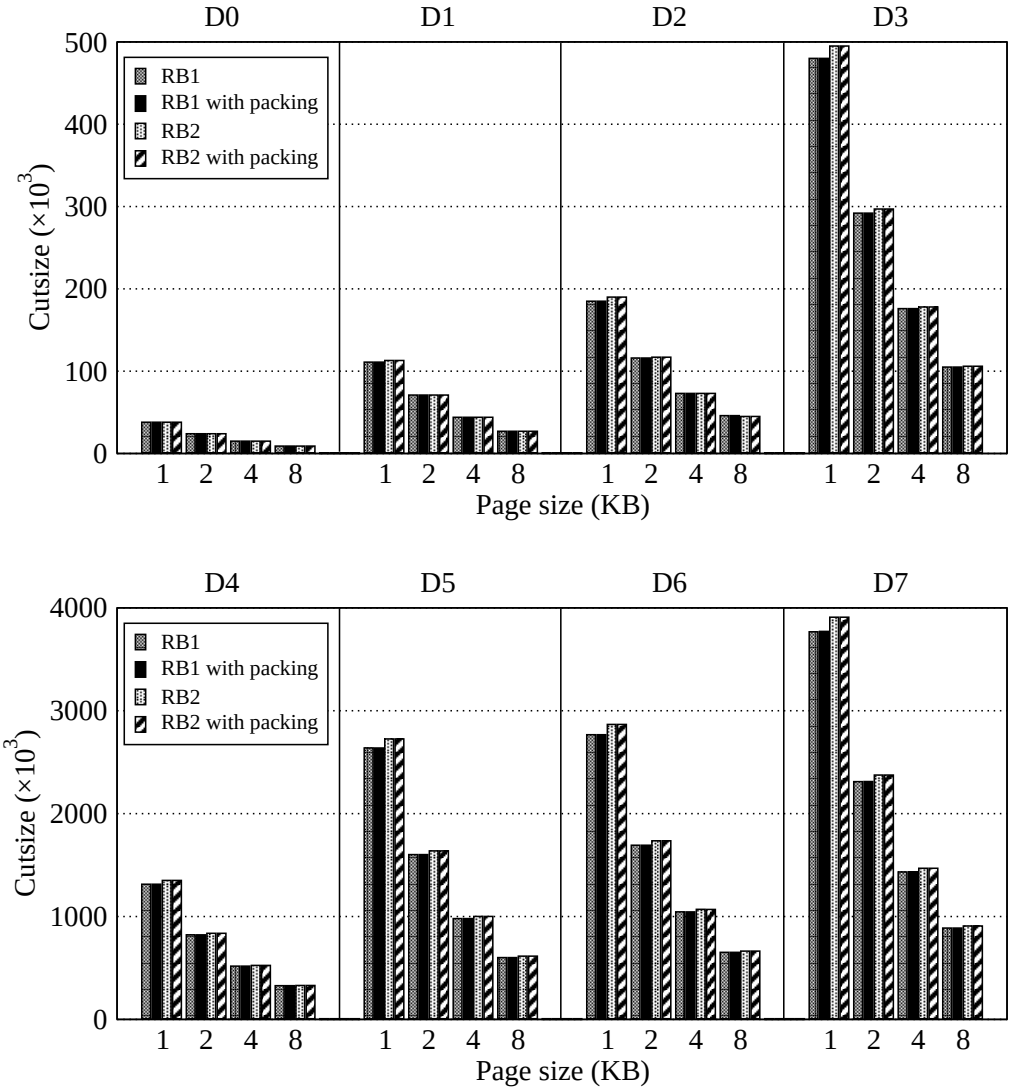


Figure 6.2: Comparison of RB1 and RB2 schemes for D0–D7 datasets in terms of the cutsize

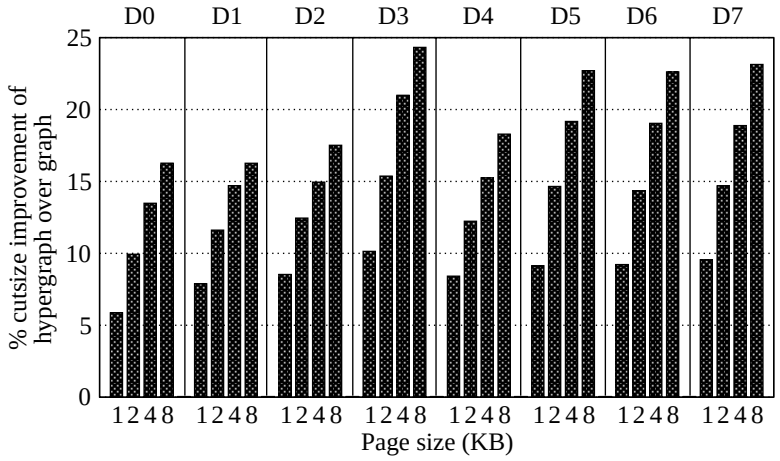


Figure 6.3: Percent cutsize improvement of the clustering hypergraph model over the clustering graph model

while yielding 1.7% higher cutsize values than RB1 with packing on the average. In the following sections, 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.

6.3.4 Comparison of Clustering Graph and Hypergraph Models

Fig. 6.3 displays the variation in the percent cutsize improvement of the clustering hypergraph model over the clustering graph model with increasing page size. Recall that cutsize values encapsulate the total disk access costs of *GaS* and *GS* 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. 6.3, the hypergraph model performs better than the graph model in terms of the cutsize for every allocation instance. In all datasets, 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 deficiency mentioned in Section 3.2.3. On the average over all allocation instances, the performance improvement of the hypergraph model over the graph model is 14.7%.

In Fig. 6.4, 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 B. Recall that the cost of aggregate network query processing includes the costs of GaS and GS 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 datasets, 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 datasets D0 and D1 using the largest buffer size of 8 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 GS 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 GS operations is 14.7%, the average improvement in the total disk access

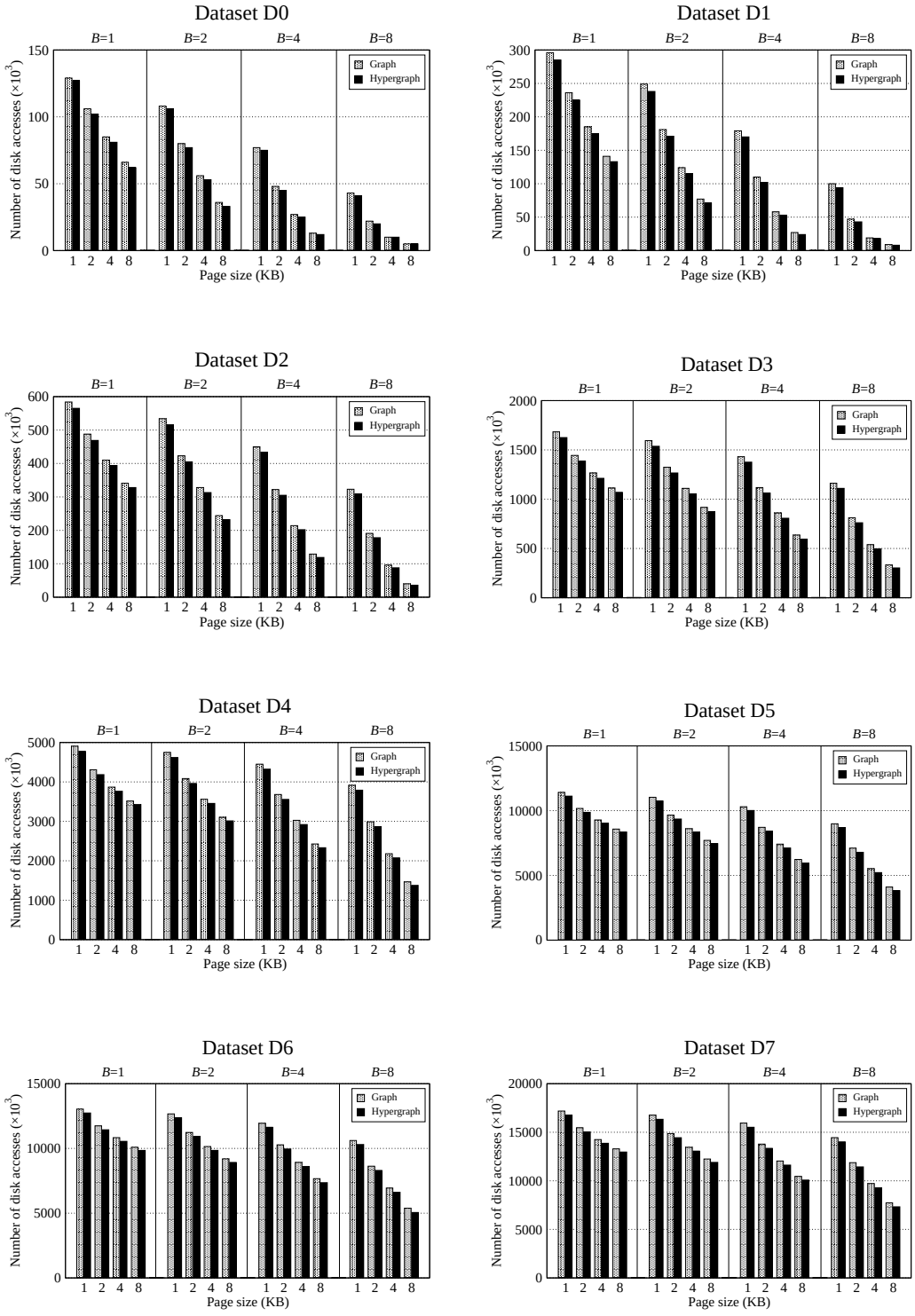


Figure 6.4: The total disk access costs of aggregate network queries for each dataset in the clustering graph and hypergraph models with increasing page size P in KB and page buffer size B in number of pages.

cost of aggragate 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.

