Master Thesis

216

ΜΠΛΕ

Using Histograms for Building and Querying Workload-Aware Small-Worlds in Peer-to-Peer Systems

Yannis Petrakis

Supervisor: **Evaggelia Pitoura**

Department of Computer Science
University of Ioannina

October 2004

Contents

Contents **i**

1 Introduction **7**

2 Related Work **11**
- 2.1 Resource Discovery in P2P Systems 11
- 2.2 Building Small-Worlds and Clusters 12
- 2.3 Range Queries and Multi-Attribute Histograms 15

3 System Overview **19**
- 3.1 Histograms as Routing Indexes 19
- 3.2 Small-Worlds in Peer-to-Peer Systems 22
- 3.3 Query Routing and Small-World Construction 23
  - 3.3.1 Query Routing 23
  - 3.3.2 Small-World Construction 25
  - 3.3.3 Index Update 26
  - 3.3.4 Load Balancing 28
  - 3.3.5 Node Leave 29

4 Histogram Distance Metrics **31**
- 4.1 Distance Metrics 31
- 4.2 Experimental Evaluation 34
  - 4.2.1 Our Data Distribution 35
  - 4.2.2 Zipf Data Distribution 37
  - 4.2.3 Summary of Experiments 38

5 Multi-Attribute Histograms **41**
- 5.1 Histogram Structure 41
- 5.2 Distance Metric 43
- 5.3 Estimation of Results 43
- 5.4 Using N Simple Histograms Instead of an N-Dimensional Histogram 45
- 5.5 Experimental Results 46

5.5.1 Our Data Distribution 47
5.5.2 Zipf Data Distribution 50
5.5.3 Summary of Experiments 51

6 Experimental Evaluation 55
6.1 Using Simple Histograms 55
6.1.1 Small-World Construction 56
6.1.2 Query Routing 60
6.1.3 Peers Leaving the Network 64
6.1.4 Distribution of Links 64
6.1.5 Summary of Experiments Using Simple Histograms 66
6.2 Multi-Attribute Histograms 66
6.2.1 Small-World Construction 67
6.2.2 Query Routing 70
6.2.3 Summary of Experiments Using Multi-Attribute Histograms 74

7 Conclusions and Future Work 75

List of Figures

1.1 View of a Peer-to-Peer system 9

3.1 Example of an equi-width histogram over an attribute x, $0 \leq x < 40$ 20

3.2 The local indexes of peers 1, 2, 3, and 4 and the routing index of link e of peer 1 for radius $R = 2$, assuming that local indexes $LI(2)$, $LI(3)$ and $LI(4)$ have the same size. 21

3.3 (left) Random and (right) small-world p2p network 23

3.4 Example of an equi-width histogram over an attribute x, $0 \leq x < 40$ 24

3.5 Index update 28

4.1 Intuitively, the distance between $H(n1)$ and $H(n2)$ should be smaller than the distance between $H(n1)$ and $H(n3)$ 32

4.2 Example of two histograms over an attribute $x \in [0,40]$ 33

4.3 (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for our data distribution and for different query ranges ($DT = 0.7$ and $bct = 1$) 36

4.4 (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for our data distribution when varying DT (mixed query workload) 36

4.5 (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for our data distribution ($DT = 0.7$) varying bct (mixed query workload) 36

4.6 (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for zipf data distribution and for different query ranges ($a = 0.9$ and $bct = 1$) 38

4.7 (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for zipf distribution, when varying a (mixed query workload) 38

4.8 (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for zipf distribution ($a = 0.9$) varying bct (mixed query workload) 38

4.9 (left) workload-aware edit distance for our data distribution, and (right) for zipf distribution, when varying DT and a correspondingly (mixed query workload where the starting bucket is not selected uniformly) . . . 39
4.10 (left) workload-aware edit distance for mixed query workload for our data distribution ($DT = 0.7$), and (right) for zipf distribution ($a = 0.9$), when varying bct (mixed query workload where the starting bucket is not selected uniformly) . . . 39

5.1 3-Attribute Histogram . . . 42
5.2 Error in the estimated number of results for multi-attribute and simple histogram . . . 45
5.3 The bold square summarizes the tuples with $x \in [30, 39]$ and $y \in [50, 59]$. . . . 46
5.4 L1 distance for our data distribution for various values of DT (mixed query workload) . . . 47
5.5 L1 distance for our data distribution ($DT = 0.7$) for various values of bct (mixed query workload) . . . 48
5.6 Edit distance for our data distribution for various values of DT (mixed query workload) . . . 48
5.7 Edit distance for our data distribution ($DT = 0.7$) for various values of bct (mixed query workload) . . . 48
5.8 Workload-aware edit distance for our data distribution for various values of DT (mixed query workload) . . . 48
5.9 Workload-aware edit distance for our data distribution ($DT = 0.7$) for various values of bct (mixed query workload) . . . 49
5.10 Workload-aware edit distance for our data distribution for various values of DT (mixed query workload where the starting bucket is not selected uniformly) . . . 49
5.11 Workload-aware edit distance for our data distribution ($DT = 0.7$) for various values of bct . . . 49
5.12 L1 distance for zipf data distribution for various values of a (mixed query workload) . . . 50
5.13 L1 distance for zipf data distribution ($a = 0.9$) for various values of bct (mixed query workload) . . . 50
5.14 Edit distance for zipf data distribution for various values of a (mixed query workload) . . . 51
5.15 Edit distance for zipf data distribution ($a = 0.9$) for various values of bct (mixed query workload) . . . 51
5.16 Workload-aware edit distance for zipf data distribution for various values of a (mixed query workload) . . . 51
5.17 Workload-aware edit distance for zipf data distribution ($a = 0.9$) for various values of bct (mixed query workload) . . . 52
5.18 Workload-aware edit distance for zipf data distribution for various values of a (mixed query workload where the starting bucket is not selected uniformly) . . . 52

5.19 Workload-aware edit distance for zipf data distribution ($a = 0.9$) for various values of *bct* 52

6.1 Influence of radius on diameter when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ 57
6.2 Influence of radius on diameter when *range* = 0 (left) and *range* = 10 (right), when $SL = 1$ 57
6.3 Influence of the number of peers on the diameter when *range* = 0 (left) and *range* = 10 (right) and $SL = 2$ 58
6.4 Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$. 58
6.5 Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 1$. 58
6.6 Clustering quality for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ 59
6.7 Clustering quality for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 1$ 59
6.8 Clustering quality for different number of peers for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$ 60
6.9 Clustering quality for different number of peers for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ 61
6.10 Performance of routing for different values of the radius when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ 61
6.11 Performance of routing for different values of the radius when *range* = 0 (left) and *range* = 10 (right), when $SL = 1$ 62
6.12 Performance of routing for different values of P_l when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$ 62
6.13 PeerRecall when varying the number of peers when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$ 63
6.14 PeerRecall as peers leave the system when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$ 64
6.15 Clustering quality as peers leave the system for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$ and $R = 2$ 64
6.16 Clustering quality as peers leave the system for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$ 65
6.17 Distribution of the links 65
6.18 Influence of radius on diameter when *range* = 0 (left) and *range* = 3 (right), when $SL = 2$ 68
6.19 Influence of radius on diameter when *range* = 0 (left) and *range* = 3 (right), when $SL = 1$ 68
6.20 Influence of the number of peers on the diameter when *range* = 0 (left) and *range* = 3 (right) and $SL = 2$ 68

6.21 Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$. 69
6.22 Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 1$. 69
6.23 Clustering quality for networks built using workload-aware edit distance for *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ 69
6.24 Clustering quality for networks built using workload-aware edit distance for *range* $= 0$ (left) and *range* $= 10$ (right), when $SL = 1$ 70
6.25 Clustering quality for different number of peers for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$ 71
6.26 Clustering quality for different number of peers for networks built using workload-aware edit distance for *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ 71
6.27 Performance of routing for different values of the radius when *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ 72
6.28 Performance of routing for different values of the radius when *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 1$ 72
6.29 Performance of routing for different values of P_l when *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ and $R = 2$ 73
6.30 PeerRecall when varying the number of peers when *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ and $R = 2$ 73

List of Tables

4.1 Input parameters for centralized experiments (Simple Histograms) 35

5.1 Input parameters for centralized experiments (Multi-Attribute Histograms) 47

6.1 Input parameters (Simple Histograms) 56
6.2 Input parameters (Multi-Attribute Histograms) 67

Περίληψη

Τα συστήματα ομοτίμων προσφέρουν έναν αποδοτικό τρόπο για το διαμοιρασμό μεγάλων ποσοτήτων δεδομένων σε ανεξάρτητους κόμβους. Σε αυτήν την εργασσία, θεωρούμε την κατασκευή συστημάτων ομοτίμων με ιδιότητες ενός μικρόκοσμου. Οι μικρόκοσμοι είναι τοπολογίες στις οποίες α) η απόσταση μεταξύ δύο οποιονδήποτε κόμβων είναι μικρή και β) υπάρχει ένας μεγάλος αριθμός συνδέσεων μεταξύ συσχετιζόμενων κόμβων. Οι κόμβοι χαρακτηρίζονται ως συσχετιζόμενοι με βάση την πιθανότητα να απαντούν στο ίδιο σύνολο ερωτήσεων. Προτείνουμε μια μη-κεντρικοποιημένη διαδικασία για την κατασκευή και ενημέρωση δικτύων μικρόκοσμου βασισμένη σε ευρετήρια δρομολόγησης. Τα ευρετήρια δρομολόγησης είναι δομές δεδομένων που περιγράφουν το περιεχόμενο γειτονικών κόμβων. Σαν ευρετήρια δρομολόγησης χρησιμοποιούμε ιστογράμματα, τα οποία είναι κατάλληλα για την απάντηση ερωτήσων διαστήματος τιμών πανω σε ένα γνώρισμα. Σαν ένδειξη συσχέτισης ανάμεσα στο περιεχόμενο των κόμβων αναπτύσσουμε μετρικές σύμφωνα με την ομοιότητα των ιστογραμμάτων, ενήμερες με βάση το φόρτο εργασίας. Εισάγουμε μια μετρική υπολογισμού της απόστασης μεταξύ ιστογραμμάτων, που λαμβάνει υπόψιν το είδος των ερωτήσεων που υποβάλλονται στο σύστημα. Η πειραματική μελέτη έδειξε οτι η μετρική αυτή είναι πολύ αποδοτική για κάθε είδος ερωτήσεων. Επεκτείνουμε τα ιστογράμματα και τις μετρικές, εισάγοντας μια δομή για πολυεπίπεδα ιστογράμματα πάνω σε περισσότερα από ένα γνωρίσματα. Τέλος, τα πειραματικά αποτελέσματα δείχνουν οτι σε σ̆υτήματα ομοτίμων μικροκόσμου που κατασκευάζονται χρησιμοποιώντας τη διαδικασία που προτείνουμε, το πλήθος των συσχετιζόμενων αποτελεσμάτων που επιστρέφονται αυξάνεται.

Acknowledgments

I would like to thank my supervisor Professor Evaggelia Pitoura for her help, support, the time spent during the elaboration of this thesis, and especially, the patience she has shown until this thesis is completed. I would like also to thank Georgia Koloniari for her useful contribution on many issues of this work, and especially on the study of the histogram distance metrics.

Abstract

Peer-to-peer systems offer an efficient means for sharing large amounts of data among autonomous nodes. In this thesis, we consider building peer-to-peer systems with small-world properties. Small-worlds are networks in which (i) the distance between any two nodes is small and (ii) there is a large number of connections among relevant nodes. We characterize relevance between nodes based on the probability that the nodes match the same set of queries. We propose a decentralized procedure for constructing and updating small-worlds based on routing indexes. Routing indexes are data structures that describe the content of neighboring nodes. We use histograms as routing indexes, and as an indication of relevance among the content of nodes we develop workload-aware metrics of histogram similarity. Our experimental results show that in small-world peer-to-peer systems built using our procedure, the percentage of relevant results returned is increased.

Chapter 1

Introduction

The popularity of file sharing systems such as Napster [3], Gnutella [1] and Kazaa [2] has spurred much current attention to peer-to-peer (p2p) computing. Peer-to-peer computing refers to a form of distributed computing that involves a large number of autonomous computing nodes (the peers) that cooperate to share resources and services [18].

The term "peer-to-peer" (p2p) refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner. With the pervasive deployment of computers, p2p is increasingly receiving attention in research, product development, and investment circles. Some of the benefits of a p2p approach include: improving scalability by avoiding dependency on centralized points; eliminating the need for costly infrastructure by enabling direct communication among clients; and enabling resource aggregation.

The resources encompass computing power, data (storage and content), network bandwidth, and presence (computers, human, and other resources). The critical function can be distributed computing; data/content sharing, communication and collaboration, or platform services. Decentralization may apply to algorithms, data, and meta-data, or to all of them.

P2P enables valuable externalities, by aggregating resources through low-cost interoperability, lower cost of ownership and cost sharing, by using existing infrastructure and by eliminating and distributing the maintenance costs and anonymity/privacy, by incorporating these requirements in the design and algorithms of p2p systems and applications, and by allowing peers a greater degree of autonomous control over their data and resources.

In a p2p system, peers are autonomous. They depend on each other for getting information, computing resources, forwarding requests, etc. which are essential for the functioning of the system as a whole and for the benefit of all peers. As a result of the autonomy of peers, they cannot necessarily trust each other and rely completely on the behavior of other peers, so issues of scale and redundancy become much more important than in traditional centralized or distributed systems.

Each peer in a p2p system is connected with only a small number of other peers (neighbors). Whenever a message need to be transferred between two peers that are not neighbors, the message should be propagated through all the intermediate peers.

Conceptually, p2p computing is an alternative to the centralized and client-server models of computing, where there is typically a single or small cluster of servers and many clients. In its purest form, the p2p model has no concept of server; rather all participants are peers (Fig. 1.1).

In our view, p2p is about sharing: giving to and obtaining from the peer community. A peer gives some resources and obtains other resources in return. In the case of Napster, it was about offering music to the rest of the community and getting other music in return. p2p is also a way of implementing systems based on the notion of increasing the decentralization of systems, applications, or simply algorithms. p2p is a way to leverage vast amounts of computing power, storage, and connectivity from personal computers distributed around the world.

Our goal is to build a peer-to-peer data sharing system. Consider a database that consists of one or more relations. Each peer stores a database with the same schema. The user poses in the network a range query over one or more attributes of the database, asking for the tuples (stored at varius peers) that satisfy the query. We want to increase the number of results (tuples) returned by propagating the query to peers that have large number of tuples that satisfy this query. Thus, a central issue in p2p systems is identifying which peers contain data relevant to a user query; we call such peers *matching* peers.

In this thesis, we propose building small-worlds based on the content of the peers. Small-worlds are networks with (i) a small distance between any two peers (small diameter) and (ii) a large number of connections among relevant peers (large clustering coefficient). The small-world phenomenon has many applications in real life [29, 13]. Friendship networks are a good example of this. Consider the friendship graph, where each peer corresponds to a person and two people are connected with an edge if they know each other. Such a graph consists of smaller sub-graphs (which are rich in short-range connections) each one representing a community and there are a few long-range links between peers of different communities (if A knows B and B knows C, then A is more likely to know C than some other random person). We define the relevance of two nodes (peers) based on the common queries that both of them match. Intuitively, the topology of a small-world network represents a number of smaller networks (groups or clusters) that are rich in links between their peers (short-range links), while they are linked to each other with a few random connections (long-range links). The motivation for such small-world p2p networks is that once in the appropriate group, all relevant to a query peers are a few links apart. Long-range links are used for routing among groups.

Previous approaches to the problem of building small-worlds, either study the problem theoretically, or they do not present a clear description of the construction procedure. Also, there are many works that cluster peers together based on their context similarity, but in most of them the number of clusters

Each peer in a p2p system is connected with only a small number of other peers (neighbors). Whenever a message need to be transferred between two peers that are not neighbors, the message should be propagated through all the intermediate peers.

Conceptually, p2p computing is an alternative to the centralized and client-server models of computing, where there is typically a single or small cluster of servers and many clients. In its purest form, the p2p model has no concept of server; rather all participants are peers (Fig. 1.1).

In our view, p2p is about sharing: giving to and obtaining from the peer community. A peer gives some resources and obtains other resources in return. In the case of Napster, it was about offering music to the rest of the community and getting other music in return. p2p is also a way of implementing systems based on the notion of increasing the decentralization of systems, applications, or simply algorithms. p2p is a way to leverage vast amounts of computing power, storage, and connectivity from personal computers distributed around the world.

Our goal is to build a peer-to-peer data sharing system. Consider a database that consists of one or more relations. Each peer stores a database with the same schema. The user poses in the network a range query over one or more attributes of the database, asking for the tuples (stored at varius peers) that satisfy the query. We want to increase the number of results (tuples) returned by propagating the query to peers that have large number of tuples that satisfy this query. Thus, a central issue in p2p systems is identifying which peers contain data relevant to a user query; we call such peers *matching* peers.

In this thesis, we propose building small-worlds based on the content of the peers. Small-worlds are networks with (i) a small distance between any two peers (small diameter) and (ii) a large number of connections among relevant peers (large clustering coefficient). The small-world phenomenon has many applications in real life [29, 13]. Friendship networks are a good example of this. Consider the friendship graph, where each peer corresponds to a person and two people are connected with an edge if they know each other. Such a graph consists of smaller sub-graphs (which are rich in short-range connections) each one representing a community and there are a few long-range links between peers of different communities (if A knows B and B knows C, then A is more likely to know C than some other random person). We define the relevance of two nodes (peers) based on the common queries that both of them match. Intuitively, the topology of a small-world network represents a number of smaller networks (groups or clusters) that are rich in links between their peers (short-range links), while they are linked to each other with a few random connections (long-range links). The motivation for such small-world p2p networks is that once in the appropriate group, all relevant to a query peers are a few links apart. Long-range links are used for routing among groups.

Previous approaches to the problem of building small-worlds, either study the problem theoretically, or they do not present a clear description of the construction procedure. Also, there are many works that cluster peers together based on their context similarity, but in most of them the number of clusters

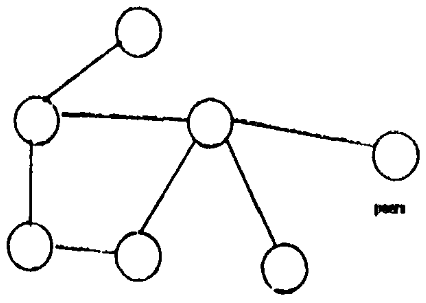

Figure 1.1: View of a Peer-to-Peer system

is fixed and global knowledge of the information the peers store is required to create the clusters.

We present an approach for building small-worlds based on a fully decentralized procedure. Our approach is based on using local indexes. A local index is a characterization of the content of a peer. Routing indexes that are created by aggregating local indexes of neighboring peers, are used to create small-worlds in a fully distributed manner, where only local information is used. We define similarity among indexes, so that peers with similar indexes match the same set of queries. Such similarity is *weighted*, so that matching frequent queries counts "more" than matching infrequent ones, thus creating workload-aware small-worlds.