6.4 Evaluation of Link-Based Storage Scheme

In this section, we first describe synthetic query generation step. Then, we compare the storage size of the junction- and link-based storage schemes and present the properties of clustering hypergraphs. We evaluate the performance of the clustering hypergraph models for the junction- and link-based storage schemes in two aspects. In Section 6.4.4, we evaluate the partition quality in terms of cutsize, which corresponds to the total number of disk accesses incurred by *GaS* and *GS* operations under the single-page buffer assumption. In Section 6.4.5, we assess the total number of disk accesses in aggregate network queries through simulations.

6.4.1 Query Generation

In order to span most elements in the network and hence to create a hypergraph large enough to represent the network, we adaptively determined a separate query count and a path length for each dataset. According to the path lengths in the queries, we formed three query sets: Q_{short}, Q_{medium}, and Q_{long}. We selected the path lengths and the number of queries in each query set as follows: For Q_{short}, Q_{medium}, and Q_{long}, the path length is respectively set to the 1/18, 1/6, and 1/2 of

Table 6.4: Properties of query sets

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

the diameter of the road network. The number of queries in each dataset is picked linearly proportional to the number of junctions. For Q_{short}, Q_{medium}, and Q_{long}, the number of queries is respectively set to the 5/10, 3/10, and 1/10 of the number of junctions in the network. Table 6.4 displays the path length and the number of queries used for each dataset and query set pair. Table 6.4 also displays the number of *GaS* and *GS* operations respectively invoked by the route evaluation and path computation queries for each dataset and query set pair. Although the total number of queries is set to be equal in both query types, *GS* operations constitute 97.7% of all operations in the query workload. This is because of the fact that, for a given source and destination junction pair, the number of *GS* operations in the path computation queries using Dijkstra's algorithm is much larger than the number of *GaS* operations in the route evaluation queries. Here, we should note that the total net costs in the clustering hypergraphs generated for the two storage schemes are exactly equal for a given query set. This enables a fair comparison between the clustering hypergraph models for the two storage schemes.

6.4.2 Storage Size of Junction- and Link-Based Schemes

In the experiments, 4 bytes are reserved for the coordinates of a junction (i.e., $C_{\text{id}} = 4$) and no space is reserved for junction attributes (i.e., $C_{\text{T}} = 0$). We used three different sizes of 16, 28, and 40 bytes for the link attributes (i.e., $C_{\text{L}} = 16$, $C_{\text{L}} = 28$, and $C_{\text{L}} = 40$) in both storage schemes. These attribute sizes, which

Table 6.5: Storage requirements of junction and link-based storage schemes (in bytes)

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

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

are even smaller than the recent proposals [78], are selected to show the actual pattern of performance difference between the two storage schemes. This way, we are able to evaluate the effect of the average record size and total storage size on the relative performance of the two storage schemes. Table 6.5 displays the total storage sizes and the average record sizes for the junction- and link-based storage schemes for each dataset and link attribute size pair. The S_T^b and s_T^b values given in Table 6.5 are exactly the same with those that can be obtained by substituting the network specific parameters in Table 6.1 and the appropriate C_L, C_{id}, and C_T values into (4.9) and (4.12). However, the S_L^b and s_L^b values computed by using (4.10) and (4.13) differ from the values in Table 6.5 because of the simplifying assumption used in these equations. As seen in the Table 6.6, both the actual average and total storage sizes are respectively 9.7% and 9.8% greater than the theoretical values, on the overall average.

As seen in Table 6.5, for $C_L = 16$, the average record sizes are almost equal in the two storage schemes, whereas the link-based scheme requires 29% more total storage than the junction-based scheme, on the average. For $C_L = 28$, the total storage sizes are almost equal in the two storage schemes, whereas the average record size of the link-based scheme is 23% less than that of the junction-based scheme, on the average. For $C_L = 40$, both the total storage size and the average

Table 6.6: Difference between theoretical and actual storage requirements of the link-based storage scheme (in bytes)

		$S_{\mathrm{L}}^{\mathrm{b}}$			$s_{\mathrm{L}}^{\mathrm{b}}$		
C_{L}	Dataset	theoretical	actual	% diff.	theoretical	actual	% diff.
16	D1	748 413	813 624	8.7	52.8	57.4	8.7
	D2	1 153 219	1 298 856	12.6	50.2	56.5	12.5
	D3	2 356 305	2 650 184	12.5	51.1	57.5	12.5
	D4	10 453 890	11 850 952	13.4	49.0	55.5	13.3
28	D1	918 633	983 844	7.1	64.8	69.4	7.1
	D2	1 429 063	1 574 700	10.2	62.2	68.5	10.1
	D3	2 909 541	3 203 420	10.1	63.1	69.5	10.1
	D4	13 014 342	14 411 404	10.7	61.0	67.5	10.7
40	D1	1 088 853	1 154 064	6.0	76.8	81.4	6.0
	D2	1 704 907	1 850 544	8.5	74.2	80.5	8.5
	D3	3 462 777	3 756 656	8.5	75.1	81.5	8.5
	D4	15 574 794	16 971 856	9.0	73.0	79.5	8.9

record size of the link-based scheme are less than those of the junction-based scheme (on the average 13% and 33%, respectively). Although, in general, the link-based scheme requires more storage than the junction-based scheme, the link-based scheme becomes more favorable than the junction-based scheme for $C_{\mathrm{L}} = 40$. This is mainly due to the fact that the proposed way of handling bidirectional links enables higher storage savings in the link-based scheme compared to the junction-based scheme. Note that the link-based storage scheme has a slightly larger average record size than the junction-based storage scheme for D4 with $C_{\mathrm{L}} = 16$. This does not comply with the analytical evaluation given in Section 4.2 because of the underlying assumption on the average record size.

6.4.3 Properties of Hypergraphs

Table 6.7 displays the properties of the clustering hypergraphs used in the experiments for the junction- and link-based storage schemes. In this table, $|n|_{\mathrm{avg}} = |\mathcal{H}|/|\mathcal{N}|$ denotes the average net size of a hypergraph. Since the *GaS* and *GS* operations incurred by the generated queries may not traverse all network elements, the number of nets for each hypergraph is less than the number of all possible nets that can be induced. As mentioned in Section 4.4.3, bidirectional links lead to identical nets in both storage schemes. These nets are detected and eliminated by a preprocessing step. Table 6.7 displays the values after this identical net elimination step.

Table 6.7: Properties of the clustering hypergraphs for the junction- and link-based storage schemes

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

As seen in Table 6.7, $\mathcal{H}_{\text{L}}$ contains considerably more (25.1% on the average) vertices than $\mathcal{H}_{\text{T}}$. Note that the total number of vertices corresponds to the number of records in a storage scheme. In a bidirectional road network, the junction- and link-based storage schemes respectively have $|\mathcal{T}|$ and $|\mathcal{L}|/2$ records, and typically $|\mathcal{T}| < |\mathcal{L}|/2$ since $d_{\text{avg}} > 2$. In terms of the number of nets, $\mathcal{H}_{\text{L}}$ contains fewer (10.5% on the average) nets than $\mathcal{H}_{\text{T}}$. This is mainly due to the junctions with degree one, which do not incur multi-pin nets in $\mathcal{H}_{\text{L}}$. In Table 6.7, the average net size in $\mathcal{H}_{\text{L}}$ is smaller than that of $\mathcal{H}_{\text{T}}$ in accordance with the discussion given in Section 4.4.3 on multi-pin nets.

6.4.4 Partitioning Quality

As in our earlier proposal for the junction-based storage scheme in Section 3.3, we use a recursive bipartitioning scheme to partition $\mathcal{H}_{\text{L}}$ into parts (see Section 3.4). Similar to the results in Section 6.3.3, the RB2 scheme is experimentally found to give slightly better results than the RB1 scheme. The slightly better performance of RB2 in the link-based storage scheme is again due to the fact that it benefits more from page packing as it generates more lightly loaded pages after partitioning. Hence, in our implementation, we adopt the recursive bipartitioning scheme RB2 and page packing approach described in Section 3.4.

Fig. 6.5 displays the partitioning quality of the clustering hypergraph models for the junction- and link-based storage schemes with the link attribute sizes $C_{\text{L}} = 16$ and $C_{\text{L}} = 28$. These experiments are conducted on the hypergraphs

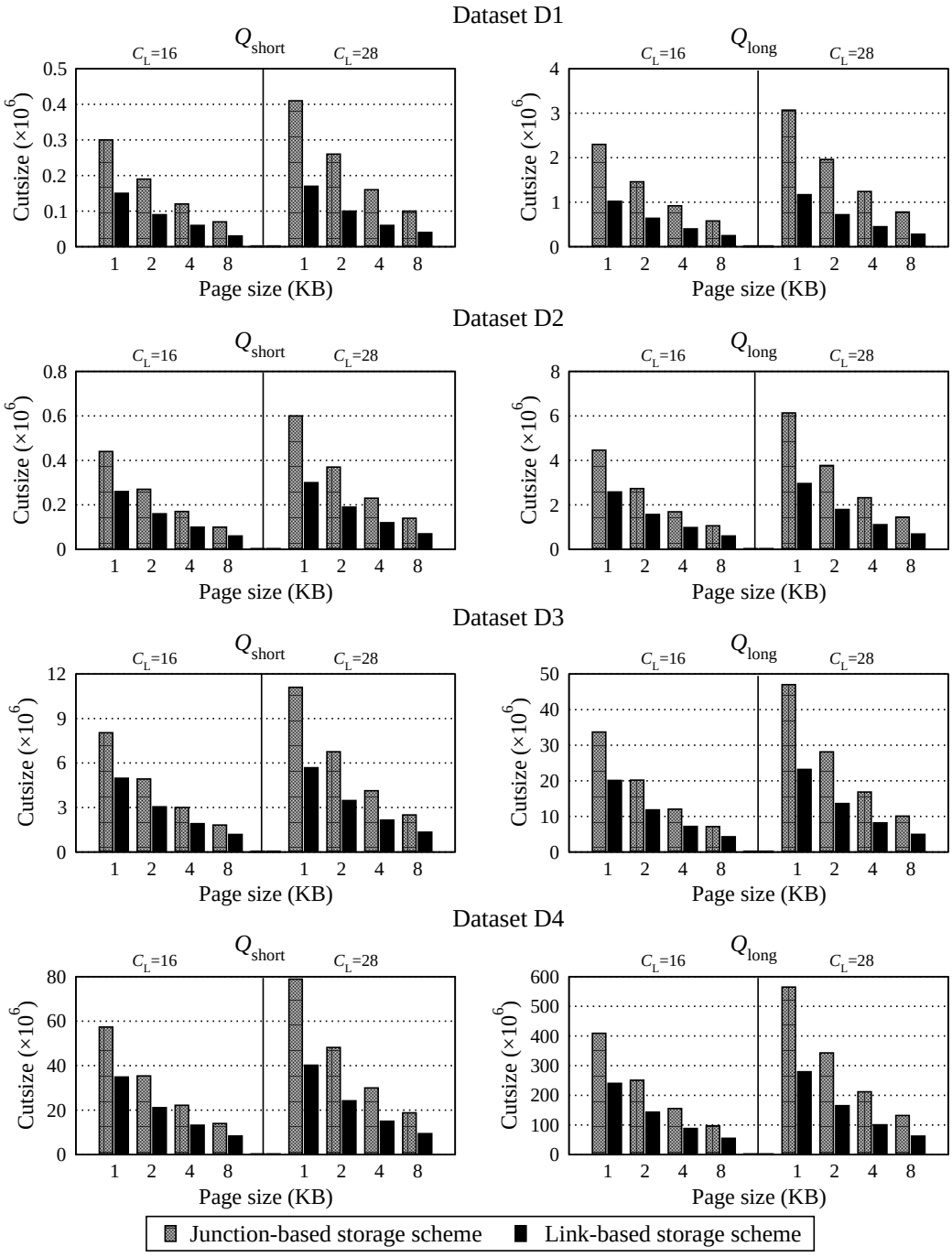


Figure 6.5: Partitioning quality of the clustering hypergraph models for the junction- and link-based storage schemes with $C_T = 0$.

Table 6.8: Averages for percent cutsize improvement of the link-based storage scheme over the junction-based storage scheme

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

generated using the query sets Q_{short} and Q_{long}. As seen in Fig. 6.5, in all cases, the link-based storage scheme achieves smaller cutsize values than the junction-based storage scheme. As expected, the cutsize values decrease with increasing page size in both storage schemes, whereas the performance gap between these two schemes does not vary significantly with varying page size.

Table 6.8 shows the average performance improvements of the clustering hypergraph model for the link-based storage scheme over that for the junction-based storage scheme for all query sets and C_L values. In Table 6.8, P denotes the disk page size. In the table, positive values indicate percent decrease in the K and cutsize values, whereas negative values indicate percent increase in the K values, achieved by the link-based storage scheme compared to the junction-based storage scheme. As seen in Table 6.8, the two storage schemes achieve almost equal K values for the $C_L = 28$ case. The junction-based storage scheme achieves 31.2% smaller K values for the $C_L = 16$ case, whereas the link-based storage scheme results in 12.0% smaller K values for the $C_L = 40$ case, on the average. These percent differences are approximately equal to the percent differences for the total storage sizes reported in Table 6.5.

As seen in Table 6.8, for the $C_L = 28$ case, which incurs almost equal K values for both storage schemes, the link-based storage scheme achieves 53.2% less cutsize values than the junction-based storage scheme, on the average. The relative

performance improvement of the link-based storage scheme over the junction-based storage scheme increases up to 57.9% when the size of the link attributes increases to $C_{\mathrm{L}} = 40$. These experimental findings are in accordance with our expectations discussed in Section 4.4.3. However, it is interesting to note that, for $C_{\mathrm{L}} = 16$, although the link-based storage scheme leads to considerably higher K values, it achieves considerably lower cutsize values (43.9% on the average). This can be attributed to the properties of the clustering hypergraphs modeling the networks with bidirectional links.

The effect of query sets on the relative performance between the two storage schemes is also important. As seen in Table 6.8, for fixed page size and C_{L} values, the performance gap between the two storage schemes increases as the path length increases in favor of the link-based storage scheme. This finding can be attributed to the increase in the number of *GS* operations with increasing path length. As mentioned in Section 4.4.3, the performance difference between the two storage schemes is expected to be higher for *GS* operations compared to the *GaS* operations.

6.4.5 Disk Access Simulations

Figs. 6.6 and 6.7 display the relative performance comparisons of the two storage schemes in terms of the number of disk accesses for both route evaluation and path computation queries. In these figures, B denotes the page buffer size. The simulation results in these figures are presented for the link attribute sizes $C_{\mathrm{L}} = 16$ and $C_{\mathrm{L}} = 28$ with the varying page and buffer sizes. The query sets Q_{short} and Q_{long} are respectively evaluated in Figs. 6.6 and 6.7 to show the effect of path length and number of queries in simulations. The average improvements over all datasets are given in Table 6.9 for all query sets and all C_{L} values.