We propose using histograms [14] as local indexes and show how such indexes can be used to construct a small-world and route range queries over one attribute. The main advantages of histograms over other techniques are that they incur almost no run-time overhead and, for most real-world databases, there exist histograms that produce low-error estimates while occupying reasonably small space. We also introduce a weighted histogram distance metric that takes into account the query workload.

Our approach is extended to support range queries over multiple attributes. Previous approaches to the problem propose multi-dimensional histograms with the drawback that they cannot be easily updated and aggregated. To this end, we propose a multi-attribute histogram structure, that takes into account the dependencies between the attribute values, and also, it can be updated and aggregated easily. The weighted histogram distance metric is extended for multi-attribute histograms.

In summary, this thesis makes the following contributions:

- presents a procedure for building content-based small-worlds in p2p systems based on routing indexes, that is, on data structures that summarize information about the content of neighboring peers;

- proposes building *workload-aware* small-worlds so that the grouping of similar peers is based on whether the peers match similar sets of queries; this is achieved by incorporating the frequency of queries in the definition of similarity between the indexes of the peers;

- exploits histograms as routing indexes for query processing and small-world construction in p2p systems and introduces appropriate workload-aware histogram distances;
- proposes a structure for multi-attribute histograms for supporting range queries over multiple attributes, and extends the workload-aware edit distance for this type of histograms.

Our experimental results show that our small-world construction procedure is effective, since in the resulting peer-to-peer system peers with small histogram distance have also small network distance. Also, routing is very efficient, in particular, for a given number of visited peers, since using histograms increases a lot the number of results returned for a given number of peers visited compared with a random network. Also, the network scales very well when increasing the number of peers, since only $logM$ number of peers need to be visited (during the join procedure) as the number M of peers increases, in order to achieve the same performance and to leave the network properties unaffected. Furthermore, the use of the workload-aware edit distance improves the performance comparing to the other distance metrics.

Thesis Outline

In chapter 2 the related work is presented. In chapter 3 we introduce histograms as routing indexes and describe how histograms are used to route, construct and maintain small-world. In chapter 4, we propose a novel workload-aware distance metric and present experiments comparing it with other distance metrics. In chapter 5, we introduce multi-attribute histograms, distance metrics for this type of histograms and experimental results comparing the different distance metrics. In chapter 6, we present our experimental results showing the performance of our small-world network. Finally, in chapter 7, we offer conclusions and directions for future work.

Chapter 2

Related Work

We propose an approach for building and querying small-world networks based on the content of the peers using routing indexes. Routing Indexes are stored for each link of a peer and summarize the content of a number of peers that are reached through this link. We consider range queries over one or more attributes and propose a multi-dimensional histogram structure that is used as a routing index when more that one attributes of a relation are summarized. We distinguish the research related to our work into three areas:
a) research on resource discovery in p2p systems, b) research on building small world networks or organizing peers in clusters and c) research on building networks for answering range queries and methods for summarizing the content of relations over more than one attribute.

2.1 Resource Discovery in P2P Systems

There are two basic types of p2p systems: ***structured*** and ***unstructured*** ones. In structured p2p systems, data items (or indexes) are placed at specific peers usually based on distributed hashing (DHTs).

Chord [12] is an an example of a structured peer-to-peer system. It is a distributed lookup protocol for the efficient location of the node that stores a desired data item. Given a key Chord maps a key onto a node. A key is associated with each data item and the key/data pair is stored at the node to which the key maps. The nodes are mapped into a virtual cycle according to the identifier provided by the hash function and data are stored to the nodes based on their own hash identifier.

A similar approach is followed in CAN [24]. CAN organizes the nodes in a d-dimensional coordinate space, which is dynamically partitioned among all the nodes, such that each node owns an individual distinct zone within the overall space. Keys are deterministically mapped onto points in the coordinate space using a uniform hash function. To retrieve an entry corresponding to a key, any node can apply the same hash function to map the key onto the point and then retrieve the data from the node corresponds to the point. CAN provides fast lookup functionality on Internet like scales.

Although these approaches provide very efficient search, they compromise peers autonomy. Also, the DHT topology is regulated since all peers have the same number of neighboring peers.

In unstructured p2p systems, there is no assumption about the placement of data items at the peers. Napster [3] is one of the first peer-to-peer systems, that relies on a centralized server which accepts the requests from all the peers. Gnutella [1] is another unstructured p2p system where a flooding algorithm (requests are are forwarded to all the neighbors) is used for query routing. However, both of these solutions introduce scalability problems. In Napster the centralized server becomes overloaded as the number of peers increases (increasing also the number of requests). In Gnutella, there are many messages in the network, thus larger network bandwidth is needed. Also, many irrelevant peers receive many requests increasing there load.

In order to achieve efficient location of the relevant data for a given query, many solutions have been proposed that use summaries describing the content of a set of other peers in the network. These summaries are used to route the query to the appropriate region in the network.

In [9] routing indexes are used for efficient routing of the queries. The routing indexes are stored for each outgoing link of a node and used for selecting the best neighbor to route the query. The routing index for each link is produced by summarizing the content of the nodes along that path. Several types of routing indexes are discussed. The simple routing index summarizes the content of all the nodes along the path. Hop-count routing indexes are aggregated routing indexes for each hop up to a maximum number of hops. Finally, in exponentially aggregated routing indexes, it is not stored the exact number of results for each network distance as in hop-count routing indexes, but a value is stored that depends on the network-distance the results can be found. They require less storage space with the drawback of loss in accuracy.

A similar approach is presented in [19]. Bloom filters are used as indexes. Each peer maintains a local Bloom Filter that represents the object in the local repository, and a remote Bloom Filter for each link obtained from its neighbors. During the query routing, each node propagates the query to the best k links based on the semantic similarity with the query. When a node discovers that a peer frequently produces good results to its requests, it attempts to move closer to it by connecting directly to that peer.

2.2 Building Small-Worlds and Clusters

Many recent research focus on organizing peers into a small-world topology that has been shown to be efficient in retrieving large number of results.

In [15] a small world network is proposed for which it can be shown that the number of messages for search is bounded by a function proportional to $(logN)^2$. The network structure is a two-dimensional lattice where each node is connected with the neighbor nodes in the lattice through short-range connections. Also each node has a long-range connection with a non-neighbor node. This long-range connection is created with probability $1/r^a$, where r is the lattice distance and a a parameter. For $a = 2$, it is shown that the above search bound is achieved. However a very strict topology it presumed. Also, there is no description of the procedure that creates this type of network.

In [13], peers are organized into clusters based on their interests, where peers within the same cluster are highly connected between them. Each peer has all the information provided by the cluster. Gossiping mechanisms are used for the information dissemination within a group. Requests that cannot be answered by the local node are forwarded to other clusters. However, there is no clear description of the procedure followed to build such a network and to navigate between the clusters. Also each peer should have knowledge about all the data the peers of its cluster store.

Small worlds in non DHT p2p systems are also discussed in [5] in the context of searchable querical data networks. Interconnection between nodes is correlated with 'similarity' of the data content of the nodes. Some principles of building and querying networks based on the content of the peers are discussed. Nodes with similar data content are clustered together. A query is propagated to the neighbor that is more similar to the target data. Some variations of the query propagation are proposed, such as the use of flooding when the first target data item is located. However, this work does not include a concrete decentralized small-world construction procedure.

In [4], a family of distributed access methods are proposed for building a small-world network and for efficient execution of various similarity-search queries (exact-match, range and k-nearest neighbor queries). The idea is that all the similar objects are nearby in the network. L_p norm is used as the distance function between two nodes keys that represent their data. The forward primitive uses only local information to process a query and make forward decisions. The query is propagated to the neighbor that has the smaller L_p distance with the query key. Range or kNN queries can be executed efficiently in two phases: first, by an exact-match query to locate the locality of the query, and then by flooding the query throughout the locality of the query.

In SSW (Semantic Small World)[16] peers are organized with semantically similar data closer to each other in clusters, forming a small world overlay network. LSI (Latent Semantic Indexing) is employed to derive the semantic vector for each data object. A peer clusters its data object with similar semantics. The location information of the peer's local data objects not belonging to the semantic subspace of the peer are published into the rest of the P2P system. A peer also holds location information of data objects belonging to its subspace but physically stored at other peers. A new peer joins the cluster whose centroid is the closest to its semantic label. Since the dimension of the semantic vector is correlated with the SSW dimensionality, only the first few dimensions are used for the construction of the small world topology, which results in loss of information.

Very recently, researchers have proposed extending DHTs with long range links towards creating small worlds. Symphony [17] is an extension of Chord. Similar to Chord, the participants are arranged along a ring. Each node that arrives chooses an id uniformly from an interval. A node manages the sub-range of the interval which corresponds to the segment on the cycle between its own id and the id of its immediate clockwise predecessor. There are also long distance links drawn from a family of harmonic distributions, in order to achieve a small-world topology. Each node except of its neighbors maintains a list of its neighbor's neighbors in order to improve the choice of neighbor for routing queries.

Also, many recent research efforts are focusing on organizing peers in clusters, which in a sense are similar to groups in small worlds. In most cases, the number or the description of the clusters is fixed and global knowledge of this information is required. The main difference between clusters and small-worlds is that a small-world refers to a particular topology, where peers in the same group are highly connected between them and there are a few links between peers of different groups. On the other hand, a cluster is a set of similar peers (based on some characteristics) where there is no assumption on how peers within the cluster are connected. Also, there is no assumption on how different clusters are connected between them.

In SETS [6], peers are partitioned into a fixed set of C clusters each one corresponding to a topic segment such that sites with similar documents belong to the same segment. Each topic segment has a description called topic centroid. Knowledge of the centroids is global (all the peers should know the centroids of all the clusters). Short distance links connect sites within a segment. Long distance links connect sites of different segments. The routing procedure consist of two steps: Global routing forwards the query to the appropriate segment. Then, local routing is used to propagate the query to a subset of sites within a segment. The disadvantage of this approach is that there must be a site for specific administrative tasks, such as updating the topic centroids and finding the appropriate topic segment for attaching a new peer.

In Semantic Overlay Networks (SONs) [8], nodes semantically related are clustered together forming a semantic overlay network. There is a classification hierarchy of the queries and the nodes based on their content. Queries are routed to the appropriate SONs, increasing the chances that matching files will be found quickly. However, there is no description on how SONs are created or how the queries are routed.

In [28], documents are classified into semantic categories. Clusters of peers are formed based on the semantic categories of the documents they store. The number of clusters is fixed and predefined during the bootstrap of the system. Each node in a cluster has information about the content of all the other peers within the cluster, in order to serve all requests for documents contributed by all the nodes of the cluster. Alternatively, this can be achieved by having a distinct set of super-peers storing cluster metadata. Also each node stores a Document Category Routing Table that maps documents categories to cluster-ids and a Node Routing Table mapping each cluster-id to a list of nodes belonging to the cluster. These tables are used for the routing of the queries to the right cluster.

Recent extensions of the DHT-based networks propose instead of associating keys to data items based just on their identifier, to associate with each data item (or peer) a vector describing its content and then use this vector as an input to the hashing functions.

In self-organizing SONs [27], LSI is used to represent documents and queries as vectors in a Cartesian space. Semantically related indices are in nearby nodes of the overlay network. The document semantics are produced using LSI. CAN is used to create a semantic overlay by using the semantic vector of the document as the key to store document index in the CAN. The query is only need to be compared against a small region centered at the query. Since the dimensionality of the CAN is much smaller than the LSI's semantic space, rolling index is used to partition the semantic space along more dimensions by rotating the semantic vectors. Also a content-aware boot-strapping helps to distribute the indices more evenly across nodes. Finally, a content-directed search algorithm is used to face the problem of the curse of dimensionality. However, there is loss of information when reducing the dimensions (using rolling indexes).

In [26] a p2p information discovery system is presented, that supports flexible queries using partial keywords and wildcards, and range queries. The key innovation is a dimension reducing scheme that effectively maps the multi-dimensional information space to physical peers. Documents that are local in the multi-dimensional space are mapped to indices that are local in the 1-dimension space and to peers that are close in the overlay network. Chord is used as the overlay network topology. Each data elements is associated with a sequence of keywords that form a multi-dimensional space. A Space-Filling Curve (SFC) is used for the mapping from the d-dimensional to the 1-dimensional space. The problem with this mapping is that not all the adjacent sub cubes in the d-dimensional space are adjacent or even close in the curve.

2.3 Range Queries and Multi-Attribute Histograms

In [25] a method based on the CAN system for efficient evaluating range queries is proposed. The answers of range queries are cached at the peers and are used to answer future range queries. The system uses a $2d$ virtual hash space similar to CAN, which is partitioned into zones. Each zone is assigned to a peer. Given a range query (q_s, q_e), it is hashed into point (q_s, q_e) in the virtual hash space (target point) and thus, the node that owns this zone stores information about the answer of the query. Not all the peers in the system participate in the partitioning. The nodes that participate are called active nodes (each one owns a zone). The rest of the peers are called passive nodes. Initially there is only one active node that owns the entire hash space. The partitioning of the hash space is dynamic and changes over time as the existing zones split (due to large load) and new zones are assigned to passive nodes. Whenever a peer gets an answer, it informs the target node in order to cache the answer. Initially, all the queries are answered by the single active node. The performance of the network increases as more queries are posed.

In [21] is presented how multi-dimensional range queries can be supported in a p2p system by using traditional spatial-database technologies (*kd*-trees and space-filling curves). When space-filling curves are used for range partitioning, data is first mapped down into a single dimension using a space-filling curve and then, the single-dimensional data is range partitioned across a dynamic set of nodes. The network graph topology is a circular linked list of nodes, enhanced with additional skip pointers for faster routing. Each node manages data in a contiguous range of values. Similarly, during the query routing, the multi-dimensional query is first converted to an appropriate set of 1-d range queries. All these queries are routed to the network. When using *kd*-trees, the data space is broken into rectangles with each node managing one rectangle. The leaf nodes of the *kd*-tree correspond to the rectangle being stored by a node. The space is split load equally instead of space equally. Skip pointers are also used to speed up the routing.

STHoles [20] is a 'workload-aware' histogram that allows bucket nesting to capture data regions. This histogram exploits query workload to spend more resources in heavily accessed areas. Regions that are more heavily queried will benefit from having more buckets with finer granularity. The problem with *kd*-tree and STHoles is that it is very difficult to merge histograms that belong to one or another type, since the tree structure of these two histograms is not determined and depends on the number of tuples (for *kd*-tree) and on the query workload (for STHoles).

In [23], two main alternatives to approximate multi-dimensional joint data distributions are proposed: multi-dimensional histograms and Singular Value Decomposition (SVD) techniques. Following the first approach, either the problem is reduced to a single dimension using space-filling curves, such as the Hilbert curve (however, two adjacent points in the n-dimensional space may be distant in the linear ordering), or the n-dimensional space is partitioned into rectangular regions (at every step the attribute for which the marginal distribution is the most in need of partitioning is selected to be partitioned). SVD techniques cannot be extended to dimensions greater than two.

To conclude, there are many works that deal with the small-world phenomenon in p2p systems. There are works that study the small-world phenomenon theoretically, such as [15], or propose p2p network topologies that follow the small-world topology [13, 5], but there is no clear decentralized construction procedure. The small-world topology has also been extended to DHT systems (such us [17]), with the drawbacks of compromising peers autonomy and requirement for strict network topology. Clustering of the peers have been proposed by many recent works [6, 8, 28]. The number or the description of the cluster is fixed in most cases, and global knowledge of this information is needed to attach each peer to the appropriate cluster.

We use histograms as routing indexes, instead of other type of indexes used in previous works, such as inverted list of keywords [9] and Bloom Filters [19]. The main advantages of histograms over other type of indexes are that they incur almost no run-time overhead and, for most real-world databases, there exist histograms that produce low-error estimates while occupying reasonably small space [14]. Histograms also perform better on answering range queries and there are type of histograms that can be easily updated and aggregated (creating the routing indexes).

There are also many works that deal with the problem of answering multi-dimensional range queries [20, 21, 23]. These works propose either techniques for reducing the problem into a single dimension (that lacks accuracy since two adjacent points in the n-dimensional space may be distant in the 1-dimensional ordering) or structures for multi-dimensional histograms. In the latter case, the structures proposed have large update cost, since the multi-dimensional space is split into regions of different size which is modified as the data changes, and also, it is difficult to be merged.

Chapter 3

System Overview

We assume a p2p system with a set of peers. This set changes as new peers leave and join the system. Each peer is connected to a small number of other peers called its *neighbors*. Peers store data items. A query q may be posed at any of the peers, while data items satisfying the query may be stored at various peers of the system. Our goal is to route the query to such matching peers efficiently.

3.1 Histograms as Routing Indexes

We focus on p2p systems where each peer stores a relation R with a numeric attribute x and on routing range selection queries on x. Each peer maintains an index of the data items stored locally at it; this is called a *local index*. As local indexes, we use histograms. Histograms are widely used as a mechanism for compression and approximation of data distributions, for selectivity estimation, approximate query answering and load balancing [14]. In this work, we use histograms for clustering and query routing in p2p systems.

A histogram $H(x)$ on an attribute x is constructed by partitioning the data distribution of x into b (≥ 1) mutually disjoint subsets called *buckets* and approximating the frequencies and values in each bucket. The problem of building an accurate histogram for a given attribute is well-studied and is not the focus of this thesis. One requirement in our context is using histograms that can be efficiently aggregated; that is given two histograms for the data items stored locally at two peers, to have a low-cost procedure for constructing one histogram for the items collectively stored at both peers. We consider *equi-width* histograms, that is, we divide the value set of an attribute x into ranges of equal width (buckets). For each bucket we keep the fraction of the number of tuples with values for x within the range of the bucket, divided by the total number of tuples the histogram summarizes. Consider the histogram shown in Fig. 3.1 over an attribute x of a relation R, $0 \leq x < 40$. The data space of x is partitioned into four equi-width buckets. The fraction of the total number of tuples that belong to the first bucket (the value of x is included within range [0,9]) is 0.3, which

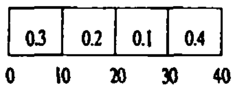

Figure 3.1: Example of an equi-width histogram over an attribute x, $0 \leq x < 40$

means that 30% of the tuples have value for x within range [0,9]. Similarly, the fraction of the total number of tuples that belong to the second bucket (the value of x is included within range [10,19]) is 0.2, and so on. In addition, we maintain the total number of all tuples (the histogram *size*), denoted by $S(n)$.

We denote by $LI(n)$ the histogram used as the local index of peer n. Besides its local index, each peer n maintains one routing index $RI(n, e)$ for each of its links e, that summarizes the content of all peers that are reachable from n using link e at a distance at most R, called ***radius***. The routing index $RI(n, e)$ is also a histogram defined next.

We shall use the notation $H(n)$ to denote a histogram for peer n (used either as a local index $LI(n)$ or as a routing index $RI(n, e)$), $H_i(n)$ to denote the frequency of the values within i-th bucket, $0 \leq i \leq b-1$, and $S(H(n))$ to denote its size. Then,

Definition 1 (histogram-based routing index) *The frequency of the values within the $i-th$ bucket of the histogram-based routing index $RI(n, e)$ of radius R of the link e of peer n is defined as:*
$RI_i(n, e) = \Sigma_{p \in P}(LI_i(p) * S(LI(p)))/\Sigma_{p \in P}S(LI(p))$,
where $S(RI(n, e)) = \Sigma_{p \in P}S(LI(p))$ and P is the set of peers p within distance R of n reachable through link e.

Thus, the frequency stored at each bucket in the routing index of a particular link e of peer n is produced by taking the fraction of the total number of tuples that correspond to this bucket for all the local indexes within the horizon (reachable through this link), divided by the total number of tuples the peers within the horizon (that are reachable through this link) store.

Similarly, if the local index of peer p need to be subtracted from the routing index of peer n, the new routing index is produced by the following equation:
$RI_i(n, e) = (RI_i(n, e) * S(RI(n, e)) - LI_i(p) * S(LI(p)))/(S(RI(n, e)) - S(LI(p)))$

An example is shown in Fig. 3.2. For a given query q, the local histogram $LI(1)$ of peer 1 provides an estimation of the number of results (matching tuples) of peer 1, while the routing index $RI(1, e)$ provides an estimation of the number of results that can be found when the query is routed through link e. The set of peers within distance R of peer 1 is called the ***horizon*** of radius R of peer 1.

We denote by $results(n, q)$ the actual number of results peer n stores for query q and by $hresults(H(n), q)$ the number of results estimated by the histogram $H(n)$ that summarizes the content of peer n. As usual, we make the ***uniform frequency*** assumption and approximate all frequencies in a bucket by their average. We also make the ***continuous values*** assumption, where all possible values in the domain of x that lie in the range of the bucket are assumed to

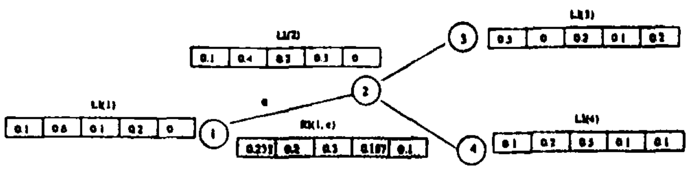

Figure 3.2: The local indexes of peers 1, 2, 3, and 4 and the routing index of link e of peer 1 for radius $R = 2$, assuming that local indexes $LI(2)$, $LI(3)$ and $LI(4)$ have the same size.

be present. However, there is a probability that although a value is indicated as present by the histogram, it does not really exist in the data (false positive). This is shown to depend on the number of buckets, the number of tuples and the range of the attribute. Details can be found in the Appendix.

Let a query $q_{kj} = \{x\colon a \leq x \leq a + k * d - 1\}$, where d is equal to the range of each bucket. We denote k ($0 \leq k < b$) as the query range (the number of buckets the query includes), j is the starting bucket of the query, $j{=}a/d$ and $a = c * d$ (the start of the query is the start of bucket j), where $0 \leq c \leq b - 1$. We also consider the queries $q_{<j} = \{x\colon x \leq a\}$ and $q_{>j} = \{x\colon x \geq a\}$. Note that query $q_{>j}$ is the same with query q_{bj}. Also if $j + k > b - 1$ for the query q_{kj}, then it is the same with the query $q_{>j}$. We can estimate *results*(n, q) using the histogram $H(n)$ of peer n based on the type of the query q as follows:

- q_{kj}: $hresults(H(n), q_{kj}) = S(H(n)) * \Sigma_{i=j}^{j+k} H_i(n)$
- $q_{<j}$: $hresults(H(n), q_{<j}) = S(H(n)) * \Sigma_{i=0}^{j} H_i(n)$
- $q_{>j}$: $hresults(H(n), q_{>j}) = S(H(n)) * \Sigma_{i=j}^{b} H_i(n)$

We defined the query q_{kj} as starting from the lower limit of a bucket ($a = c$ $*$ d), for simplicity.

In the general case where the start or the end of the query can be any point of the bucket the query can be defined as $q_{kj} = \{x\colon a_1 \leq x \leq a_2 + k * d - 1\}$ where $\lfloor j = a_1/d \rfloor$, a_1 and a_2 can be any integer values within the range $[0, d * b - 1]$ and $a_1 \leq a_2 + k * d$. Also $q_{>j} = \{x\colon x \geq a_1\}$ and $q_{<j} = \{x\colon x \leq a_2\}$, where $\lfloor j = a_1/d \rfloor$ and $\lfloor j = a_2/d \rfloor$ correspondingly. In order to estimate the number of results we use the uniform frequency assumption for the values of each bucket. The fraction of the total number of tuples that satisfy each type of query is:

- q_{kj}: $perc(H(n), q_{kj}) = \Sigma_{i=j+1}^{j+k-1} H_i(n) + H_j(n) * (1 - (a_1 - j * d))/d + H_{j+k}(n) * (a_2 - (j + k + 1) * d)/d$ (1)
- $q_{<j}$: $perc(H(n), q_{<j}) = \Sigma_{i=0}^{j-1} H_i(n) + H_j(n) * (a_1 - j * d)/d$ (2)
- $q_{>j}$: $perc(H(n), q_{>j}) = \Sigma_{i=j+1}^{b} H_i(n) + H_j(n) * (1 - (a_2 - j * d))/d$ (3)

The estimated number of results based on the type of query is:

- q_{kj}: $hresults(H(n), q_{kj}) = S(H(n)) * perc(H(n), q_{kj})$ (4)
- $q_{<j}$: $hresults(H(n), q_{<j}) = S(H(n)) * perc(H(n), q_{<j})$ (5)
- $q_{>j}$: $hresults(H(n), q_{>j}) = S(H(n)) * perc(H(n), q_{>j})$ (6)

In the rest of the thesis, we assume that queries start and end at the lower and the upper limits of the buckets correspondingly.

3.2 Small-Worlds in Peer-to-Peer Systems

Ideally, we would like to route each query q only through the peers that have the largest number of results for q. To this end, we define *PeerRecall* as our performance measure. *PeerRecall* expresses how far from the optimal a routing protocol performs. Let V be a set of peers ($V \subseteq N$), by $Sresults(V, q)$ we denote the sum of the numbers of results returned for a query q by each peer that belongs to V.

$$Sresults(V, q) = \Sigma_{v \in V} results(v, q) \tag{3.1}$$

Definition 2 (PeerRecall) *Let Visited ($Visited \subseteq N$) be the set of peers visited during the routing of a query q and Optimal ($Optimal \subseteq N$) be the set of peers such that $|Optimal| = |Visited|$ and $v \in Optimal \Leftrightarrow results(v, q) \geq results(u, q), \forall\, u \notin Optimal$. We define PeerRecall as: $PeerRecall(Visited, q) = Sresults(Visited, q)/Sresults(Optimal, q)$.*

Intuitively, to increase *PeerRecall*, peers that match similar queries must be linked to each other. The network *distance* between two peers n_i and n_j, $dist(n_i, n_j)$ is the length of the shortest path from n_i to n_j in the p2p network. The *diameter* of the network is the maximum distance between any two peers in the network. The *clustering coefficient* of a network captures the probability that two neighbors of a peer are also neighbors themselves; it is the average fraction of pairs of neighbors of a peer that are neighbors of each other. Small-world networks are characterized by: (i) a small diameter and (ii) a large clustering coefficient [29]. To increase PeerRecall, small-worlds of peers should be formed based on whether the peers match similar queries. Fig. 3.3 shows a random and a small-world p2p network. In a small-world network, peers that match a query are nearby, thus once in the right group all matching peers are nearby.

To cluster peers, we propose using their local indexes. That is, we cluster peers that have similar local histograms. For this to work, the distance (d) between two histograms must be descriptive of the difference in the number of results to any given query.

Property 1 *Let $LI(n_1)$, $LI(n_2)$ and $LI(n_3)$ be the local indexes of three peers n_1, n_2 and n_3. For each query q, if $d(LI(n_1), LI(n_2)) \geq d(LI(n_1), LI(n_3))$, then $|results(n_1, q)/S(LI(n_1))$ - $results(n_2, q)/S(LI(n_2))| \geq$ $|results(n_1, q)/S(LI(n_1))$ - $results(n_3, q)/S(LI(n_3))|$.*

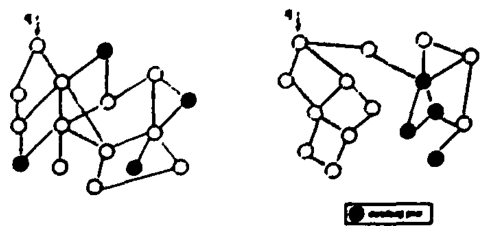

Figure 3.3: (left) Random and (right) small-world p2p network

If the distance between the local indexes $LI(n_1)$ and $LI(n_2)$ of peers n_1 and n_2 is smaller than the distance between the local indexes $LI(n_1)$ and $LI(n_3)$ of peers n_1 and n_3, we want also the difference in the percentage of tuples that satisfy the query for n_1 and n_2 to be greater than the difference in the percentage of tuples that satisfy the query for n_1 and n_3.

We expect peers that have small histogram distance, to have also small difference in the number of results they return for each given query. Since our goal is to maximize the returned number of results, we want to build the network such as the structure to help on retrieving large number of results. Peers that are more likely to answer the same set of queries (and thus, have small distance between their local indexes) we want to be nearby in the overlay network.

We consider latter two distance metrics (the L_1 and the edit distance) and propose weighted versions of both that satisfy the above property, taking into account the query workload. For different query workload the distance between the two histograms will be different, since the difference in the number of results varies depending on the query.

3.3 Query Routing and Small-World Construction

We describe first how histogram-based routing indexes can be used to route a query and then how small-worlds are constructed. We distinguish between two types of links: *short-range* or short links that connect similar peers and *long-range* or long links that connect peers with non-similar content. Two peers belong to the same *group* if and only if there is a path consisting only of short links between them.

3.3.1 Query Routing

A query q may be posed at any peer n. Our goal is to route the query q through a set of peers that gives a large number of results for q, that is, we want to maximize *PeerRecall*. To this end, we use the following heuristic: each peer that receives a query propagates it through those of its links whose routing indexes indicate that the peers reachable through them provide the largest number of matching tuples. In particular, each peer n that receives

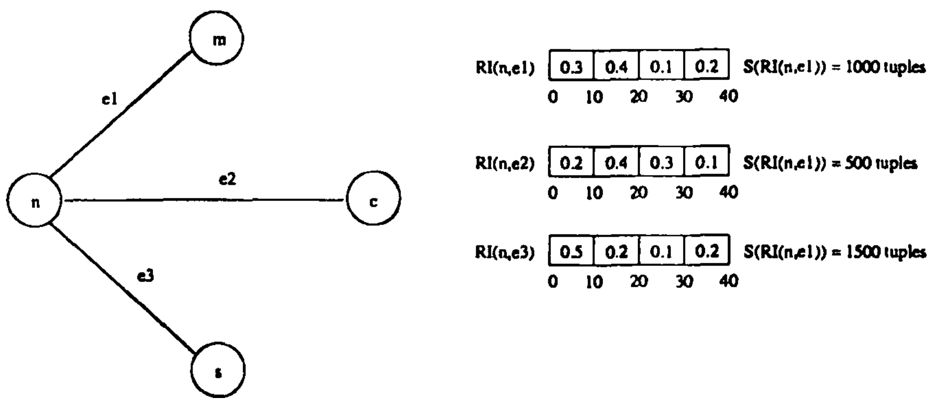
m
e1
n
e2
c
e3
s
RI(n,e1) 0.3 0.4 0.1 0.2 S(RI(n,e1)) = 1000 tuples
0 10 20 30 40
RI(n,e2) 0.2 0.4 0.3 0.1 S(RI(n,e1)) = 500 tuples
0 10 20 30 40
RI(n,e3) 0.5 0.2 0.1 0.2 S(RI(n,e1)) = 1500 tuples
0 10 20 30 40

Figure 3.4: Example of an equi-width histogram over an attribute x, $0 \leq x < 40$

a query q propagates the query through the link e whose routing index gives the most matches ($hresults(RI(n,e),q) > hresults(RI(n,l),q)$ $\forall$ link $l \neq e$) and has not been followed yet. By following this link, the query is propagated towards the peers that are estimated to provide the largest number of results and thus *PeerRecall* is increased. The routing of a query stops either when a predefined number of peers is visited (*MaxVisited*) or when a satisfactory number of results is located.

Consider the situation in which the query q reaches peer n of Fig. 3.4. Let the query be $10 \leq x < 15$. The estimated number of results for each of the routing indexes of links e1, e2 and e3 is (based on equation (4) in Section 3.1):
$hresults(RI(n,\mathrm{e}1),q) = RI_1(n,\mathrm{e}1) * S(RI(n,\mathrm{e}1)) * perc(RI(n,\mathrm{e}1),q) = 0.4 * 1000 * 0.5 = 200$
$hresults(RI(n,\mathrm{e}2),q) = RI_1(n,\mathrm{e}2) * S(RI(n,\mathrm{e}2)) * perc(RI(n,\mathrm{e}2),q) = 0.4 * 500 * 0.5 = 100$
$hresults(RI(n,\mathrm{e}3),q) = RI_1(n,\mathrm{e}3) * S(RI(n,\mathrm{e}3)) * perc(RI(n,\mathrm{e}3),q) = 0.2 * 1500 * 0.5 = 150$
Thus, peer n selects link e1 to propagate the query since the estimated number of results is the largest.

By following this procedure, it is possible to reach a situation in which no matching peers are found. This can happen if the peer n that poses a query has no matching links ($hresults(RI(n,e),\ q) = 0$ $\forall$ link e of n), which means that the matching peers (if any) are outside the radius R of n. In this case, query routing would stop without returning any results. To handle this case, we use a variation of the above procedure until we find the first matching link. Specifically, if no matching link has been found during the routing of the query, and the current peer has no matching links, then the long-range link of this peer is followed (even if it does not match the query). The idea is that we want to move to another small-world of the network, since the current small-world (bounded by the horizon) has no matching peers. In the case that the peer has no long-range link or we have already followed all long-range links, the query is propagated through a short link to a neighbor peer and so on until a long-range link is found.

In the above procedure each peer propagates the query to only one of its neighbors (corresponding to a depth-first traverse of the network). A variation can be used in order to exploit the grouping of the peers for faster retrieval of the results. More specifically, when the query reaches a peer p of the appropriate group, flooding can be used, in order to propagate the query to all the peers that are connected with p through short links (thus, propagate the query to all the neighbors of the same group).

The question is how we can determine whether the query has reached appropriate group or not. Let the query q_{kj}, where k is the range of the query and j the starting bucket. If $hresults(H(n), q_{kj}) = max_i(hresults(H(n), q_{ki}))$ $\forall\, 0 \leq i \leq b-1$, then peer n belongs to the appropriate group. The idea is that if the r buckets that include the query range correspond to the combination of k buckets of the local index that gives the maximum number of results for this particular query, then all the peers of the same group will have large number of results to these buckets and it will be effective to flood the query to this group.

This variation of the query procedure can be effective for fast retrieval of the results. However it does not ensure that many results will be found. The problem is that each new peer that joins the system links to the most similar of the existing peers. This does not guarantee that peers linked through short links can answer the same set of queries. Also, it is possible that a query (for example a query with large range) can be answered by peers of more than one groups. However, when using flooding, after the flooding starts, the propagation of the query is limited within one group only.

3.3.2 Small-World Construction

We describe next how routing indexes can be used to build small-world networks. Each new peer that enters the system tries to find a relevant group of peers and links with SL of them through short links. The idea is to use the local index of the new peer as a query and route this towards the peers that have the most similar indexes. Then the new peer is connected with the SL peers that are most similar to it. Each new peer also links with a peer that does not belong to the group through a long link with probability P_l. Short links are inserted so that the peers with relevant information are located nearby in the p2p network and a large clustering coefficient is attained. Long links are used for keeping the diameter small. The reason is that we want to be able to find both all relevant results once in the right group, and the relevant group once in another group, thus increasing *PeerRecall*.

In particular, when a new peer p wishes to join the system, a join message that contains its local index $LI(p)$ is posed as a query to a well known peer in the system. The join message also maintains a list L (initially empty) with all peers visited during the routing of the join message. The join message is propagated until up to $JMaxVisited$ peers are visited.

Whenever the join message reaches a peer n:

1. The distance $d(LI(n), LI(p))$ between local indexes $LI(n)$ and $LI(p)$ is calculated.

2. Peer n and the corresponding distance are added to list L.

3. If the maximum number of visited peers $JMaxVisited$ is reached, the routing of the join message stops.

4. Else, the distances $d(LI(p), RI(n, e))$ between the local index $LI(p)$ of the new peer p and the routing indexes $RI(n, e)$ that correspond to each of the outgoing links e of peer n are calculated.

5. The message is propagated through the link e with the smallest distance that has not been followed yet, because there is a higher probability to find the relevant group through this link. When the message reaches a peer with no other links that have not been followed, backtracking is used.

Consider the example of Fig. 3.4. Let the join message be at peer n and the appropriate link need to be selected to propagate the message. Let the histogram distance between the local index of the new peer p and the routing indexes for each link of n be:
$d(LI(p), RI(n, e1)) = 10$
$d(LI(p), RI(n, e2)) = 5$
$d(LI(p), RI(n, e3)) = 15$
Then the join message will be propagated through link $e2$, since the routing index of peer n for this link has the smallest distance from the local index of the new peer p.

When routing stops, the new peer selects to be linked through short links to the SL peers of the list L whose local indexes have the smaller distances from the local index of the new peer, and with probability P_l to one of the rest of the peers in the list through a long link.

An issue is how the peer that will be attached to the new peer through the long link is selected. One simple approach is to select randomly one of the rest of the peers within the list (that does not belong to the SL peers selected to be linked through short links). Other more sophisticated methods can be followed for selecting the long link based on the histogram distances, but this is left as future work.

3.3.3 Index Update

When a new peer n_k joins the system, it must inform the peers within distance R about the data it stores, in order to update their routing indexes. To this end, n_k sends a message $New(LI(n_k), Counter)$ to all its neighbors, where $Counter$ is set to R. Upon receipt of a New message, each peer n_i merges the received $LI(n_k)$ index with the routing index of the corresponding link. Then, it reduces $Counter$ by one, and if $Counter$ is nonzero, it sends a $New(LI(n_k), Counter-1)$ message to all other of its neighbors. This way, the local index of the new peer is propagated to the existing peers.

In addition, the new peer must construct its own routing indexes. Thus, it should receive the local indexes of all the peers within its horizon for each of its links and construct the routing indexes for each link based on the Def. 1 of Section 3.1. This is achieved through a sequence of $FW(Index, Counter, flag)$ messages. In particular, each peer n_i upon receipt of a New message from a peer n_j, it replies to n_j with a $FW(LI(n_i), R, False)$ message where $Counter$ is set to R. The use of the $flag$ parameter will be explained shortly. Upon receipt of a $FW(LI(n_i), Counter, False)$ message, each peer n_j, decrements the $Counter$ by one, and if $Counter$ is nonzero, it sends a $FW(LI(n_i), Counter - 1, False)$ message back to the peer that has sent the New message to it. This way, the local indexes reach the new peer n_k. Peer n_k creates its routing indexes by merging the corresponding local indexes received by the various FW messages.

As data changes locally at a peer, its local index need to be updated. Also, the routing indexes of all the peers within its horizon need update. Thus, periodically each peer informs its local index about the changes in the data stores locally. Next, each peer that updates its local index, should inform all the peers within its horizon to update their routing indexes. This is done by a sequence of New messages.