As seen in Figs. 6.6 and 6.7, the link-based storage scheme outperforms the junction-based storage scheme for almost all simulation cases. In Figs. 6.6 and 6.7, for the $C_{\mathrm{L}} = 16$ case with a single-page buffer, the link-based storage scheme performs better than the junction-based storage scheme in all simulations except for the case of D1 with $P = 8$ and Q_{short}. For the $C_{\mathrm{L}} = 16$ case with larger

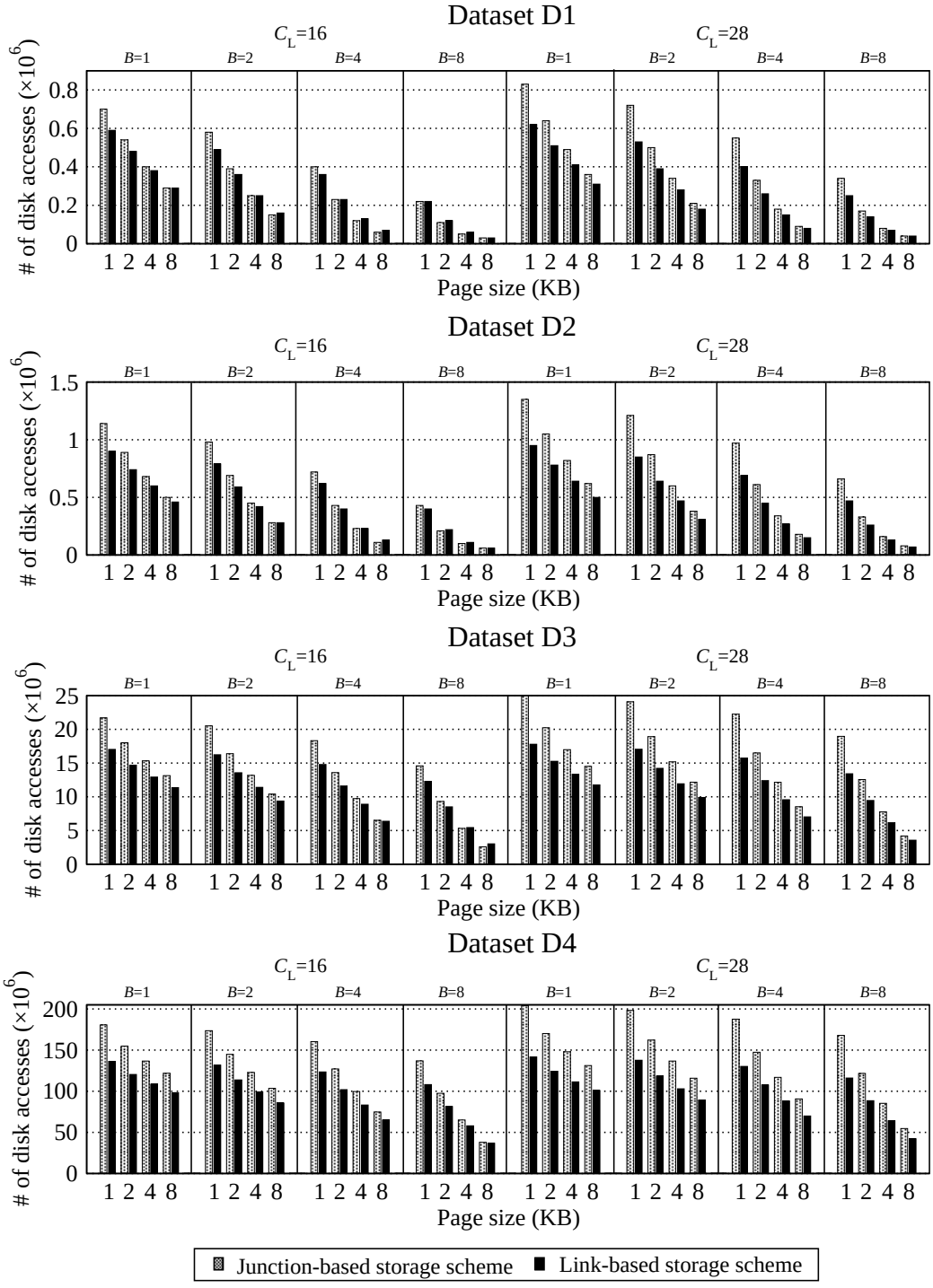


Figure 6.6: Disk access comparisons of the two storage schemes in aggregate network query simulations for the Q_{short} query set and $C_{\text{T}} = 0$.

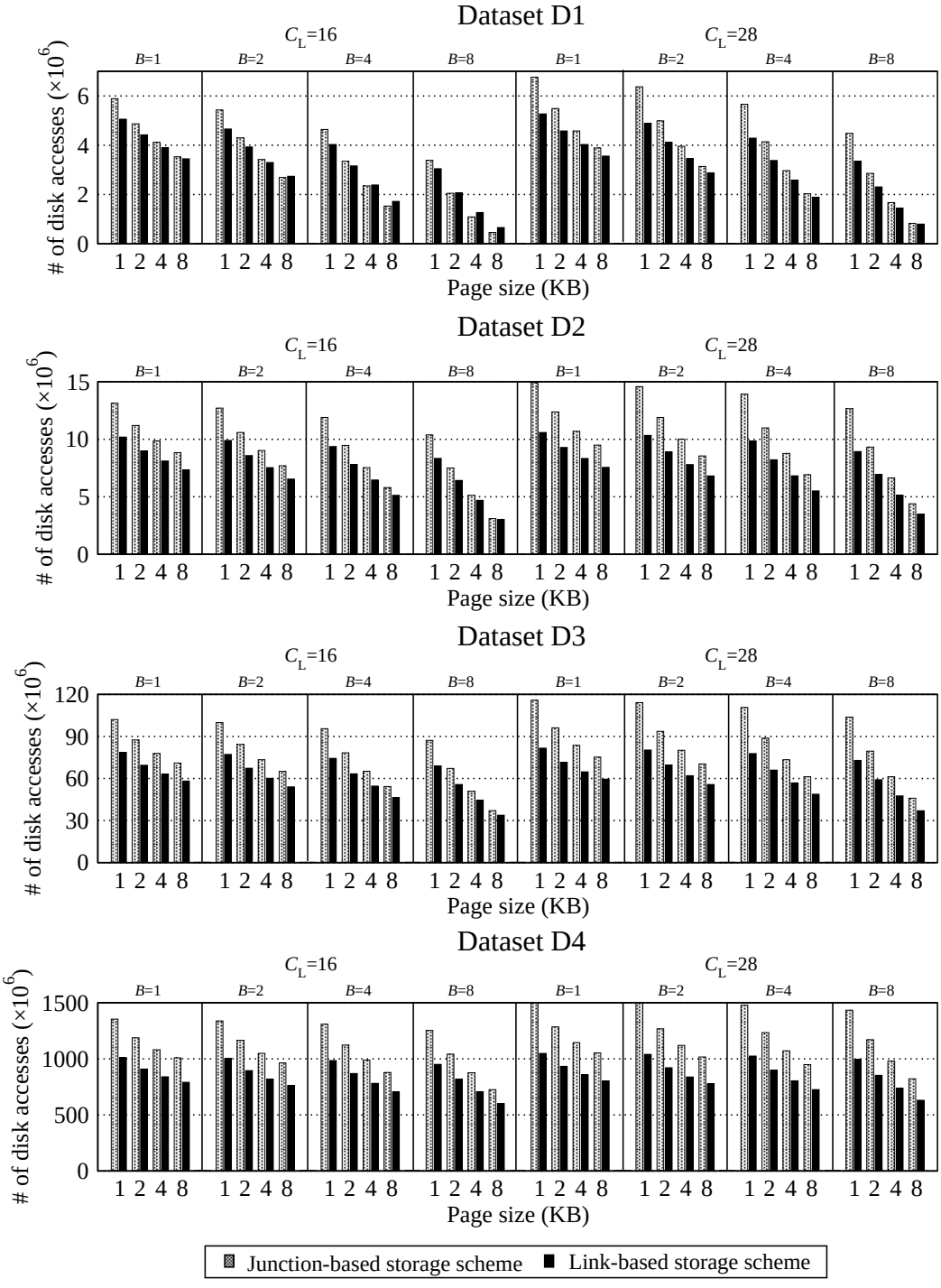


Figure 6.7: Disk access comparisons of the two storage schemes in aggregate network query simulations for the Q_{long} query set and $C_{\text{T}} = 0$.

Table 6.9: Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme in terms of total number of disk accesses for $C_T = 0$

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

page and buffer sizes, especially with short queries, the junction-based storage scheme performs slightly better than the link-based storage scheme. This is due to the fact that average record sizes are almost equal, but the total storage of the link-based storage scheme is 29% larger than that of the junction-based storage scheme.

The comparison of the two storage schemes in Table 6.9 is consistent with the results presented in Table 6.8. However, the final improvements in the simulations are less than the improvements in actual total costs of *GaS* and *GS* operations. As seen in Table 6.8, the average improvement in the total disk access cost of *GaS* and *GS* operations for a single-page buffer is 43.9% and 53.2% for $C_L = 16$ and for $C_L = 28$, respectively. Nevertheless, in Table 6.9, the average improvement in the total disk access cost of aggregate network queries for a single-page buffer is 11.7% and 22.6% for $C_L = 16$ and for $C_L = 28$, respectively. This is mainly due to the additional overhead of *Find* operations incurred by the internal steps of the shortest path algorithm used in path computation queries.