We now explain the use of the $flag$ parameter. $Flag$ is used because the insertion of a new peer may change further the horizons of existing peers. Take for example the network of Figure 3.5 with $R = 2$. Say a new peer, peer 13, enters the network and links to both peers 1 and 3. The local index of 13 must be propagated to 1, 2 and 3, 4; this is achieved through the New messages. Peer 13 must also construct its own routing indexes; this is achieved through the FW messages with $flag$ equal to $False$. However, note that the insertion of 13 has changed the relative distance of some peers. In particular, now peer 3 (1) belongs to the horizon of 1 (3) since their distance (through the new peer 13) is now 2. Thus, the local index of 3 (1) must now be merged with the corresponding index of 1 (3).

$Flag$ is used as follows. $Flag$ is initially set to $False$. When the new peer n_k receives a $FW(Index, Counter, False)$ message, it changes $Flag$ to $True$, decrements $Counter$ by one, and if $Counter$ is nonzero, it propagates a message $FW(Index, Counter - 1, True)$ to all of its other neighbors. Upon receipt of a $FW(Index, Counter, True)$ message, each peer merges the $Index$ with its corresponding routing index, decrements $Counter$ by one, and if $Counter$ is nonzero, it sends a $FW(Index, Counter - 1, True)$ message to its neighbors. This way, indexes of peers whose horizons change by the introduction of the new peer are propagated to each other. When a peer wishes to leave the system, it sends an update message to all its neighbors with a counter set to the radius R. When the message reaches a peer, the peer performs the update at its routing index and propagates the message further until the counter reaches 0. Furthermore, it sends it own local index through the same link with a counter set to R to inform the peers that are now included in its horizon, since the departure of the peer has resulted in the decrease of its distance with other peers.

Note that, as indicated in Figure 3.5, it is possible that the local index of

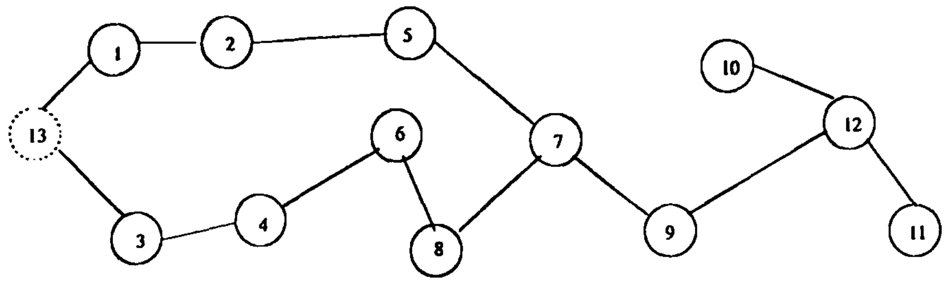

Figure 3.5: Index update

a peer n_i is included in more than one routing indexes of some other peer n_j. However, we may want to avoid this, because during search, two different paths will lead us to the same peer. This problem can be overcome by using peer identifiers. Each peer stores the identifiers of the peers that are included in each of its routing indexes. When a local index reaches a peer during the join procedure, the peer first checks whether it has already stored this index at the routing index of some other link. The problem is that it cannot be applied in the situation that the peer sends the *New* message in parallel to all its neighbor. Thus, the *New* messages should be sent sequentially if we are interested in avoiding a situation that more than one routing indexes to include the same local index.

3.3.4 Load Balancing

By following the above procedure, peers that stay longer in the p2p network are connected with more short links than recently arrived ones, since the probability of old peers being selected as neighbors of arriving peers increases as they stay in the network. This is not necessarily undesirable, since peers that stay connected for longer periods of time are more stable. On the other hand, peers with more links will receive more requests and thus, it is possible that they are overloaded. This can be remedied by following a simple reconstruction procedure. Whenever a peer gets overloaded (i.e., the number of its short links increase beyond a threshold $FLimit$), the peer may delete some of its links. When a peer n decides to delete a link, it selects the link e with peer p that has the largest distance $d(LI(n), RI(n, e))$ among all of its links (that is, the less similar one). Peer p is then connected through a short link with another peer in the group, selected by sending a join message as above. More specifically, before n breaks the link with p, it sends a message that is routed inside the group and tries to find the peer with the less semantic distance with p (that can accept new connections). Then it informs p about its new neighbor in order to establish connection with it. The routing procedure whenever such a message reaches a peer v is the following:

1. The distance $d(LI(v)$, $LI(p))$ between local indexes $LI(v)$ and $LI(p)$ is

calculated.

2. Peer v and the corresponding distance are added to list L (similar to the list used during the join procedure).

3. If the maximum number of visited peers is reached, the routing of the message stops.

4. Else, the distances $d(LI(p), RI(v, e))$ between the local index $LI(p)$ of the peer p and the routing indexes $RI(v, e)$ that correspond to each of the outgoing short links e of peer v are calculated.

5. The message is propagated through the short link e with the smallest distance that has not been followed yet. When the message reaches a peer with no other links that have not been followed, backtracking is used.

When routing stops, the peer n selects the peer of the list L whose local index has the smaller distances from the local index of the p, and informs p about the connection it should create sending a message to it.

After this procedure takes place, peers with fanout greater than $FLimit$ will converge to fanout $FLimit$. Also, the fanout of peers with fanout a little smaller than $FLimit$ will also converge to $FLimit$, since new peers will attach to them. Finally, peers with small fanout will have fewer new neighbors than peers with larger fanout, since the probability the routing of the message to reach these peers is smaller.

3.3.5 Node Leave

Before a peer exits the system, it should decide about the new connections that will be established after it leaves. We must ensure that the network will remain connected and also that the peer's departure will not affect the grouping of the peers. The procedure tries to link the neighbors of the peer between them, keeping the network connected and the properties of the group the peer belongs to unaffected. In order to keep the network connected, the idea is to link in a path all the neighbors of the peer n that leaves. Also, each neighbor that is linked through short link with n, should connect with another neighbor that has short link with n, keeping the grouping of the peers unaffected.

Peer n that wishes to leave, first propagates a message to its neighbors asking for their local indexes and a list NL with their own neighbors. For each of its neighbors peer n stores:

1. the local index

2. the list NL of its own neighbors and the type of links with them

Before peer n leaves, it needs to decide about the new connections that will be created after its departure. The procedure is the following:

While $NL \neq$ *empty*

1. Peer n selects a neighbor k from NL. $NL = NL - \{k\}$
2. If the link between k and n is short ($link(k,n) = short$), another peer $u \in NL$ (neighbor of n) is selected to link with k through short link. This peer should satisfy the following properties: first, it should be connected with n through short link ($link(u,n) = short$), and second, it should be the peer with the smallest histogram distance from k ($\forall$ neighbor z of n $d(LI(k),LI(u)) = \min_z(d(LI(k),LI(z)))$, where $z \in NL$). If $link(k,n) = long$, k is marked to link through long link with peer $u \in NL$ for which $link(u,n) = short$).
3. $k = u$.

List NL is used so that peer n can check whether two of its neighbors are linked together in every iteration, without having to send messages to all the neighbors.

After n has decided about all the new connections that will be created, it propagates a message to each neighbor x containing the new connections x will create. As soon as all the new connections are established n leaves the system. Also the routing indexes of the peers within the horizon of n should be updated after it leaves.

The above procedure ensures that the network remains connected. The reason is that there is a path between all the neighbors of the departing peer n. Also, the grouping of the peers remains unaffected, since each neighbor k of n for which $link(k,n)=short$ connects with the most similar n's neighbor m of the same group (links with n through short link). Whenever a new peer joins the system connects with the most similar of the existing peers. Thus n with high probability will be the most similar peer to m and k, which means that it is expected to be very similar also to each other. This is the reason we only need to look at the neighbors of the peer that leaves for establishing the new connections.

The number of messages needed so that peer n collects all the required information (list NL and local index) is $2*degree(n)$, where $degree(n)$ is the number of $n's$ neighbors. Then for each neighbor, peer n selects another neighbor to link with it, and sends a message to both of them. Thus, $2 * degree(n)$ more messages are needed to inform the neighbors about the new connections that will be created. The total number of messages is $4 * degree(n)$. Of course after the new links are established the routing indexes of the peers within the horizon of the neighbors of n need to be updated.

Note: If n has only one neighbor then there is no need of creating new connections.

Chapter 4

Histogram Distance Metrics

As discussed in the previous section, histograms over one attribute are used as local indexes. The routing of the join message is based on the distance between the histograms. Thus, the distance metric used plays crucial role in the performance of the network. We expect the distance between the two histograms (corresponding to the local indexes of two peers) to be descriptive of the difference in the number of results the peers store. For this reason, we introduce a workload-aware edit distance that in the average case is shown to be analogous to the difference in the number of results returned.

4.1 Distance Metrics

The L_1-distance of two histograms $H(n_1)$ and $H(n_2)$ is defined as:

Definition 3 **(L_1 distance between histograms)** *Let two histograms $H(n_1)$ and $H(n_2)$ with b buckets, their L_1 distance, $d_{L_1}(H(n_1), H(n_2))$ is defined as:* $d_{L_1}(H(n_1), H(n_2)) = \Sigma_{i=0}^{b-1} |H_i(n_1) - H_i(n_2)|$.

For example, the L_1 distance of the histograms in Fig. 4.2 is:
$d_{L_1}(H(n1), H(n2)) = 0.1 + 0 + 0.4 + 0.5 = 1$.

The histograms we study are *ordinal* histograms, that is, there exists an ordering among their buckets, since they are built on numeric attributes. For ordinal histograms, the position of the buckets is important and thus, we want the definition of histogram distance to also take into account this ordering. This property is called *shuffling dependence*. For example, for the three histograms of Fig. 4.1, the distance between histograms $H(n1)$ and $H(n2)$ that have all their values at adjacent buckets ($H_0(n1)$ and $H_1(n2)$ respectively) should be smaller than the distance between histograms $H(n1)$ and $H(n3)$ that have their values at buckets further apart. This is because, the difference between the number of results provided by peer $n1$ and the number of results provided by peer $n2$ is smaller for a larger number of range queries than for peers $n1$ and $n3$. The

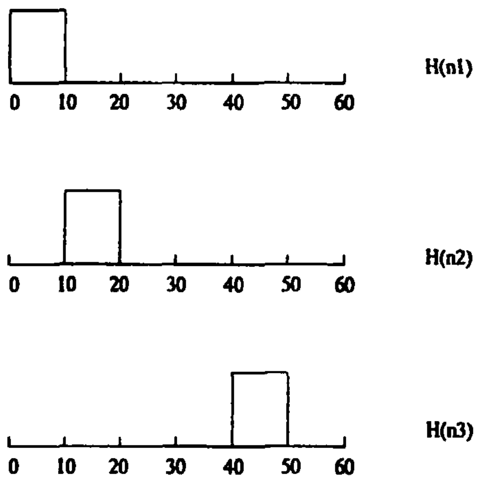

Figure 4.1: Intuitively, the distance between $H(n1)$ and $H(n2)$ should be smaller than the distance between $H(n1)$ and $H(n3)$

shuffling dependence property does not hold for d_{L_1}, since the three histograms have the same pair-wise distance.

We consider an edit distance based similarity metric for which the shuffling dependence property holds. The edit distance of two histograms $H(n_1)$ and $H(n_2)$ is the total number of all necessary minimum movements for transforming $H(n_1)$ to $H(n_2)$ by moving elements to the left or right. It has been shown that this can be expressed by the following definition [7]:

Definition 4 (edit distance between histograms) *Let two histograms $H(n_1)$ and $H(n_2)$ with b buckets. Their edit distance, $d_e(H(n_1), H(n_2))$, is defined as: $d_e(H(n_1), H(n_2)) = \Sigma_{i=0}^{b-1}|\Sigma_{j=0}^{i}(H_j(n_1) - H_j(n_2))|$.*

Let us define as:
$pref(l) = \Sigma_{i=0}^{l} H_i(n_1) - \Sigma_{i=0}^{l} H_i(n_2)$,
where $pref(l) = 0$ for $l \geq b-1$ and $l < 0$

Then, the edit distance can be written as:
$d_e(H(n_1), H(n_2)) = \Sigma_{l=0}^{b-1}|pref(l)|$

Edit distance takes the preffix sums of the array that represent the histogram. Consider the two histograms of Fig. 4.2. There edit distance is: $d_e(H(n1), H(n2)) = 0.1 + 0.1 + 0.5 + 0 = 0.7$.

We want a distance metric that will be descriptive of the difference in the number of results for a given query (Property 1). In the ideal case, we want the distance between two histograms $H(n_1)$ and $H(n_2)$ to correspond to the difference in the number of results between the peers n_1 and n_2 for each query.

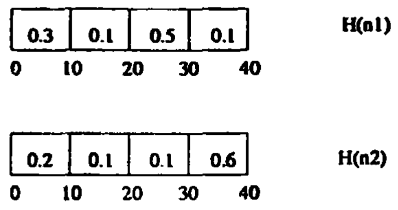

Figure 4.2: Example of two histograms over an attribute $x \in [0,40]$

For a query q_{kj}, the estimated difference in the results, *hdiffer*, of two peers n_1 and n_2 is:
$hdiffer(n_1, n_2, q_{kj}) = |hresults(H(n_1), q_{kj})/S(H(n_1)) - hresults(H(n_2), q_{kj})/S(H(n_2))|$
$= |\Sigma_{i=j}^{j+k}(H_i(n_1) - H_i(n_2))|$

The difference in the results for query q_{kj} is equal to:

$$hdiffer(n_1, n_2, q_{kj}) = |pref(j+k) - pref(j-1)| \qquad (4.1)$$

(proof in the Appendix).

From Equation 4.1, for $k = b-1$ that is for queries of the form $q_{>i}$ Property 1 holds for edit distance. It also holds for queries of the form $q_{<i}$. It does not hold however, in general, so we extend Definition 4 as follows, based on Equation 4.1.

Definition 5 (workload-aware edit measure between histograms) *Let two histograms $H(n_1)$ and $H(n_2)$ with b buckets. Their workload-aware edit measure, $wd_e(H(n_1), H(n_2))$, is defined as:*

$$wd_e(H(n_1), H(n_2)) =$$
$$\Sigma_{k=0}^{b-1}\, \Sigma_{j=0}^{b-1} w_{kj} |pref(j+k) - pref(j-1)|$$

where $0 \leq w_{kj} \leq 1$ and $\Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj} = 1$.

We shall show that wd_e satisfies Property 1, if we adjust the weights properly. Let a query q_{kj}; recall that j is the bucket the query starts from and k is the range of the query. Property 1 holds for q_{kj}, if we set $w_{kj} = 1$ and all other weights to zero. In this case, $wd_e = hdiffer(n_1, n_2, q_{kj})$, which means that the distance between the histograms is equal to the difference in the number of results returned for query q_{kj}.

In general, let f_{kj} be the frequency of the queries q_{kj} in the workload. By setting $w_{kj} = f_{kj}$, the wd_e distance approximates the expected difference in the number of results over all queries. Thus, in general, when we have knowledge about the range k of the query workload and the starting bucket j, we can use this knowledge to favor popular queries. By using the frequencies of the queries as weights, differences in the result sets of more popular queries influence more the distance of the histograms, while unpopular queries have a smaller impact. When no knowledge about the workload is available we can simply use d_e.

We use the same technique to extend w_{L_1} to take into account the workload. Let us define as:
$L1(i) = H_i(n_1) - H_i(n_2)$
then, the L_1 distance can be written as:
$d_{L_1}(H(n_1), H(n_2)) = \Sigma_{l=0}^{b-1}|L1(l)|$

Similarly to Equation 4.1 (proof in the Appendix),

$$hdiffer(n_1, n_2, q_{kj}) = \Sigma_{i=j}^{j+k}|L1(i)| \qquad (4.2)$$

Thus, L_1 distance is analogous to the difference in the number of results only for the queries q_{kj} with $k = 0$, that is for queries covering a single bucket. For being the distance descriptive of the difference in the number of results (Property 1), we define a workload-aware version of the L_1-distance as follows based on Equation 4.2:

Definition 6 (workload-aware L_1 measure between histograms) *Let two histograms $H(n_1)$ and $H(n_2)$ with b buckets. Their workload-aware L_1 measure, $wd_{L_1}(H(n_1), H(n_2))$, is defined as:*

$$wd_{L_1}(H(n_1), H(n_2)) = \Sigma_{k=0}^{b-1}\ \Sigma_{j=0}^{b-1} w_{kj}|\Sigma_{i=j}^{j+k}\ L1(i)|$$

where $0 \leq w_{kj} \leq 1$ *and* $\Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj} = 1$.

By incorporating weights into d_{L_1} the new metric is shuffling dependent and it is proved to be equivalent to wd_e. Both weighted versions (Definitions 5 and 6) are distance metrics. Proofs can be found in the Appendix.

4.2 Experimental Evaluation

We run a set of experiments to evaluate the histogram distance metrics. For simplicity of presentation, in the reported experiment, we use histograms with 10 buckets and $x \in [0, 99]$. We used a workload with queries having range (k) varying from 0 (covering data in 1 bucket) to 4 (covering data in 5 buckets). We use 10 histograms $H(i)$, $0 \leq i < 10$, with 10 buckets each. Two different data distributions are used. In the first, for histogram $H(i)$, DT is the fraction of the total number of tuples that are included within bucket i and the rest tuples are uniformly distributed among the rest of the buckets. We also used the zipf data distribution. For each histogram there is a ranking of the buckets based on the number of the tuples each bucket summarizes. The probability a tuple to belong to the bucket with rank r is analogous to $P_i = 1/r^a$, where a is a parameter that varies in our experiments. For histogram $H(i)$ bucket i is set to be the most popular and the popularity reduces as the distance between bucket i and the other buckets increases. Another parameter in our experiments is the number bct of buckets that include the $(100 * DT)$% of the tuples (in our data distribution) or that are the most popular (in zipf distribution). More specifically, for histogram $H(i)$ buckets $i, ..., i+bct-1$ include the $(100*DT)$% of the tuples (for

Table 4.1: Input parameters for centralized experiments (Simple Histograms)

Parameter	Default Value	Range
Number of buckets	10	
Domain of x	[0, 99]	
Tuples per peer	1000	
Range of queries		0-4
DT	0.7	0.5-0.9
a	1.0	0.9-3.0
bct	1	1-3

our data distribution) or are the most popular (for zipf distribution). We compute the distance between $H(0)$ and each histogram $H(i)$, $1 \leq i < 10$, using the three distance metrics. Our performance measure is the difference in the number of results for each histogram with $H(0)$, that is: $hdiffer(H(i), H(0), q_{kj})$ with respect to the distance of the respected histograms (that is, whether Property 1 is satisfied). The desired behavior is the difference of the number of results estimated by two histograms to be analogous to their distance. Table 4.1 summarizes the parameters of the experiments.

4.2.1 Our Data Distribution

For the first experiment, the starting bucket j of the query is chosen uniformly and performed the experiments for different ranges r. As expected, the L_1 distance (Fig. 4.3(left)) does not perform well, especially for our data distribution. The distance of the histograms has no relation with the difference in the number of results. This is because the L_1 distance compares only the respective buckets of each histogram without taking into account their neighboring buckets which however influence the behavior of queries with ranges larger than 0.

The edit distance (Fig. 4.3(center)) performs better than L_1, since it takes into account the position of the buckets. In particular, as the distance between the histograms increases, their respective differences in the results also increase. However, for each query range this occurs until some point after which the difference in the results becomes constant irrespectively of the histogram distance. This is explained as follows. The edit distance between two histograms takes into account the ordering of all buckets, while a query with range r involves only r buckets, and thus it does not depend on the difference that the two histograms may have in the rest of their buckets. For example, for a query with range 0, the difference in the results remains constant while the histogram distances increase, because the query involves only single buckets while the edit distance considers the whole histogram. The edit distance works better for queries with large ranges.

The behavior of the workload-aware edit distance is depicted in Fig. 4.3(right).

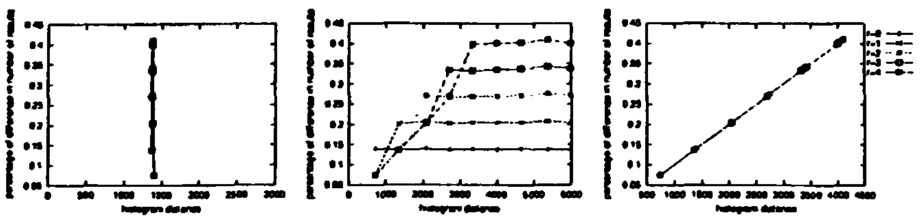

Figure 4.3: (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for our data distribution and for different query ranges ($DT = 0.7$ and $bct = 1$)

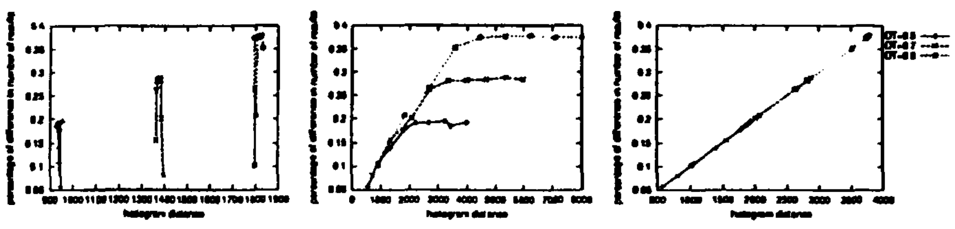

Figure 4.4: (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for our data distribution when varying DT (mixed query workload)

For each query, we set the weights that correspond to the range of the query to 1 and all other weights to zero. More specifically, for a query with range k, we set $w_{kj} = 1, \forall\ j$. The distance is proportional to the difference in the number of results (which is the ideal performance).

For the following experiments, the starting bucket j of the query is chosen uniformly and we use different ranges (between 0 and 4) for the queries, where queries of different ranges have different frequencies.

Figures 4.4 and 4.5 present the performance of the distance metrics for our data distribution when varying DT and bct correspondingly. As expected, the L_1 distance (Fig. 4.4 (left) and 4.5(left)) does not perform well. When bct=1, due to the nature of the data (for each histogram the large data distribution

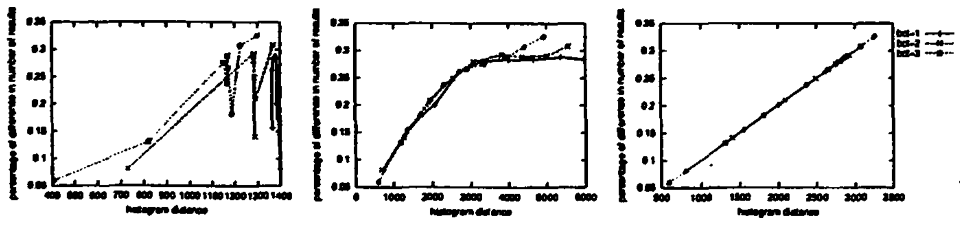

Figure 4.5: (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for our data distribution ($DT = 0.7$) varying bct (mixed query workload)

exists within a different bucket), all compared histograms have nearly the same L_1 distance with $H(0)$. When $bct > 1$, there is an overlapping between that buckets that correspond to the large data distribution. Thus the L_1 distance is not always the same, however again it does not perform well since the histogram distance does not follow always the difference in the number of results.

The edit distance (Fig. 4.4 (center) and 4.5(center)) performs better than L_1, since it takes into account the order of buckets. In particular, as the distance between the histograms increases, their respective differences in the results also increase. However, for each value of DT this occurs until some point after which the difference in the results becomes constant irrespectively of the histogram distance.

The performance of the workload-aware edit distance is depicted in figures 4.4 (right) and 4.5(right). The weights are set equal to the frequency of the queries. More specifically, $w_{kj} = f_k$ $\forall$ starting bucket j, where f_k is the frequency of the queries with range k.The performance is the ideal. As proved, the difference in the results increases analogously to the difference of the histograms.

We also evaluated the workload-aware edit distance for a workload of queries with range from 0 to 4, for the case in which the starting bucket of the queries is not chosen uniformly but some starting points are more popular (Fig. 4.9 (left) and 4.10 (left)). The weights are: $w_{kj} = f_{kj}$ where k is the range and j the starting bucket of the query, and f_{kj} is the frequency of the queries with range k starting from bucket j. The distance satisfies Property 1 for this query workload as well. Although for this workload the histogram distance is not exactly analogous to the difference in the number of results, it satisfies the property that the largest the histogram distance, the largest the difference in the results.

4.2.2 Zipf Data Distribution

For the first experiment, where the starting bucket j of the query is chosen uniformly and used different query ranges r, L_1 (Fig. 4.6(left)) performs better than when using our data distribution. The reason is the distribution of the data among more buckets. Edit distance (Fig. 4.6(center)) also performs better for zipf distribution for the same reason. Workload-aware edit distance (Fig. 4.6(right)) performs better then the other two distance metrics and it is proportional to the difference in the number of results.

We performed also the experiments using different ranges (between 0 and 4) for the queries, where queries of different ranges have different frequencies (Fig. 4.7 and 4.8). The workload-aware edit distance (Fig. 4.7 (right) and 4.8 (right)) has the same performance as above. The L_1 (Fig. 4.7 (left) and 4.8 (left)) and *edit* distance (Fig. 4.7 (center) and 4.8 (center)) perform better due to the nature of the data (the tuples are distributed across more buckets). However their performance is worse than the workload-aware edit distance, since the histogram distance is not analogous to the difference in the number of results. Also, for large values of a (which means that the most popular buckets have much more tuples than the rest), the performance decreases, since the tuples

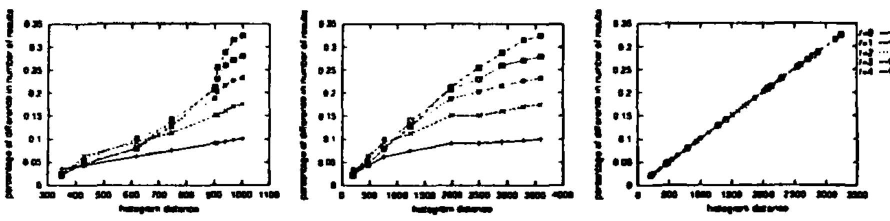
Figure 4.6: (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for zipf data distribution and for different query ranges ($a = 0.9$ and $bct = 1$)

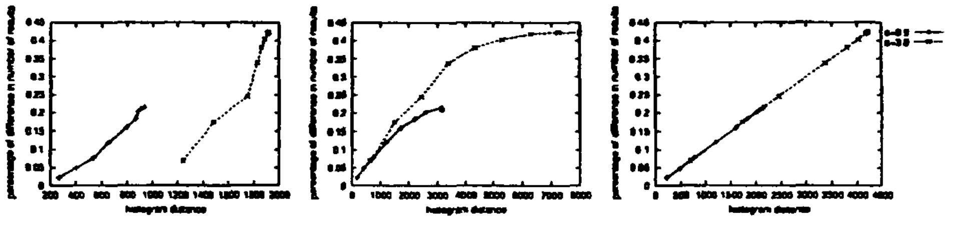
Figure 4.7: (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for zipf distribution, when varying a (mixed query workload)

are distributed among less buckets.

Also, for the case in which the starting bucket of the queries is not chosen uniformly but some starting points are more popular (Fig. 4.9 (right) and 4.10 (right)), workload-aware edit distance satisfies Property 1 also for this data distribution. Although for this workload the histogram distance is not exactly analogous to the difference in the number of results, it satisfies the property that the largest the histogram distance, the largest the difference in the results.

4.2.3 Summary of Experiments

To conclude, for both distributions and for all the values of DT and a, the workload-aware edit distance performs better than L_1 and edit. The histogram distance is nearly analogous to the difference in the number of the results. When

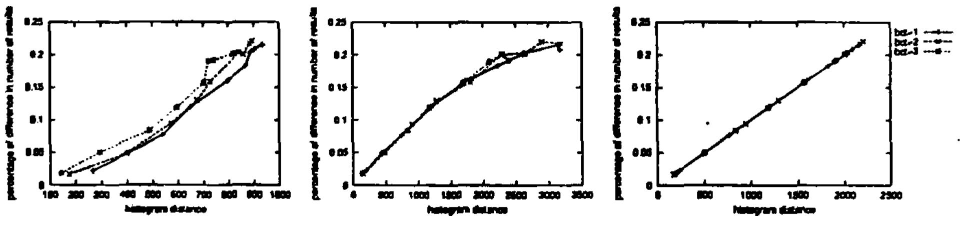
Figure 4.8: (left) L1 distance, (center) edit distance and (right) workload-aware edit distance for zipf distribution ($a = 0.9$) varying bct (mixed query workload)

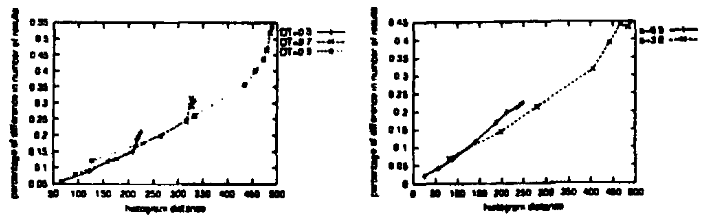

Figure 4.9: (left) workload-aware edit distance for our data distribution, and (right) for zipf distribution, when varying DT and a correspondingly (mixed query workload where the starting bucket is not selected uniformly)

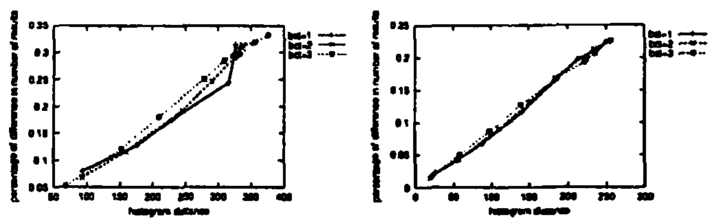

Figure 4.10: (left) workload-aware edit distance for mixed query workload for our data distribution ($DT = 0.7$), and (right) for zipf distribution ($a = 0.9$), when varying bct (mixed query workload where the starting bucket is not selected uniformly)

using L_1 distance metric, the histogram distance is nearly constant independently of the difference in the number of results the histograms summarize. Edit distance is nearly proportional to the difference in the results until some value of histogram distance. Beyond that point the distance increases although the difference in the number of results remains constant. This point depends on the range of the query. As the query range increases, edit distance metric performs better. Also, L_1 and edit distance metrics perform better for zipf distribution and when increasing bct, due to the distribution of the tuples to more buckets.

Chapter 5

Multi-Attribute Histograms

So far, we have assumed that the user queries the system based only on the range of one attribute. Now we will study the situation in which the query consists of the range of more than one attribute. Consider a relational database with *attr* numerical attributes A_1, ..., A_{attr}, distributed among the peers of a peer-to-peer network. Each peer maintains a local index that summarizes the information stored locally. The simplest solution is to create a different histogram that approximates the values for each attribute. The problem with this approach is that there is dependence between the values of the attributes that should be taken into account in the index.

5.1 Histogram Structure

The R-tree [11] structure is very close to what we need. Each level of the R-tree represents the distribution of the values of a particular attribute. This structure is more suitable for equi-depth histograms. The problem with this type of histogram is that the update cost is very large, since whenever new tuples are inserted, the R-tree will need to be built again. Also, the merge cost is large and this makes difficult the construction of the routing indexes. Finally it is more difficult to define an appropriate distance metric between equi-depth histograms.

We use a variation of the R-tree, where the main difference is that each node of the tree corresponds to an equi-width histogram, instead of an equi-depth. Each node represents a histogram and approximates the value frequencies of the corresponding attribute in a particular region. The total number of regions that the attributes space is split is $b_1 * b_2 * ... * b_{attr}$, where b_1, b_2,...,b_{attr} is the number of buckets for the attributes A_1,...,A_{attr} correspondingly. The buckets of the leaf nodes represent the regions to which the attribute space is split by the tree.

An example of the above approach follows (Fig. 5.1). Consider three attributes A, B, C. The values of each attribute are split into three equi-width

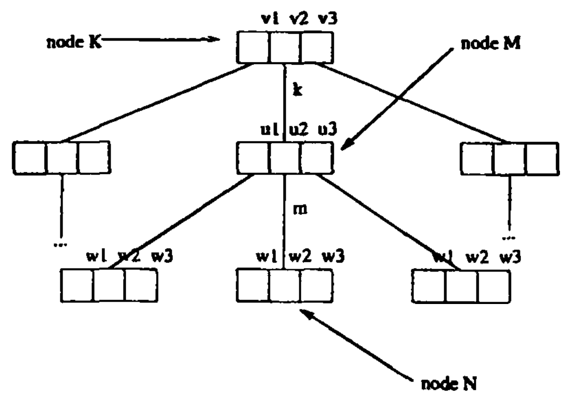

Figure 5.1: 3-Attribute Histogram

buckets. The histogram for these attributes is a tree with three levels, each one corresponding to a particular attribute.

In the first level, we approximate the frequencies of attribute A in each bucket. In the second level, for the tuples in each bucket of the first level, we approximate the frequencies of attribute B. For example, for the tuples with values between $v1$ and $v2$ for A, we approximate the values of attribute B. Similarly, in the third level, for the tuples in each bucket of the second level, we approximate the frequencies of attribute C. For example, node N corresponds to the histogram that approximates the frequencies of the values for attribute C for the tuples with values between $v1$ and $v2$ for attribute A and between $u1$ and $u2$ for attribute B.

An issue is how we select the attribute that will be partitioned at each level. We select to partition the most critical attribute at each level. Since each query may not contain ranges over all the attributes of the multi-attribute histogram, we prefer the higher levels to correspond to the attributes that are queried more. Thus, smaller part of the tree will need to be traversed during the results estimation.

Now we define the routing index for the multi-attribute histograms similarly to the Definition 1 of Section 3.1. We shall use the notation $H^{lj}(n)$ to denote a histogram that corresponds to the $j-th$ node of level l in the tree (used either as a local index $LI^{lj}(n)$ or as a routing index $RI^{lj}(n,e)$), $H_i^{lj}(n)$ to denote the frequency of the values within its i-th bucket, $0 \leq i \leq b_l - 1$, where b_l is the number of buckets for the attribute at level l, $1 \leq l \leq attr$, where $attr$ is the number of attributes the histogram summarizes. Also, we use $S(H(n))$ to denote its size of the multi-attribute histogram. Then,

Definition 7 (histogram-based routing index) *The frequency of the values within $i-th$ bucket for the histogram that corresponds to the $j-th$ node of the attribute at level l of the histogram-based routing index $RI(n,e)$ of the link e of peer n is defined as:* $RI_i^{lj}(n,e) = \Sigma_{p \in P}(LI_i^{lj}(p) * S(LI(p)))/\Sigma_{p \in P} S(LI(p))$

and $S(RI(n,e)) = \Sigma_{p\in P} S(LI(p))$ *where* P *is the set of peers* p *within distance* R *of* n *reachable through link* e.

5.2 Distance Metric

Consider a multi-attribute histogram of $attr$ numerical attributes $A_1,\ldots,A_{attr}$. Let b_1, $b_2,\ldots,b_{attr}$ be the number of buckets that corresponds to each attribute. Also we assume that attribute A_i corresponds to level i in the tree. We can use each of the known histogram distance metrics to calculate the distance between two multi-attribute histograms. Lets $d(H^{lj}(n_1), H^{lj}(n_2))$ be the distance between two nodes of two multi-level histograms that correspond to the j-th node of level l in the tree. The distance between the multi-level histograms is calculated by taking the sum of the distances for each level of the tree. The distance for level l is:

$$ldist_l = \Sigma_{j=1}^{b_{l-1}^{l-1}} d(H^{lj}(n_1), H^{lj}(n_2))/b_{l-1}^{l-1} \tag{5.1}$$

We take the sum of the distances between the corresponding nodes of the level and divide it with the number of the nodes to give equal weight to each attribute when calculating the distance of the whole multi-level histogram.
The total distance between two multi-attribute histograms $H(n_1)$ and $H(n_2)$ is:

$$D(H(n_1), H(n_2)) = \Sigma_{l=1}^{attr} ldist_l \tag{5.2}$$

where $attr$ is the number of attributes.

This distance metric gives equal weight to each level of the tree (to each attribute) and also to each node in the same level. Thus, each region the tree splits the attribute space has the same contribution to the distance.

In Section 4.1 we have shown that adjusting the weights properly the workload-aware edit distance between two histograms in the average case is equal to the difference in the number of results for a given query. Using this distance in equation 5.1, $ldiff_l$ gives the average difference in the number of results for attribute A_l taking into account the distribution of the values for the attributes of the higher levels in the tree. Thus, distance D that corresponds to the sum for all the levels, in the average case is analogous to the total difference in the number of results.

5.3 Estimation of Results

Our system support complex range queries over more then one attributes. The query can be a conjunction (i.g. $v1 \leq A < v2$ and $u1 \leq B < u2$ and $w1 \leq C < w2$) or a disjunction (i.g. $v1 \leq A < v2$ or $u1 \leq B < u2$ or $w1 \leq C < w2$) of clauses over the attributes. A query may also contain both disjunctions and conjunctions (i.g. $v1 \leq A < v2$ and ($v1 \leq B < v2$ or $w1 \leq C < w2$)).

During the query routing, the query message is propagated through the link that more results are expected to be found. This estimation is done based on the routing indexes, in our approach the multi-dimensional histograms. In order to estimate the expected number of results stored at the peer that corresponds to the multi-dimensional histogram, the buckets of the leaf nodes that satisfy the query need to be specified (the buckets that overlap with the range of the query). During the traversal of the tree from the root to leaves in a Breadth First Search manner, only the links that correspond to buckets that overlap with the predicates of the query are followed. The procedure continues until the last level of the tree has been reached. The buckets of the leaf nodes we finally reach will be taken into account for the results estimation. More specifically, the expected number of results is the sum of the number of tuples each bucket summarizes.