According to Figs. 6.6 and 6.7, as expected, increasing page size and increasing buffer size independently decrease the number of disk accesses in the two storage schemes. The performance gap between the storage schemes decreases

Table 6.10: Averages for percent performance improvement of the link-based storage scheme over the junction-based storage scheme in terms of total number of disk accesses for $C_{\mathrm{L}} = 28$

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

with increasing P. For $C_{\mathrm{L}} = 16$, there are even cases where the junction-based storage scheme performs better than the link-based storage scheme. This experimental finding can be attributed to the total number of records fetched to the memory for query processing and the total storage size difference between the two schemes. As seen from Table 6.4, the length of queries is quite small in D1 and D2 for Q_{short}, and hence the hit rate for records in the buffer increases with increasing page and buffer sizes. On the other hand, for $C_{\mathrm{L}} = 16$, the total number of records and the total storage size of the junction-based scheme is smaller than the link-based scheme and thus the junction-based scheme is expected to perform better with large buffers that can store a considerable portion of the dataset.

Recall that C_{T} and C_{L} are two important factors that affect the average record size and total storage size. The experimental results reported and discussed so far were obtained for a fixed $C_{\mathrm{T}} = 0$ with varying C_{L} values of 16, 28, and 40 bytes. In order to represent a study on varying C_{T}, we also conducted a set of experiments for a fixed $C_{\mathrm{L}} = 28$ with varying C_{T} values of 4, 8, and 16 bytes. The total disk access simulation results of these experiments are displayed in Table 6.10. As seen in the table, the link-based scheme performs better than the junction-based scheme for every combination of simulation parameters except for $B = 8$, $P = 8$, $C_{\mathrm{T}} = 16$ and Q_{short}. The performance gap between the two storage schemes

decreases with increasing C_{T} values, as expected. However, even for the largest C_{T} value of 16 bytes, the link-based storage scheme respectively incurs 14.9%, 17.7%, and 18.7% less disk accesses than the junction-based storage scheme for Q_{short}, Q_{medium}, and Q_{long} query sets, on the average.

6.5 Evaluation of Efficient Successor Retrieval Operations

In this section, we first describe the query generation and properties of constructed hypergraphs. Then, in order to confirm the validity of the proposed successor retrieval operations and associated clustering models, we compare the performance of the proposed *Get-unProcessed-Successors* ($GuPS$) operations modeled by $\mathcal{H}_{\mathrm{GuPS}}$ ($GuPS, \mathcal{H}_{\mathrm{GuPS}}$) and *Get-unVisited-Successors* ($GuVS$) operations modeled by $\mathcal{H}_{\mathrm{GuVS}}$ ($GuVS, \mathcal{H}_{\mathrm{GuVS}}$) against *Get-Successors* (GS) operations modeled by $\mathcal{H}_{\mathrm{GS}}$ ($GS, \mathcal{H}_{\mathrm{GS}}$). In Section 6.5.3, we evaluate 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 6.5.4, we estimate the total number of disk accesses in query processing through simulations.

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. We should also note that, as all links in each dataset are bidirectional, we utilize the storage saving mentioned in Section 3.1 via storing only the successor list of each junction. The total storage size of the network data excluding size of the points of interest and index structures are 1378, 2258, 4510, and 21067 KB in the D1, D2, D3, and D4 datasets, respectively.

6.5.1 Query Generation

For each dataset, we generated three synthetic query sets Q_{short}, Q_{medium}, and Q_{long} depending on the shortest path length of the queries. In order to attain a

Table 6.11: Properties of query sets

		Number of		# of operations that may incur disk access			
	Path	operations				% improvement	
Dataset	length	Queries	*GS/GuPS/GuVS*	*GuPS*	*GuVS*	*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
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
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

high level network coverage, we adaptively determined a different path length and a query count for each dataset and query set (D, Q) pair. Here, 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, 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 6.11 displays the properties of these synthetic query sets.

In Table 6.11, 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 query processing strategies using *GS*, *GuPS*, and *GuVS* operations, we enforced the number of these operations to be the same for each (D, Q) pair. In Table 6.11, 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

Table 6.12: Properties of generated hypergraphs

		Q_{short}			Q_{medium}			Q_{long}		
Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{N}_{\text{GS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$
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
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{N}_{\text{GuPS}}\rvert$	$\lvert\mathcal{H}_{\text{GuPS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuPS}}\rvert$	$\lvert\mathcal{H}_{\text{GuPS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuPS}}\rvert$	$\lvert\mathcal{H}_{\text{GuPS}}\rvert$	$\lvert n\rvert_{\text{avg}}$
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
	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{N}_{\text{GuVS}}\rvert$	$\lvert\mathcal{H}_{\text{GuVS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuVS}}\rvert$	$\lvert\mathcal{H}_{\text{GuVS}}\rvert$	$\lvert n\rvert_{\text{avg}}$	$\lvert\mathcal{N}_{\text{GuVS}}\rvert$	$\lvert\mathcal{H}_{\text{GS}}\rvert$	$\lvert n\rvert_{\text{avg}}$
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

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 the Table 6.11 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.

6.5.2 Properties of Hypergraphs

Table 6.12 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 size of hypergraphs, respectively. Recall that, for a given dataset, the number of vertices of the three clustering hypergraphs $\mathcal{H}_{\text{GS}}$, $\mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$ is the same for all of the three query sets. As mentioned in Section 3.3.1, in $\mathcal{H}_{\text{GS}}$, there exists a single net for each junction on which a *GS* operation is invoked. However, in Table 6.12, for each (D, Q) pair, the number of nets is slightly less than the number of junctions since the network coverage of queries is 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 for $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$, respectively. On the overall, the average number of nets per junction is 2.40 and 2.68 for $\mathcal{H}_{\text{GuPS}}$ and $\mathcal{H}_{\text{GuVS}}$, respectively.

Table 6.13: Number 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]

6.5.3 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 operations. So, K value ranges are reported in Table 6.13 for dataset and page size pairs. As seen in Table 6.13, for each dataset, the number of allocated pages decreases linearly with increasing page size as expected.

Fig. 6.8 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. In all allocation instances, both $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ achieve significantly smaller cutsize values than $(GS, \mathcal{H}_{\text{GS}})$. The cutsize values decrease with increasing page size since the number of records that can be packed in a page increases.

Table 6.14 shows the average cutsize improvement of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ for the Q_{short}, Q_{medium}, and Q_{long} query sets. As seen in Table 6.14, $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ achieve 46.9% and 68.8% improvement in cutsize over $(GS, \mathcal{H}_{\text{GS}})$, on the overall average. For a fixed query set, the performance gaps between $(GS, \mathcal{H}_{\text{GS}})$ 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 $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ as the query set changes from Q_{short} to Q_{long}. This can be explained by the increase in the number of operations in the query sets. A large number of operations is likely to decrease the number of unevaluated successors of individual junctions.