For example consider the 3-attribute histogram of Fig. 5.1 and the conjunction query: q_1 and q_2. Lets assume the predicate q_1 of the form $u1 < B < u2$, where B is the attribute of the second level and q_2 of the form $w1 < C < w2$, where C is the attribute of the third level in the above schema. In order to specify the leaf buckets that satisfy this query the procedure is the following: all the links of the first level's buckets are followed. For each node of the second level (that corresponds to attribute$B$) we select the buckets that overlap with the range of the query q_1. Then only the links of these buckets are followed leading to the corresponding nodes of the next level.

Continuing, we evaluate the second predicate q_2. We follow the same procedure starting from the nodes reached by the above procedure. When reaching the third level, the buckets that overlap with q_2 are selected. These are the buckets that should be taken into account for the results estimation.

We denote S the set of these buckets. The estimated number of results is the sum of the number of tuples each bucket bc^m of set S summarizes:

$Est = \Sigma_{m=1}^{C} tuples(bc^m)$

where C is the size of S and $tuples(bc^m)$ the number of tuples bucket bc^m summarizes.

If the query consists of both conjunctions and disjunctions, then it is transformed into disjunctions of the conjunctions (Disjunctive Normal Form) and we handle each conjunction separately. The estimated number of results is the maximum number of the results estimated by the conjunctions. For example, a query: q_1 and (q_2 or q_3) is transformed into: (q_1 and q_2) or (q_1 and q_3). The estimated number of results for the query (q_1 and q_2) and the query (q_1 and q_3) is calculated, and we take the sum of them as the results estimation.

In the above procedure, we assumed that the buckets satisfying a query are included as a whole to the query range. In the case that the query range overlaps with only a part of a bucket, the uniform frequency assumption for the values of each bucket is assumed (as in simple histograms) to estimate the number of results. In particular, each bucket in S does not contribute the total number of tuples it summarizes to the estimated number of results, but only the expected number of tuples that satisfy the query based on the uniform frequency assumption. More specifically, for each bucket bc^m in S lets bc_i^m be the bucket at level i of the path from the root to bc^m. Let $perc_i^m$ be the fraction of the bucket bc_i^m that overlaps with the query ($perc_i^m$ is evaluated based on the equations

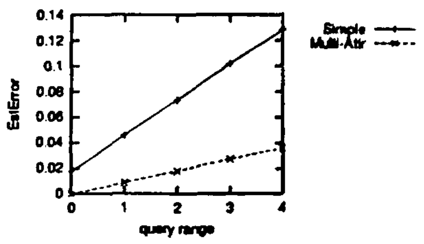

Figure 5.2: Error in the estimated number of results for multi-attribute and simple histogram

(1)-(3) of section 3.1). Then, the number of tuples that bc^m contributes to the estimated number of results is:

$Est^m = tuples(bc^m) * \Sigma_{i=1}^{attr} perc_i^m$, where *attr* is the number of attributes. Thus, the estimated number of results is:

$Est = \Sigma_{m=1}^{C} Est^m$.

5.4 Using N Simple Histograms Instead of an N-Dimensional Histogram

Instead of using multi-dimensional histograms when there are more that one attributes, an alternative structure is the use of one simple histogram per attribute. The whole histogram consists of *attr* simple histograms, where *attr* is the number of attributes. The problem with this approach is that it does not take into account the dependence between the values of different attributes. Instead, we assume the values of different attributes are independent between them. Let $H_i(n)$ be the simple histogram for attribute i and $perc(H_i(n), q^i)$ the fraction of the tuples the query predicate for attribute i satisfies (similar to Section 3.1). Assuming that the attributes values are related with the uniform distribution, the estimated number of results is:
$hresults(H(n), q) = perc(H_1(n), q^1) * perc(H_2(n), q^2) * ... * perc(H_{attr}(n), q^{attr}) * S(H(n))$.

We performed experiments to calculate the difference *EstimatedError* between the actual number of results $results(n, q)$ and the estimated number of results $hresults(H(n), q)$ for a query q, a peer n and its histogram $H(n)$, when there are two attributes.

$EstError = |results(n, q) - hresults(H(n), q)|/S(H(n))$

We compare the use of two simple histograms with the two-dimensional histogram (Fig. 5.2).

We notice that the use of a two-attribute histogram performs much better for all query ranges, since it takes into account the dependence between the values of different attributes. Also, the error in the estimation is analogous to the range of the query, due to the greater number of results when increasing the range. Also the increase rate is smaller for multi-attribute histograms.

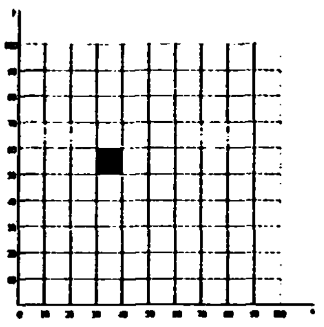

Figure 5.3: The bold square summarizes the tuples with $x \in [30,39]$ and $y \in [50,59]$.

5.5 Experimental Results

We run a set of experiments to evaluate the histogram distance metrics. For simplicity of presentation, in the reported experiment, we use two-attribute histograms on attributes x and y with 10 buckets per attribute and x, $y \in [0,99]$. The tuples of the two attributes are summarized by a multi-attribute histogram with 10 buckets for each level (attribute). Consider a two-dimensional space, each dimension representing the tuple values on one of the attributes (Fig. 5.3). Summarizing the tuples based on a two-attribute histogram, each tuple corresponds to a square in the two-dimensional space based on the values of the attributes x and y, as shown in figure 5.3. We used a mixed workload with queries having range (k) varying from 0 (covering data in 1 bucket) to 4 (covering data in 5 buckets) for each attribute. We use 100 two-attribute histograms $H(i,l)$, $0 \leq i < 10$ and $0 \leq l < 10$, with 10 buckets per attribute each one. Also we evaluate the multi-attribute histograms for two data distributions. In the first, for each histogram $H(i,l)$, DT is the fraction of the total number of tuples that are included within the square defined by the buckets i and l for attributes x and y correspondingly, and the rest tuples are uniformly distributed among the rest of the buckets. We also used the zipf data distribution, where there is a ranking of squares (i,l) based on the number of the tuples with values for x, y within each square. We use two values for parameter a, 0.9 and 3. Similarly to the experiments with simple histograms, we vary the number *bct* of buckets that include most tuples from 1 to 3. We compute the distance of each histogram with $H(0,0)$ using the three distance metrics. Our performance measure is the difference in the number of results for each histogram with $H(0,0)$, that is: $hdiffer(H(i,l), H(0,0), q^{r}_{kj})$ with respect to the distance of the respected histograms (that is, whether Property 1 is satisfied). The desired behavior is the difference of the number of results estimated by two histograms to be analogous to their distance. Table 5.1 summarizes the parameters of the experiments.

Table 5.1: Input parameters for centralized experiments (Multi-Attribute Histograms)

Parameter	Default Value	Range
Number of histograms	100	
Number of attributes	2	
Number of buckets per attribute	10	
Domain of x, y	[0, 99]	
Tuples per peer	1000	
Range of queries		0-4
DT	0.7	0.5-0.9
a	0.9	0.9-3.0
bct	1	1-3

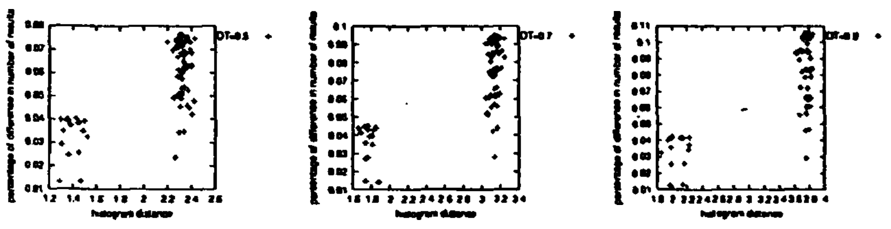

Figure 5.4: L1 distance for our data distribution for various values of *DT* (mixed query workload)

5.5.1 Our Data Distribution

We calculated the average performance of the three distance metrics for a mixed query workload of queries with range from 0 to-4. The starting bucket of the query is chosen uniformly and the weights for the weighted edit distance are set equal to the frequency of the queries. More specifically, $w^r_{kj} = f^r_k$ $\forall$ starting bucket j, where f^r_k is the frequency of the queries with range k for the attribute that corresponds to the query predicate r.

L_1 has the worst overall performance. Using our data distribution (Fig. 5.4 and 5.5), although the distance between many histograms is constant, the difference in the number of results is not (especially when $bct < 3$).

The edit distance behaves better. For our data distribution (Fig. 5.6 and 5.7), the difference in results increases as the histogram distance increases, but in many cases the histogram distance increases although the difference in the number of results remains constant. Also, when $bct > 2$ the behavior is even better, since the tuples are more distributed across the buckets.

The workload-aware edit distance follows the desired property for all the values of *DT* and *bct* (Fig. 5.8 and 5.9). The difference in the number of results is analogous on average to the histogram distance.

In the next experiment (Fig. 5.10 and 5.11) we evaluate the workload-aware

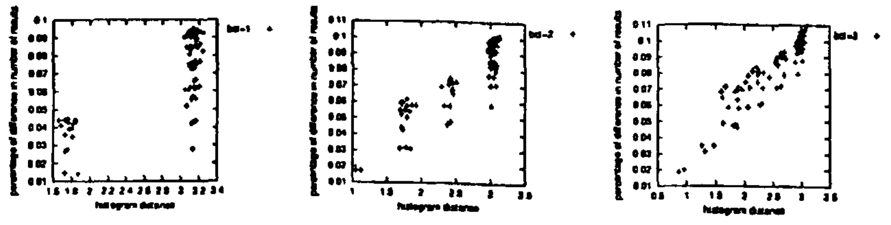

Figure 5.5: L1 distance for our data distribution ($DT = 0.7$) for various values of *bct* (mixed query workload)

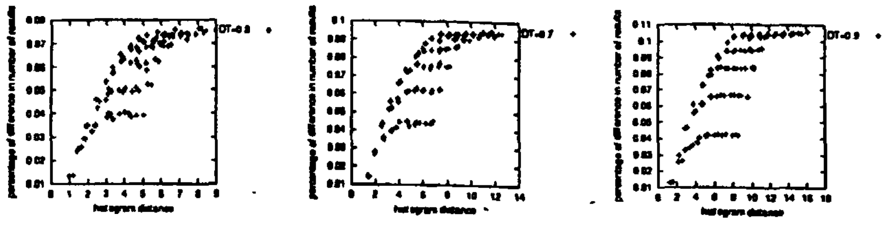

Figure 5.6: Edit distance for our data distribution for various values of DT (mixed query workload)

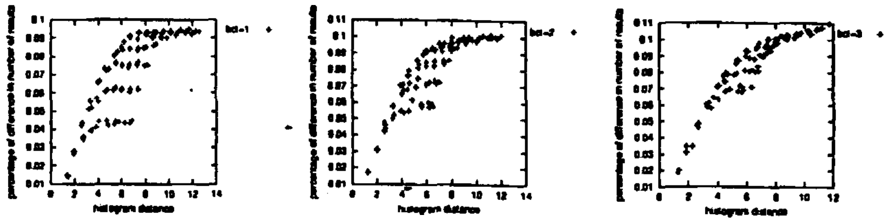

Figure 5.7: Edit distance for our data distribution ($DT = 0.7$) for various values of *bct* (mixed query workload)

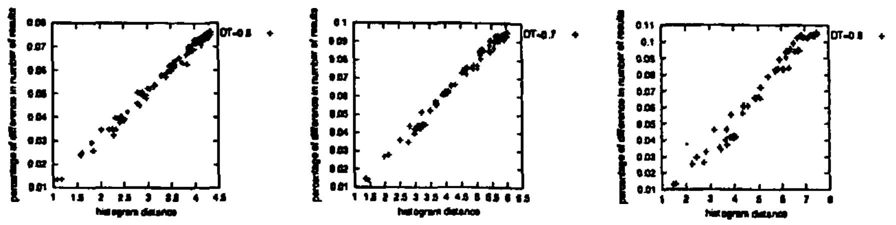

Figure 5.8: Workload-aware edit distance for our data distribution for various values of DT (mixed query workload)

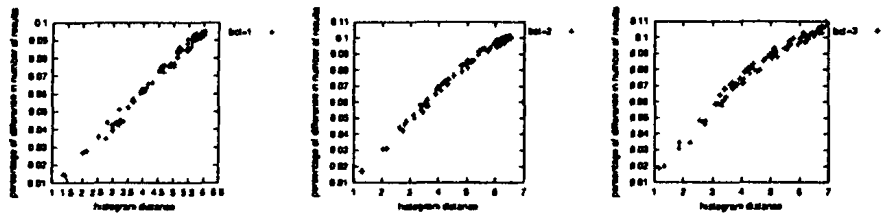

Figure 5.9: Workload-aware edit distance for our data distribution ($DT = 0.7$) for various values of *bct* (mixed query workload)

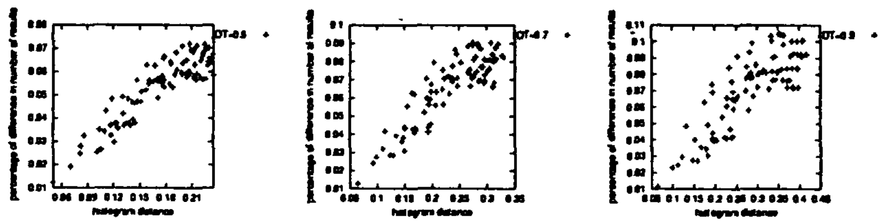

Figure 5.10: Workload-aware edit distance for our data distribution for various values of DT (mixed query workload where the starting bucket is not selected uniformly)

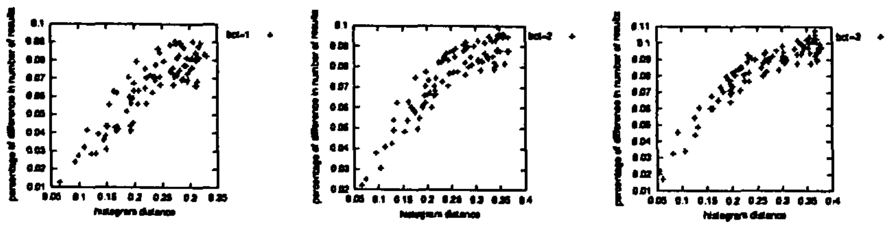

Figure 5.11: Workload-aware edit distance for our data distribution ($DT = 0.7$) for various values of *bct*

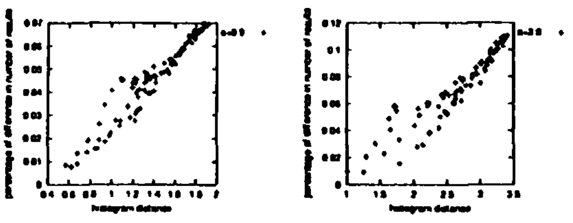

Figure 5.12: L1 distance for zipf data distribution for various values of a (mixed query workload)

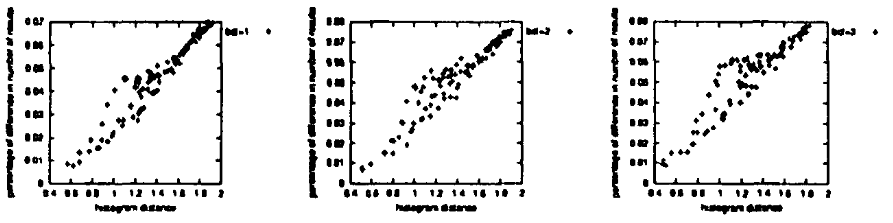

Figure 5.13: L1 distance for zipf data distribution ($a = 0.9$) for various values of *bct* (mixed query workload)

edit distance for a mixed query workload of queries with range from 0 to 4, for the case in which the starting bucket of each query predicate is not chosen uniformly but some starting points are more popular. The weights are: $w^r_{kj} = f^r_{kj}$ where k is the range and j the starting bucket of the query predicate r, and f^r_{kj} is the frequency of the query with range k starting from bucket j for the attribute that corresponds to query predicate r. The distance satisfies Property 1 for this query workload as well in the average case, although there is greater variance in our data distribution.

5.5.2 Zipf Data Distribution

Also for zipf distribution, we calculated the average performance of the three distance metrics for a mixed query workload of queries with range from 0 to 4, where the starting bucket of the queries is chosen uniformly and the weights for the weighted edit distance are set equal to the frequency of the queries.

L_1 (Fig. 5.12 and 5.13) has the worst overall performance, although performs better than in the other distribution. The reason is that the tuples are more distributed across the buckets. For large values of a, which means that the most popular squares have much more tuples than the rest, the performance decreases, since the tuples are distributed across less squares.

Edit distance behaves better than L_1. For zipf distribution (Fig. 5.14 and 5.15) the performance is much better (thatn in our data distribution) due to the distribution of the data across more buckets. For large values of a, the performance decreases also for this distance metric.

The workload-aware edit distance follows the desired property a and *bct* (Fig.

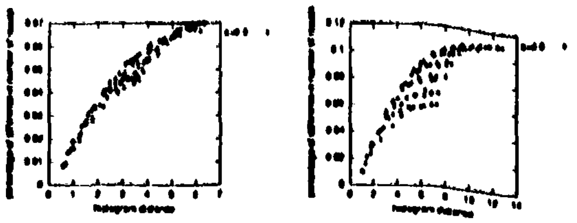
Figure 5.14: Edit distance for zipf data distribution for various values of a (mixed query workload)

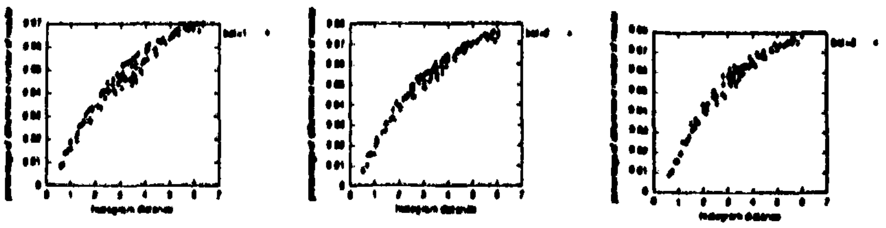
Figure 5.15: Edit distance for zipf data distribution ($a = 0.9$) for various values of bct (mixed query workload)

5.16, 5.17). The difference in the number of results is analogous on average to the histogram distance.

In the next experiment (Fig. 5.18 and 5.19) we evaluate the workload-aware edit distance for a mixed query workload of queries with range from 0 to 4, for the case in which the starting bucket of each query predicate is not chosen uniformly but some starting points are more popular. This distance metric satisfies Property 1 also for this query workload and data distribution.