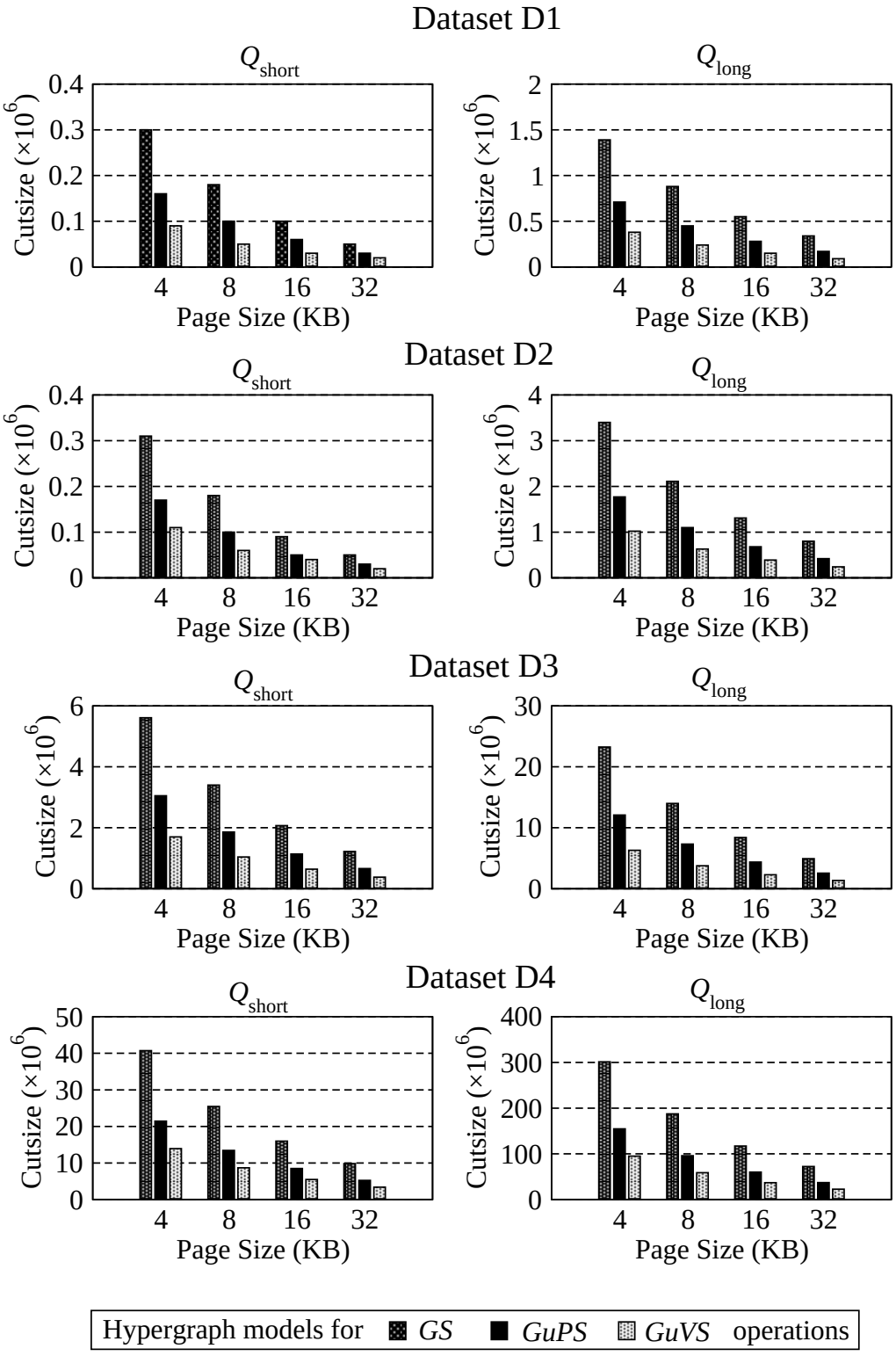


Figure 6.8: Partitioning quality of clustering hypergraph models for *GS*, *GuPS*, and *GuVS* operations. Cutsize is equal to the total number of disk accesses due to the respective successor retrieval operation under the single-page buffer assumption.

Table 6.14: Averages for percent cutsize improvements of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$

	$(GuPS, \mathcal{H}_{\text{GuPS}})$			$(GuVS, \mathcal{H}_{\text{GuVS}})$		
P	Q_{small}	Q_{medium}	Q_{long}	Q_{small}	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

Table 6.15: Averages for percent performance improvements of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ in terms of total number of disk accesses.

		$(GuPS, \mathcal{H}_{\text{GuPS}})$			$(GuVS, \mathcal{H}_{\text{GuVS}})$		
B	P	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

6.5.4 Disk Access Simulations

Figs. 6.9 and 6.10 compare the performance of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ in terms of total number of disk accesses for the Q_{short} and Q_{long} query sets, respectively. The values displayed in Figs. 6.9 and 6.10 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 6.15 shows the average percent performance improvement of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ over all datasets.

As seen in Figs. 6.9 and 6.10, $(GuPS, \mathcal{H}_{\text{GuPS}})$ performs considerably better than $(GS, \mathcal{H}_{\text{GS}})$ in all simulation instances. However, $(GuVS, \mathcal{H}_{\text{GuVS}})$ performs better than $(GS, \mathcal{H}_{\text{GS}})$ 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 $(GuVS, \mathcal{H}_{\text{GuVS}})$

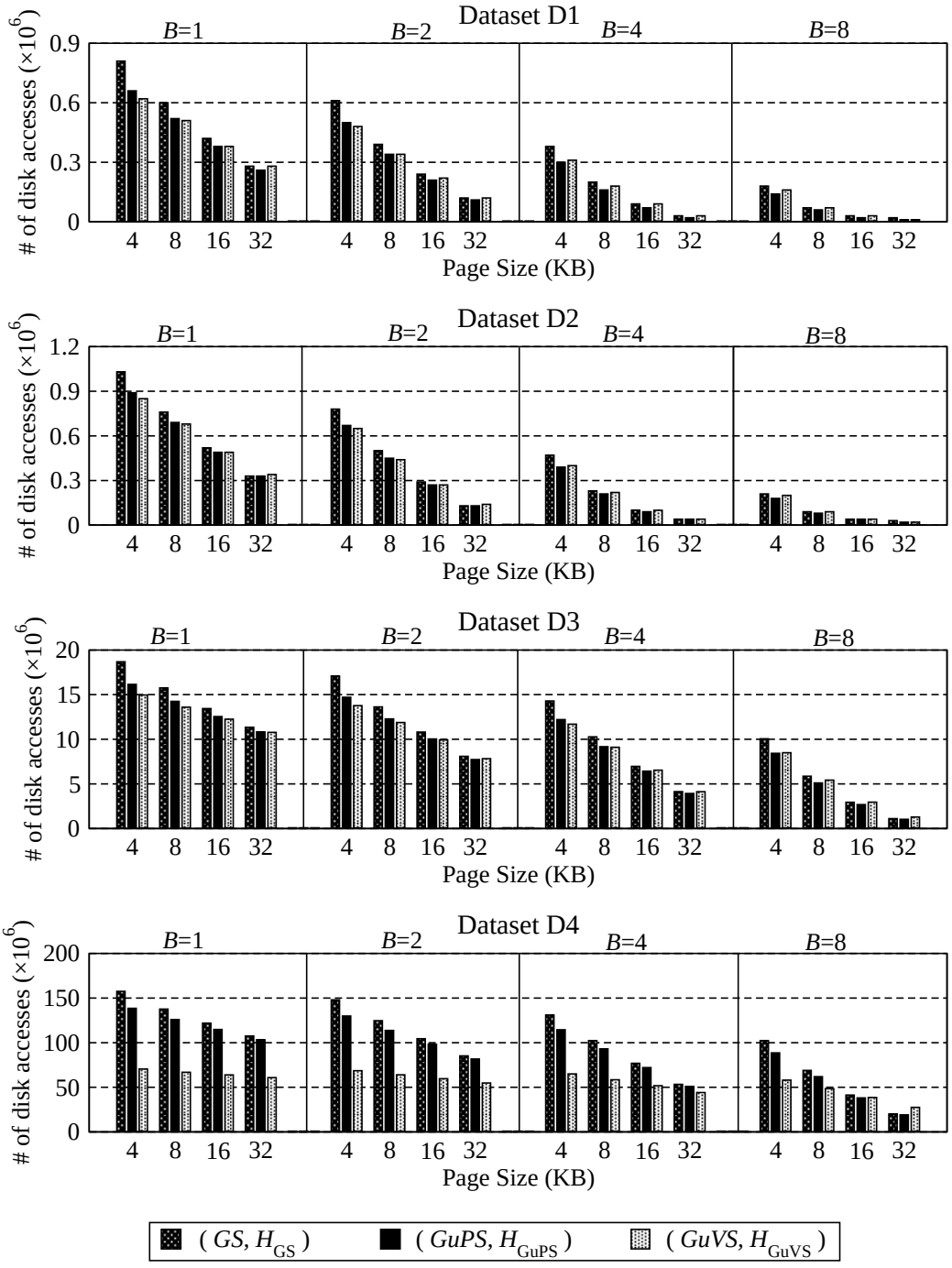


Figure 6.9: Total disk access cost of $(GS, \mathcal{H}_{\text{GS}})$, $(GuPS, \mathcal{H}_{\text{GuPS}})$, and $(GuVS, \mathcal{H}_{\text{GuVS}})$ in query simulations using different page size P in KB and buffer size B in number of pages for Q_{short} query set.

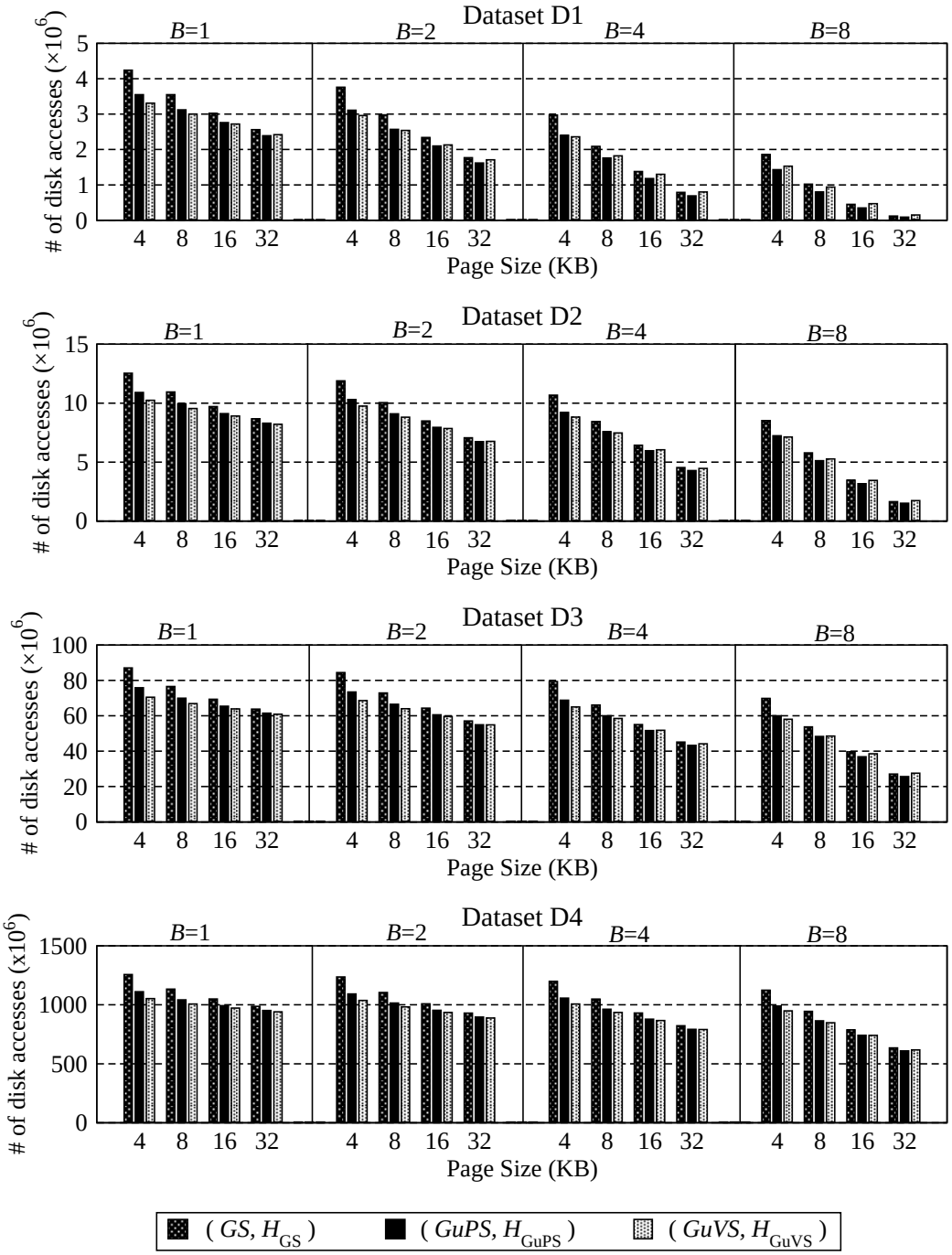


Figure 6.10: Total disk access cost of $(GS, \mathcal{H}_{\text{GS}})$, $(GuPS, \mathcal{H}_{\text{GuPS}})$, and $(GuVS, \mathcal{H}_{\text{GuVS}})$ in query simulations using different page size P in KB and buffer size B in number of pages for Q_{long} query set.

Table 6.16: Averages for percent performance improvements of $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$ in terms of the number of disk accesses incurred only by the successor retrieval operations.

		$(GuPS, \mathcal{H}_{\text{GuPS}})$			$(GuVS, \mathcal{H}_{\text{GuVS}})$		
B	P	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

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 $\mathcal{H}_{\text{GS}}$, $\mathcal{H}_{\text{GuPS}}$, and $\mathcal{H}_{\text{GuVS}}$ respectively capture the exact cost of disk accesses to be incurred by the 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 6.15 also confirm this finding. As seen in Table 6.15, $(GuVS, \mathcal{H}_{\text{GuVS}})$ performs better than $(GS, \mathcal{H}_{\text{GS}})$ in all but 4 out of 48 cases where these 4 disimprovement (negative values) cases occur for large page and buffer size values. Furthermore, comparison of Tables 6.14 and 6.15 shows that average percent performance improvements in simulation results are considerably less than average cutsize improvements. In order to further clarify this issue, we provide Table 6.16 which displays the average percent performance improvements in terms of disk accesses only due to the successor retrieval operations in simulations. Comparison of Tables 6.14 and 6.16 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.

According to Figs. 6.9 and 6.10, as expected, increasing the page size and

increasing the buffer size independently decreases the number of disk accesses in all simulation instances. Comparison of Figs. 6.9 and 6.10 shows 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 6.16, for fixed page and buffer sizes, the performance improvement of both $(GuPS, \mathcal{H}_{\text{GuPS}})$ and $(GuVS, \mathcal{H}_{\text{GuVS}})$ over $(GS, \mathcal{H}_{\text{GS}})$, in terms of the disk access cost due to the successor retrieval operations, slightly increase with increasing query length. However, as seen in Table 6.15, 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.

Chapter 7

Conclusions and Future Work

New proposals for efficient storage and access schemes are inevitable when the massive size of modern spatial databases are considered. In this thesis, we proposed storage and access schemes for efficient query processing on road networks. In particular, we showed that the state-of-the-art clustering graph model for the junction-based storage scheme is not able to correctly capture the disk access costs of successor retrieval operations in query processing on road network. For the junction-based storage scheme, we proposed a novel clustering hypergraph model that utilizes the network usage frequencies obtained from previous query logs and enables the correct estimation of the disk access costs of successor retrieval operations. We introduced two adaptive partitioning schemes based on recursive bipartitioning when the number of parts is not known in advance. Allocation of data to disk pages according to the clustering by partitioning of the proposed hypergraph results in a considerable efficiency improvement in aggregate query processing when compared to the earlier proposals that are based on the clustering graph model.

We introduced the link-based storage scheme where each record stores the data associated with a link together with the link's connectivity information. Our detailed comparative analysis on the properties of the junction- and link-based storage schemes showed that the link-based storage scheme is more amenable to clustering. We proposed a clustering hypergraph model for the link-based storage

scheme to partition the network data into disk pages. We showed that the link-based storage scheme results in better data allocation for processing aggregate network queries.

We introduced a kernel operation *Get-Unevaluated-Successors* (*GUS*) for spatial network queries as a new successor retrieval operation, which is overlooked in the literature. We proposed a clustering hypergraph model that captures the disk access cost of *GUS* operations correctly for the junction-based storage scheme. 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 algorithm, and the latter operation typically arises in the incremental network expansion framework. The results of our experimental simulations 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.

Although this work focused on route evaluation and path computation queries, the developed framework can easily be applied to other types of network queries such as dynamic path computation [78], nearest neighbor [23], range search and closest pairs [59]. Some of these types of queries, such as variants of nearest neighbors and closest pairs, require storing points of interest (POI). The storage of POI is generally handled separately from the network topology, as the updates on POI are frequent when compared to the changes in the network topology. To efficiently handle such queries, we are also conducting research on embedding POI into our storage schemes. The storage schemes mentioned in this work are generic representations of networks, and hence, any index can be built on top of these storage schemes. Application of the link-based storage scheme in graph topologies may also be beneficial for research problems in other fields.

Our work on record-to-page allocation problem for road networks can also be extended in several ways. For example, it is possible to mine the frequently occurring patterns in query logs using existing sequence mining approaches. Frequent sequences can also be stored in consecutive disk pages in order to minimize

the disk access cost of network queries. Mining frequent sequences is considered as a future work.

Bibliography

[1] R. Agrawal and J. Kiernan. An access structure for generalized transitive closure queries. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 429–438, 1993.

[2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: a survey. *VLSI Journal*, 19(1-2):1–81, 1995.

[3] C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, 2008.

[4] C. Aykanat, A. Pinar, and U. V. Çatalyürek. Permuting sparse rectangular matrices into block-diagonal form. *SIAM Journal of Scientific Computing*, 25(6):1860–1879, 2004.

[5] J. Banerjee, S. Kim, W. Kim, and J. Garza. Clustering DAG for CAD databases. *IEEE Transactions on Software Engineering*, 14(11):1684–1699, November 1988.

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331, 1990.

[7] C. Berge. *Graphs and Hypergraphs*. North-Holland Publishing Company, 1973.

[8] T. Brinkhoff. A framework for generating network-based moving objects. *Geoinformatica*, 6(2):153–180, 2002.

[9] T. Brinkhoff. Data files: Oldenburg, San Joaquin. `http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/`, 2007.