5.5.3 Summary of Experiments

To conclude,the experiments show that for both distributions and for all the values of DT and a, the workload-aware edit distance performs better than L_1 and edit. The histogram distance is nearly analogous to the difference in the number of the results. Also, all the distance metrics perform better for zipf

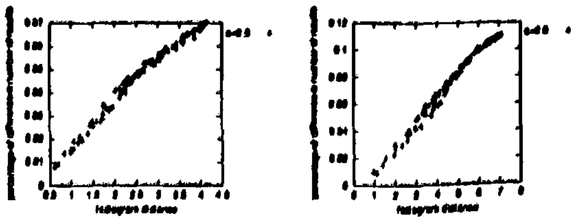
Figure 5.16: Workload-aware edit distance for zipf data distribution for various values of a (mixed query workload)

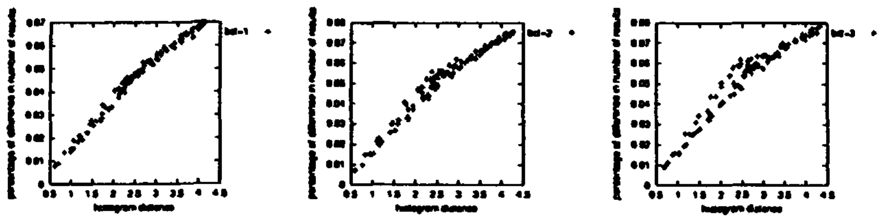

Figure 5.17: Workload-aware edit distance for zipf data distribution ($a = 0.9$) for various values of *bct* (mixed query workload)

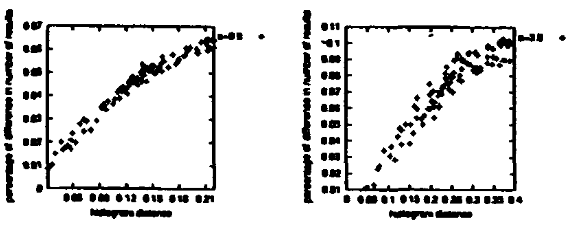

Figure 5.18: Workload-aware edit distance for zipf data distribution for various values of a (mixed query workload where the starting bucket is not selected uniformly)

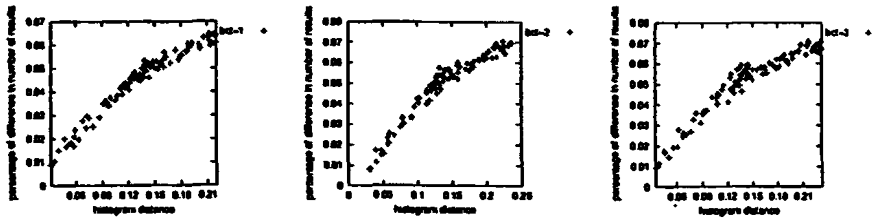

Figure 5.19: Workload-aware edit distance for zipf data distribution ($a = 0.9$) for various values of *bct*

distribution and when increasing *bct*, due to the distribution of the tuples to more buckets.

Chapter 6

Experimental Evaluation

In this section we evaluate the network built following our construction procedure. We show experimentally that it follows the small-world properties. We also evaluate the performance of the network on answering queries. We study the effect of the different distance metrics on the topology of the constructed network and the querying performance.

We simulated the peer-to-peer network as a graph where each node corresponds to a peer. The size of the network varies from 500 to 1500 peers and the radius of the horizon from 1 to 3. Each new peer creates 1 to 2 short links (SL = 1 or 2) and one long link with probability P_l that varies between 0.2 and 0.6. The routing of the join message stops when a maximum number ($JMaxVisited$) of $logM$ peers is visited, where M is the number of peers in the network. The routing of a query stops when a maximum number ($MaxVisited$) of peers is visited. This number is set to 5% of the existing peers in the network. The behavior of the network is evaluated also with peers joining and leaving the system (with the same rate).

We evaluate the small-world construction procedure when the L_1 distance (*L1_dist*), the edit distance (*edit_dist*) and the workload-aware edit distance (*wedit_dist*) is used. We also compare the constructed small-world network with a randomly constructed p2p network, that is a p2p system in which each new peer connects randomly to existing peers (random construction and routing) (*random*) and a randomly constructed p2p system that uses histograms only for query routing (*random_join*). Finally, we performed experiments to see how the network performs when large number of peers leave the system and the effect of the load balancing procedure in the distribution of the links among the peers.

6.1 Using Simple Histograms

Each peer stores a relation with an integer attribute $x \in [0, 999]$ that contains 1000 tuples. The tuples are summarized by a histogram with 100 buckets. In the

Table 6.1: Input parameters (Simple Histograms)

Parameter	Default Value	Range
Number of peers M	500	500-1500
Radius of the horizon	2	1-3
Number of short links (SL)	2	1-2
Probability of long link (P_l)	0.4	0.2-0.6
Perc of peers visited during query routing($MaxVisited$)	5	
Peers visited during join procedure($JMaxVisited$)	$logM$	
Histogram-Related Parameters		
Number of buckets	100	
Domain of x	[0, 999]	
Tuples per peer	1000	
Range of queries	0, 10	

data distribution we use, 70% of the tuples of each peer belong to one bucket, and the rest are uniformly distributed among the rest of the buckets. The tuples in each bucket also follow the uniform distribution. The input parameters are summarized in Table 6.1.

6.1.1 Small-World Construction

We study the properties of the network created using our small-world construction procedure. In particular, we evaluate the diameter and the clustering of the network. For these experiments, we assume two query workloads with ranges 0 and 10 each (whose results occupy 1 and 11 buckets correspondingly) to tune the weight for the workload-aware edit distance. We compare the constructed clustered network with a randomly constructed p2p system.

Diameter

In the first two experiments, we keep the size of the network fixed and vary the radius of the horizon from 1 to 3. We present the diameter of the network with respect to the radius of the horizon. We conducted the same experiment for query ranges 0 and 10 and number of short links 1 and 2 (Fig. 6.2 and 6.1). Using two short links the diameter decreases for all histogram distance metrics. Also, for two short links the diameter is below 10 for each value of the radius and for each distance metric, which means that it satisfies the small-world property of a small diameter (i.e. logarithmic order in the number of peers). We then vary the number of peers in the network from 500 to 1500. We use two short links and a radius equal to two. As shown in Fig. 6.3, the diameter of the network scales well when increasing the number of peers and

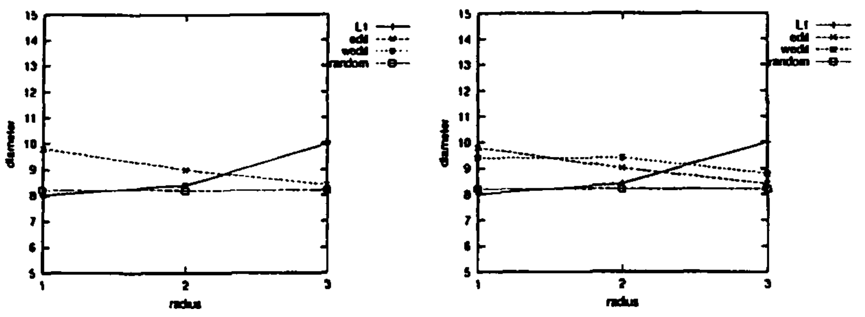

Figure 6.1: Influence of radius on diameter when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$

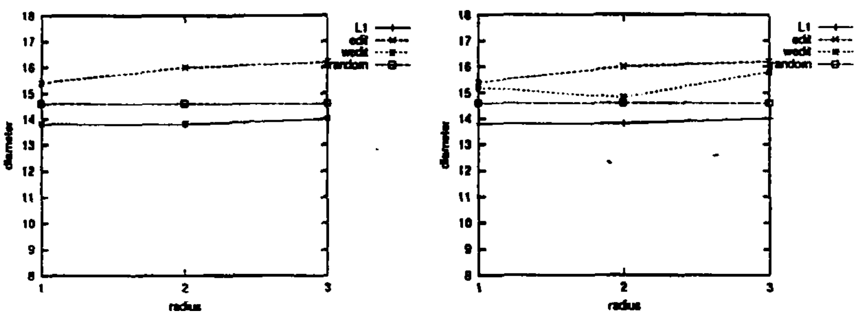

Figure 6.2: Influence of radius on diameter when *range* = 0 (left) and *range* = 10 (right), when $SL = 1$

remains logarithmic in order to the number of peers.

Quality of Clustering

In this set of experiments, we evaluate the quality of clustering. We measure the average histogram distance between the peers that are at various network distances from each other in the created p2p network. We use a fixed size network of 500 peers and radius 2, and conduct the same experiment for $SL = 1$ (Fig. 6.5 and 6.7) and $SL = 2$ (Fig. 6.4 and 6.6). The average histogram distance of peers with respect to their network distance is presented. As the network distance between two peers increases, their histogram distance increases for edit distance and for $SL = 2$ (Fig. 6.4 (center)), since this distance metric counts the number of permutations needed to make the histograms similar, and thus there will be an ordering of the peers based on the position of the buckets that summarize the largest number of tuples for each histogram. For $SL = 1$ (Fig. 6.5 (center)) this occurs until some network distance, since the use of only one short links results in week grouping of the relevant peers. This means that for $SL = 2$ the more similar two peers are, the closer in the network they are

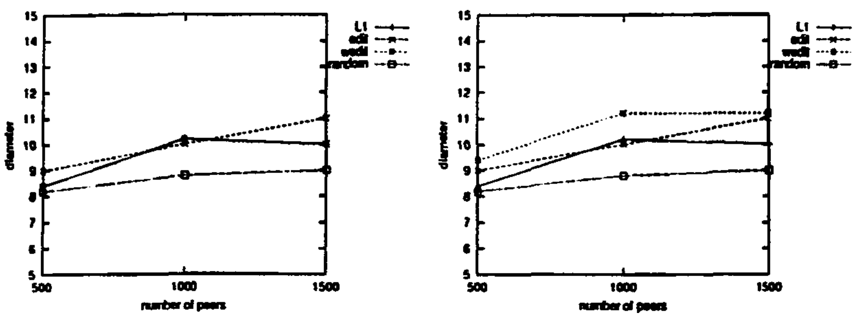

Figure 6.3: Influence of the number of peers on the diameter when *range* = 0 (left) and *range* = 10 (right) and $SL = 2$

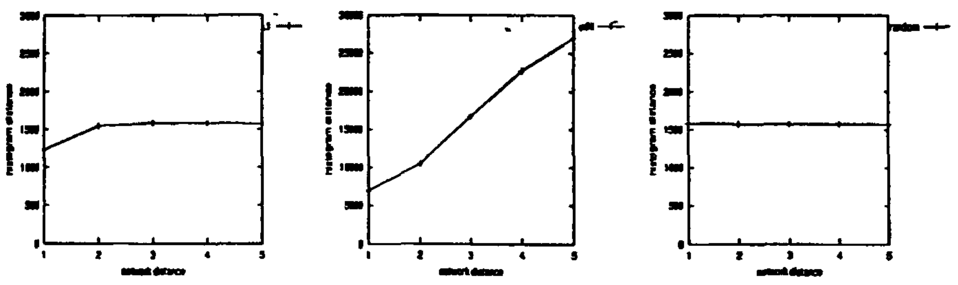

Figure 6.4: Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$

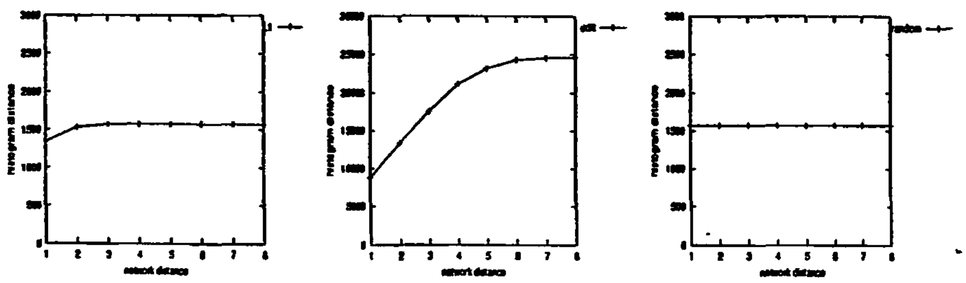

Figure 6.5: Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 1$

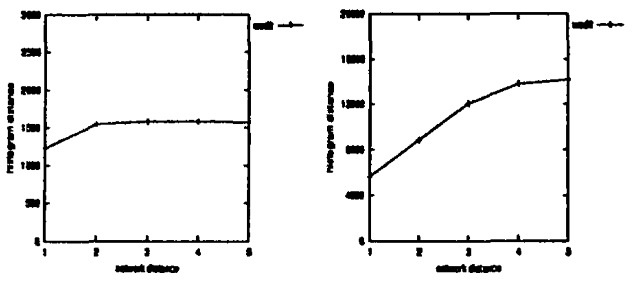

Figure 6.6: Clustering quality for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 2$

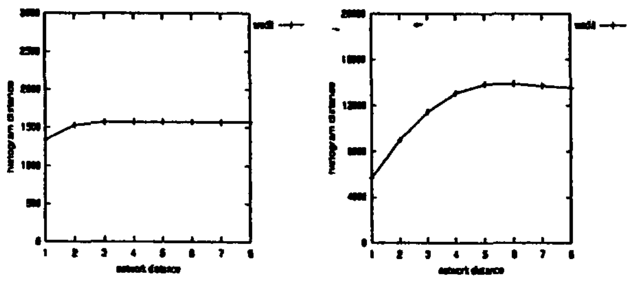

Figure 6.7: Clustering quality for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 1$

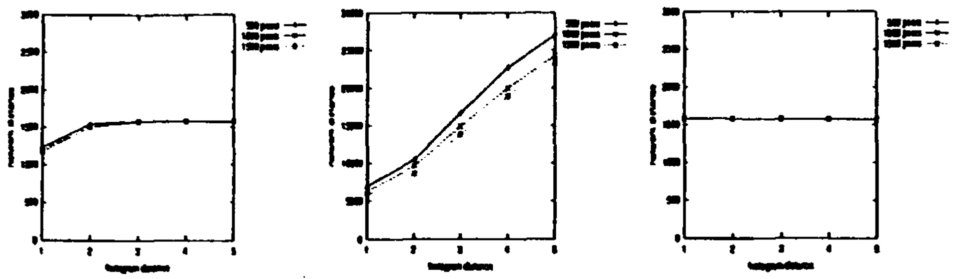
Figure 6.8: Clustering quality for different number of peers for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$

located.

For L_1 distance (Fig. 6.5 (left) and 6.4 (left)) this applies only for the neighboring peers (peers at distance 1), thus only peers that have the 70% of their tuples within the same bucket are grouped together. For networks distances greater than one, the histogram distances are nearly the same, since this distance metric does not take into account the ordering of the buckets.

The workload edit distance depends on the range of the queries. For range 0 (Fig. 6.6 (left) and 6.7(left)), the clustering quality is similar to L_1, since there is no need of ordering the peers based on the buckets positions. We are only interested in clustering together peers the have the 70% of their tuples within the same buckets. For range 10 and 2 short links (Fig. 6.6 (right) 6.7 (right)), histogram distance increases as the network distance increases, since we are interested in this ordering (peers that have the large number of tuples within buckets that are nearby, are more likely to answer the same query with range 10). For 1 short link this happens until some point due to the weak grouping of the peers.

For the random network (Fig. 6.4 (right) and 6.5 (right)), the histogram distance is constant for all network distances, since there is no grouping of similar peers.

Next we show the clustering quality as network scales (Fig. 6.8). We vary the number of peers in the network from 500 to 1500 and we present the clustering quality for each with respect to the size of the network. We use two short links and a radius equal to two. We notice that the clustering remains unaffected as the number of peers increases, which means that *logM* number of visited peers when a peer joins the system are enough to achieve good grouping of the peers as network scales.

6.1.2 Query Routing

In this set of experiments, we evaluate the performance of query routing using *PeerRecall* as our performance measure (as defined in Def. 2). We compare the constructed clustered network with a randomly constructed p2p network,

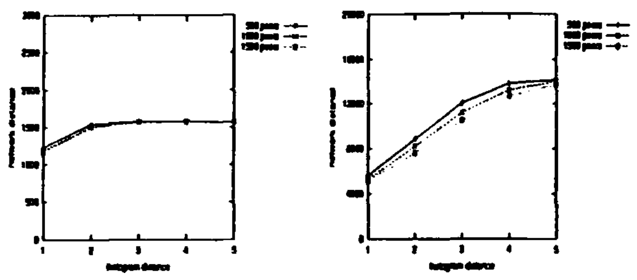
Figure 6.9: Clustering quality for different number of peers for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 2$

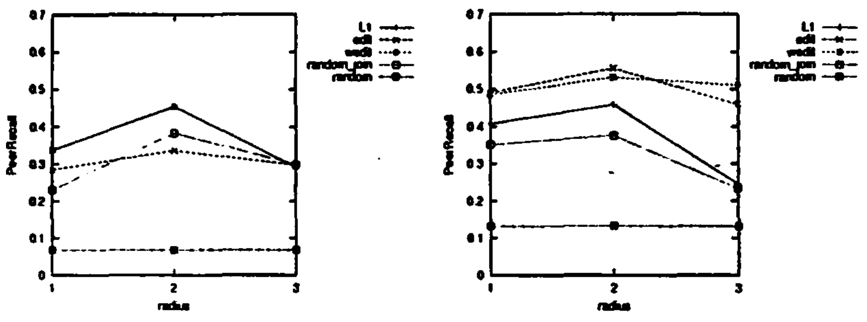

Figure 6.10: Performance of routing for different values of the radius when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$

that is a p2p system in which each new peer connects randomly to existing peers (random construction and routing) (*random*). We also consider a randomly constructed p2p system that uses histograms for query routing only (*random_join*).

We use a network of 500 peers and examine the influence of the horizon in the query routing performance for both 1 and 2 short links and for queries with range 0 (Fig. 6.11(left) and 6.10(left)) and 10 (6.11(right) and 6.10(right)). The radius of the horizon varies from 1 to 3. *PerrRecall* with respect to the radius is presented. Using histograms for both construction and query routing results in much better performance than using histograms only for routing or not using histograms at all. Also using 2 short links results is much better performance, since peers in the same group have more links between them and it is easier to retrieve all the results. For radius 2 and for 2 short links, we achieve the best performance. *PeerRecall* decreases for radius greater than 2 (when $SL = 2$) for all distance metrics. The reason is that many peers are included within the horizon of a particular peer and thus, a very large number of peers correspond to each routing index. This results in losing much more information than using radius 2, due to the inaccuracy in the summarized information of the routing indexes. For 1 short link, the performance increases even for radius 3, since there

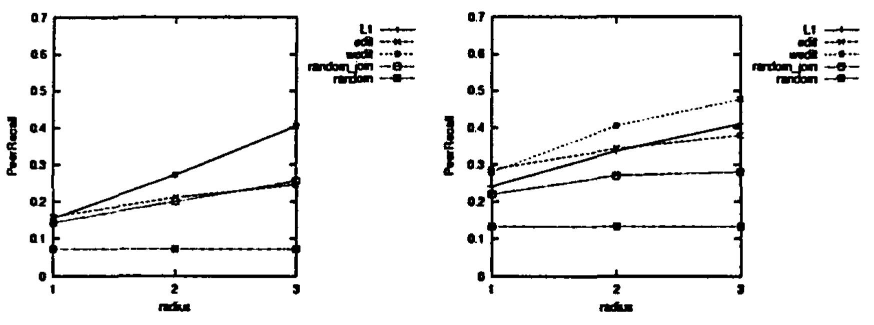

Figure 6.11: Performance of routing for different values of the radius when *range* = 0 (left) and *range* = 10 (right), when $SL = 1$

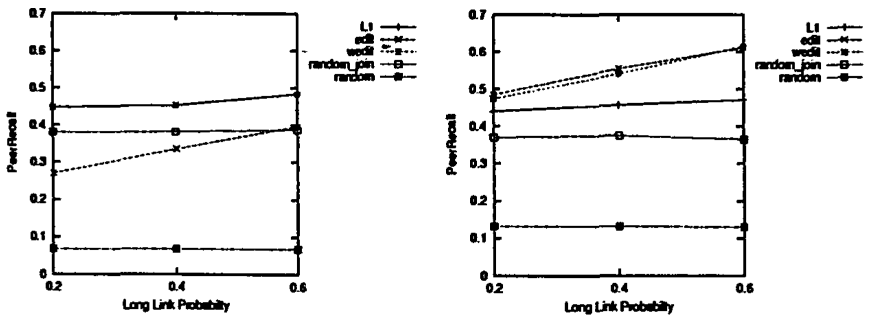

Figure 6.12: Performance of routing for different values of P_l when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$

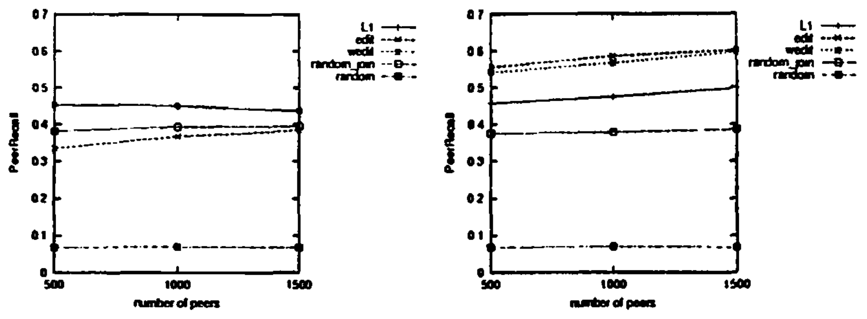

Figure 6.13: PeerRecall when varying the number of peers when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$

are less links and the number of peers that are included within the horizon of a particular peer is much smaller and there is less loss of information.