[10] T. N. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[11] T. Caldwell. On finding minimum routes in a network with turn penalties. In *Communications of the ACM*, pages 107–108, 1961.

[12] B. B. Cambazoglu, C. Aykanat, and B. Uçar. An identical net detection algorithm for redundant net elimination in hypergraphs. working manuscript.

[13] U. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition of parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

[14] U. V. Çatalyürek and C. Aykanat. PaToH: partitioning tool for hypergraphs. Technical report, Computer Engineering Department, Bilkent University, 1999. `http://www.cs.bilkent.edu.tr/~aykanat/pargrp/patoh/`.

[15] U. S. Census Bureau. Topologically integrated geographic encoding and referencing system (TIGER). `http://www.census.gov/geo/www/tiger/`, 2002.

[16] E. P. F. Chan and H. Lim. Optimization and evaluation of shortest path queries. *The VLDB Journal*, 16(3):343–369, 2007.

[17] E. P. F. Chan and N. Zhang. Finding shortest paths in large network systems. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 160–166, 2001.

[18] C.-K. Cheng and Y.-C. A. Wei. An improved two-way partitioning algorithm with stable performance. *IEEE Transactions on Computer Aided Design*, 10(12):1502–1511, 1991.

[19] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *Proceedings of the International Conference on Very Large Data Bases*, pages 865–876, 2005.

[20] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Data Bases*, pages 426–435, 1997.

[21] S. Dar and H. V. Jagadish. A spanning tree transitive closure algorithm. In *Proceedings of IEEE International Conference on Data Engineering*, pages 2–11, 1992.

[22] A. Dasdan and C. Aykanat. Two novel multiway circuit partitioning algorithms using relaxed locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(2):169–178, 1997.

[23] V. T. de Almeida and R. H. Güting. Using dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases. In *Proceedings of the ACM Symposium on Applied Computing*, pages 58–62, 2006.

[24] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of ACM*, 32(3):505–536, 1985.

[25] E. Demir and C. Aykanat. Efficient successor retrieval operations for aggregate query processing on clustered road networks. *Information Sciences*, under review.

[26] E. Demir, C. Aykanat, and B. B. Cambazoglu. Clustering spatial networks for aggregate query processing: A hypergraph approach. *Information Systems*, 33(1):1–17, 2008.

[27] E. Demir, C. Aykanat, and B. B. Cambazoglu. A link-based storage scheme for efficient aggregate query processing on clustered road networks. *Information Systems*, accepted for publication.

[28] K. Deng, X. Zhau, H. T. Shen, S. Sadiq, and X. Li. Instance optimal query processing in spatial networks. *The VLDB Journal*, 2008. DOI 10.1007/s00778-008-0115-0.

[29] K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 353–356, 2007.

[30] U. S. Department of Transportation - Federal Highway Administration. The national highway planning network. `http://www.fhwa.dot.gov/planning/nhpn/`, 2004.

[31] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[32] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[33] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

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

[35] S. Gupta, S. Kopparty, and C. Ravishankar. Roads, codes, and spatiotemporal queries. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 115–124, 2004.

[36] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceeding of the ACM International Conference on Management of Data*, pages 47–57, 1984.

[37] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[38] H. Hu, D. L. Lee, and V. C. S. Lee. Distance indexing on road networks. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 894–905, 2006.

[39] K. Hua, J. Su, and C. Hua. Efficient evaluation of traversal recursive queries using connectivity index. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 549–558, 1993.

[40] C. Huang, L. Meng, and 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*, pages 209–212, 2002.

[41] X. Huang, C. S. Jensen, and S. Šaltenis. The islands approach to nearest neighbor querying in spatial networks. In *Proceedings of the Advances in Spatial and Temporal Databases*, pages 73–90, 2005.

[42] Y.-W. Huang, N. Jing, and 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*, pages 215–222, 1996.

[43] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Optimizing path query performance: graph clustering strategies. *Transportation Research, Part C*, 8(1):381–408, 2000.

[44] E. Ioup, K. Shaw, J. Sample, and M. Abdelguerfi. Efficient AKNN spatial network queries using the M-Tree. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 1–4, 2007.

[45] S. Jang and J. Yoo. Processing continuous skyline queries in road networks. In *Proceedings of the International Symposium on Computer Science and Its Applications*, pages 353–356, 2008.

[46] C. S. Jensen, J. Kolar, T. B. Pedersen, and I. Timko. Nearest neighbor queries in road networks. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 1–8, 2003.

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

[48] N. Jing, Y.-W. Huang, and E. A. Rundensteiner. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):409–432, 1998.

[49] S. Jung and S. Pramanik. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1029–1046, 2002.

[50] G. Karypis and V. Kumar. hMeTiS: A hypergraph partitioning package version 1.5.3. Technical report, University of Minnesota, Department of Computer Science & Engineering, Army HPC Research Center, Minneapolis, MN 55455, 1998.

[51] G. Karypis and 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/`.

[52] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.

[53] M. Kolahdouzan and C. Shahabi. Voronoi-based K nearest neighbor search for spatial network databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 840–851, 2004.

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

[55] W.-S. Ku, R. Zimmermann, H. Wang, and C.-N. Wan. Adaptive nearest neighbor queries in travel time networks. In *Proceedings of the ACM International Workshop on Geographic Information Systems*, pages 210–219, 2005.

[56] P. A. Larson and V. Deshpande. A file structure supporting traversal recursion. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 243–252, 1989.

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

[58] K. Mouratidis, M. L. Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *Proceedings of the International Conference on Very Large Data Bases*, pages 43–54, 2006.

[59] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 802–813, 2003.

[60] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufmann, 2002.

[61] H. Sagan. *Space-Filling Curves*. Springer-Verlag, Berlin, 1994.

[62] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, 1990.

[63] H. Samet. chapter *Spatial data structures,* in *Modern Database Systems: The Object Model, Interoperability, and Beyond*. Addison-Wesley/ACM Press, 1995.

[64] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2006.

[65] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the ACM International Conference on Management of Data*, pages 43–54, 2008.

[66] J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *Proceedings of the ACM International Workshop on Geographic Information Systems*, pages 200–209, 2005.

[67] J. Sankaranarayanan, H. Alborzi, and H. Samet. Distance join queries on spatial networks. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 211–218, 2006.

[68] J. Sankaranarayanan, H. Alborzi, and H. Samet. Enabling query processing on spatial networks. In *Proceedings of the IEEE International Conference on Data Engineering*, page 163, 2006.

[69] J. Schiller and A. Voisard, editors. *Location-Based Services*. Morgan Kaufmann Publishers Inc., 2004.

[70] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for K-nearest neighbor search in moving object databases. *Geoinformatica*, 7(3):255–273, 2003.

[71] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[72] S. Shekhar and A. Fetterer. Path computation in advanced traveler information systems. In *Proceedings of the 6th Meeting and Exposition of the Intelligent Transportation Society of America*, 1996.

[73] S. Shekhar, A. Kohli, and M. Coyle. Can proximity-based access methods efficiently support network computations. Technical Report TR-93-57, Computer Science Department, University of Minnesota, 1993.

[74] S. Shekhar, A. Kohli, and M. Coyle. Path computation algorithms for advanced traveler information systems. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 31–39, 1993.

[75] S. Shekhar and D. R. Liu. A connectivity-based access method for networks and network computation. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):102–117, 1997.

[76] S. Shekhar and H. Xiong, editors. *Encyclopedia of GIS*. Springer, 2008.

[77] B. Ucar and C. Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. *SIAM Journal of Scientific Computing*, 25(6):1837–1859, 2004.

[78] Q. Wang, L. Xu, and J. Qui. Research and realization of the optimal path algorithm with complex traffic regulations in GIS. In *Proceedings of the IEEE International Conference in Automation and Logistics*, pages 516–520, 2007.

[79] S. Winter. Weighting the path continuation in route planning. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*, pages 173–176, 2001.

[80] S. Winter. Route specifications with a linear dual graph. In *Proceedings of the International Symposium on Advances in Spatial Data Handling*, pages 329–338, 2002.

[81] S.-H. Woo and S.-B. Yang. An improved network clustering method for I/O-efficient query processing. In *Proceedings of the ACM international symposium on Advances in geographic information systems*, pages 62–68, 2000.

[82] M. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse nearest neighbors in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 18(4):540–553, 2006.

[83] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *Proceedings of the ACM International Conference on Management of Data*, pages 443–454, 2004.

[84] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):820–833, 2005.

[85] J. S. Yoo and S. Shekhar. In-route nearest neighbor queries. *Geoinformatica*, 9(2):117–137, 2005.