For queries of range 0, the performance for L_1 and workload-aware edit distance is exactly the same, since for this range these distance metrics are the same. Edit distance performs much worst, since for range 0 the difference in the number of results between two histograms is independent from their distance (as shown in section 4.2 Fig. 4.3(center))

For queries of range 10, edit and workload-aware edit distances have nearly similar performance, since both of them make ordering of the peers based on the positions of the buckets with large number of results. L_1 is much worst, since it does not make this ordering, which is very effective for ranges larger than 0 (peers that have the large number of tuples within buckets that are nearby, are more likely to answer the same query if the query range is large enough).

Next, we examine how our algorithms perform with a larger number of peers. We vary the size of the network from 500 to 1500. Radius is set to 2. We use 2 short links and queries with range 0 and 10 (Fig. 6.13). *PeerRecall* increases a little as the number of peers increase, for all histogram distance metrics. When the network is initially created and the first peers join the system, we cannot achieve the best structure, since each new peer has few choices about the peer it will attach. But as the network scales, the join procedure can achieve the best performance since the network has the expected structure, allowing the new peers to make the best choices for the peers to attach. This is the reason for which the performance increases a little as network scales. This experiment shows also that *logM* visited peers during the join procedure are enough for the network to scale.

We also vary the probability of creating a long link for each peer that joins the system between 0.2 and 0.6, and present *PeerRecall* for each of these probabilities. As this probability increases, the performance increases a little (Fig. 6.12) for the three distance metrics, since it is easier to navigate among the groups and find more relevant to the query peers. But even for a few long links the performance is very good.

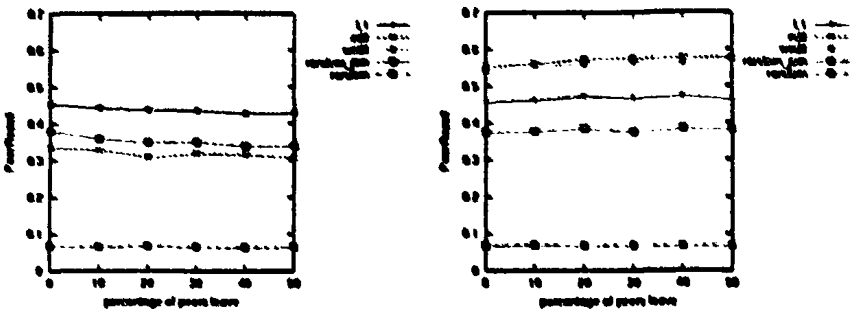

Figure 6.14: PeerRecall as peers leave the system when *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$

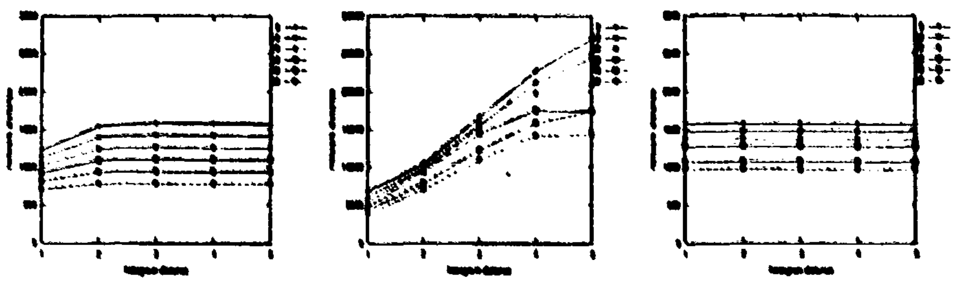

Figure 6.15: Clustering quality as peers leave the system for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$ and $R = 2$

6.1.3 Peers Leaving the Network

We also run the same set of experiments to study the behavior of the network as peers leave the system. In the ideal case we want all peers to have the same fanout after the load balancing procedure. After the network of 500 peers has been built, a percentage of the peers leave the system (varying from 10% to 50%). The radius of the horizon is set to two and $SL = 2$. We present the *PeerRecall* (Fig. 6.14) and the quality of clustering (Fig. 6.15 and 6.16) with respect to the percentage of peers leave, for query ranges 0 and 10. We notice that the performance and the clustering quality remains nearly unaffected. Thus, the update procedure followed when peers leave the system (section 3.3.4) preserves the properties of the network unaffected and keeps the network connected.

6.1.4 Distribution of Links

In this set of experiments, we build the p2p system incrementally to show how the links are distributed among old and new peers. We apply the load balancing procedure discussed in section 3.3.3. Figure 6.17 presents the distribution of the fanout of the peers in the system before the load balancing procedure, and after

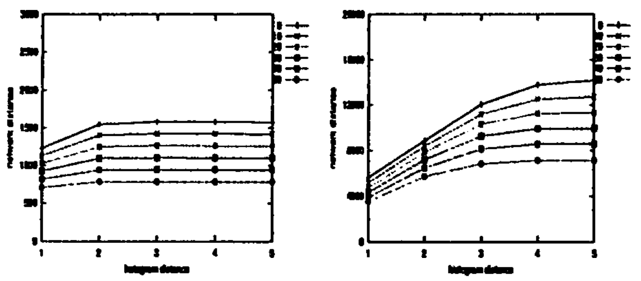
Figure 6.16: Clustering quality as peers leave the system for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 2$ and $R = 2$

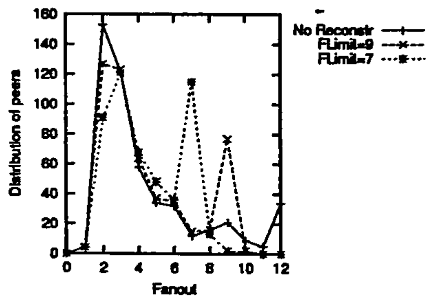

Figure 6.17: Distribution of the links

the procedure (for two different values of $FLimit$). In all cases, each peer may have at most 12 links. We notice that after the load balancing procedure the fanout of the peers is distributed mainly among two different regions. More specifically, many peers have $FLimit$ links and many other have about 2 and 3 links. This is explained as follows: before the load balancing procedure takes place, most of the peers have about 2 or 3 links. After the load balancing procedure is completed, the fanout of the peers is distributed mainly among two different regions. One region is the same as before the procedure takes place, and the other is the region around $FLimit$, since all the peers for which the fanout was above $FLimit$ will have new fanout equal to $FLimit$, and also, peers with fanout a little less than $FLimit$ are more likely to be requested to accept new neighbors. By reducing further the value of $FLimit$, these two regions will reach each other. In the ideal case, there will be only one region of high concentration, where nearly all the peers will have fanout around $FLimit/2$.

6.1.5 Summary of Experiments Using Simple Histograms

To conclude, the network constructed with the proposed procedure is a small-world network. The diameter remains of logarithmic order to the number of peers and peers with relevant data are grouped together such as the network distance between two peers to correspond also to the difference in the number of results they maintain. Also, only $logM$ number of peers need to be visited during the join procedure in order the network to satisfy these properties, which means that it scales very well as the number of peers increases. The performance is much better when using histograms than in the *random* or in the *random_join* network. Finally, workload-aware edit distance performs very well for all the query workloads, instead of the L_1 that works well only when the query includes one bucket, and the edit distance that performs well only with queries of larger ranges.

6.2 Multi-Attribute Histograms

Each peer stores a relation that includes 1000 tuples with two integer attributes $x, y \in [0, 99]$ with 1000 tuples. The tuples of the two attributes are summarized by a multi-attribute histogram with 10 buckets for each level (attribute). Consider a two-dimensional space, each dimension representing the tuple values on one of the attributes (Fig. 5.3). Summarizing the tuples based on a two-attribute histogram, each tuple corresponds to a square in the two-dimensional space based on the values of the attributes x and y, as shown in figure 5.3. In the data distribution we use, the 70% of the tuples of each peer belong to one square, and the rest are uniformly distributed among the rest of the squares. The tuples in each square also follow the uniform distribution. The input parameters are summarized in Table 6.2.

Table 6.2: Input parameters (Multi-Attribute Histograms)

Parameter	Default Value	Range
Number of peers M	500	500-1500
Radius of the horizon	2	1-3
Number of short links (SL)	2	1-2
Probability of long link (P_l)	0.4	0.2-0.6
Number of peers visited during join ($JMaxVisited$)	$logM$	
Perc of peers visited during routing ($MaxVisited$)	5	
Histogram-Related Parameters		
Number of buckets per attribute	10	
Domain of x	[0, 99]	
Domain of y	[0, 99]	
Tuples per peer	1000	
Range of queries	0, 10	

6.2.1 Small-World Construction

We study the properties of the network built using our small-world construction procedure, evaluating the diameter and the clustering of the network. We assume two query workloads with ranges 0 and 3 for each attribute (whose results occupy 1 and 16 squares correspondingly) to tune the weight for the workload-aware edit distance. We compare the constructed clustered network with a randomly constructed p2p system.

Diameter

In the first two experiments, we keep the size of the network fixed and vary the radius of the horizon from 1 to 3. We conducted the same experiment for $SL = 1$ (Fig. 6.19) and $SL = 2$ (Fig. 6.18). When using two short links the diameter decreases for all types of histogram distances. Also, for two short links the diameter is below 10, for each value of the radius and for each distance metric, which means that it satisfies the small-world property of a small diameter (i.e., a diameter of logarithmic order in the number of peers). We then vary the number of peers in the network from 500 to 1500. We use 2 short links and a radius equal to 2. As shown in Fig. 6.20, the diameter of the network scales well when increasing the number of peers and remains of logarithmic order to the number of peers.

Quality of Clustering

In this set of experiments, we evaluate the quality of clustering. Similarly to the experiments for simple histograms, we measure the average histogram distance

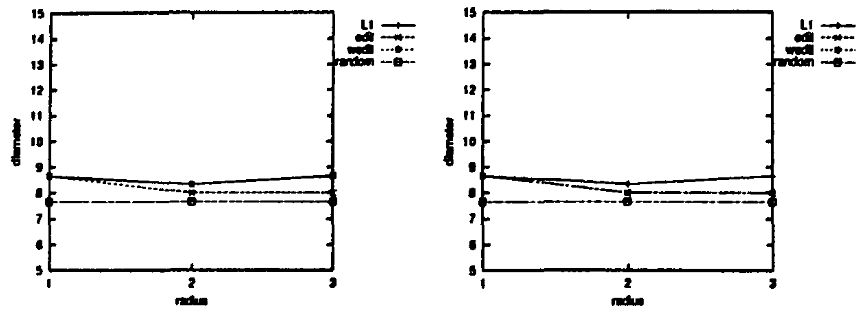

Figure 6.18: Influence of radius on diameter when *range* = 0 (left) and *range* = 3 (right), when $SL = 2$

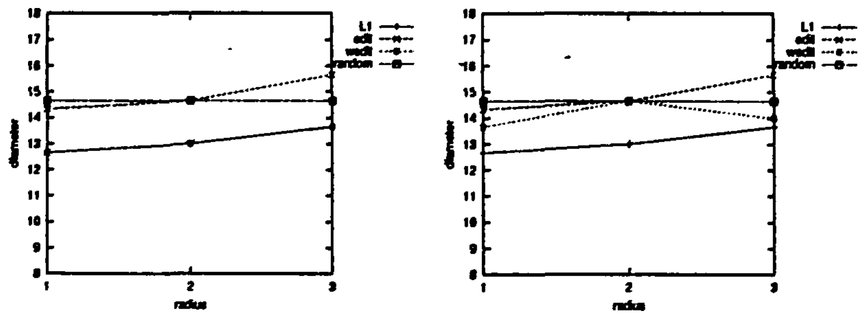

Figure 6.19: Influence of radius on diameter when *range* = 0 (left) and *range* = 3 (right), when $SL = 1$

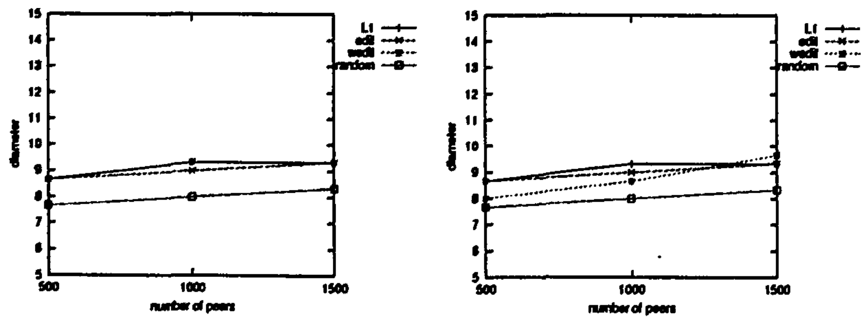

Figure 6.20: Influence of the number of peers on the diameter when *range* = 0 (left) and *range* = 3 (right) and $SL = 2$

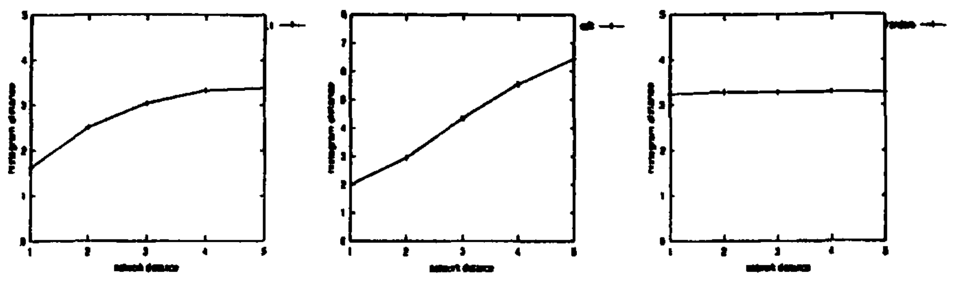

Figure 6.21: Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$

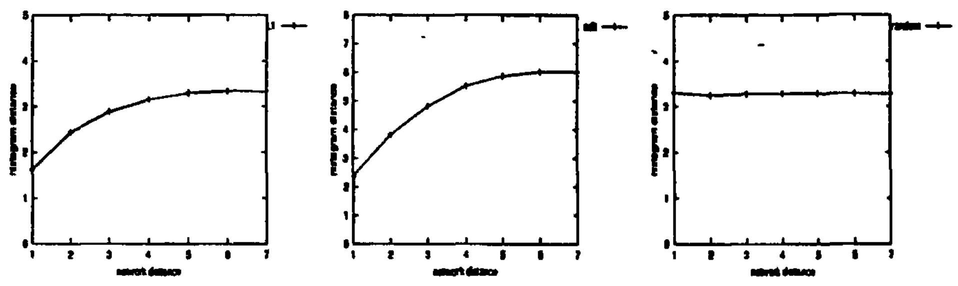

Figure 6.22: Clustering quality for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 1$

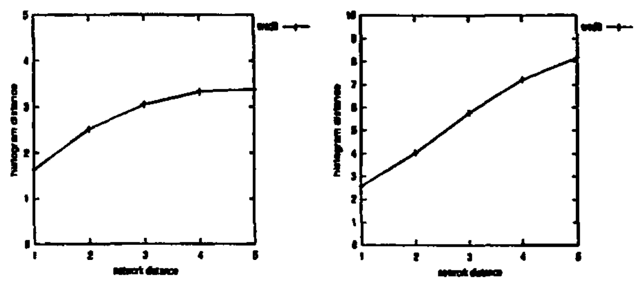

Figure 6.23: Clustering quality for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 3 (right), when $SL = 2$

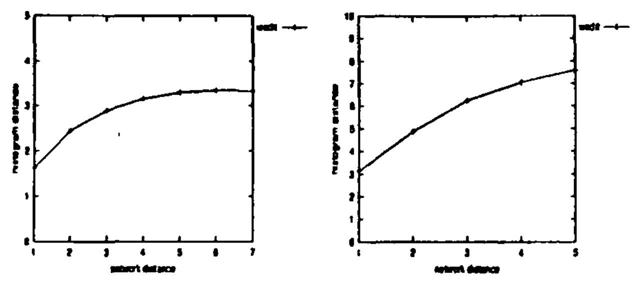

Figure 6.24: Clustering quality for networks built using workload-aware edit distance for *range* = 0 (left) and *range* = 10 (right), when $SL = 1$

between the peers that are at various network distances from each other in the constructed overlay network. We use a fixed size network of 500 peers and radius 2, and conduct the same experiment for $SL = 1$ (Fig. 6.22 and 6.24) and $SL = 2$ (Fig. 6.21 and 6.23). The results are similar to the simple histograms. For edit distance, as the network distance between two peers increases, their histogram distance increases too. When $SL = 2$ (Fig. 6.21 (center)) this is more clear than using 1 short link only (Fig. 6.22 (center)), since with one short link we have week grouping of the relevant peers.

For L_1 distance (Fig. 6.22 (left) and 6.21 (left)) after some network distance the histogram distances are nearly the same, since this distance metric does not take into account the ordering of the buckets.

The workload edit distance depends on the range of the queries. For range 0 (Fig. 6.23 (left) and 6.24(left)), the clustering quality is similar to L_1, since there is no need of ordering the peers based on the buckets positions. We are only interested in clustering together peers the have the 70% of their tuples within the same buckets for each attribute. For range 3 (Fig. 6.23 6.24), histogram distance increases as the network distance increases, since we are interested in this ordering (peers that have the large number of tuples within squares that are nearby, are more likely to answer the same query with range 3).

For the random network (Fig. 6.21 (right) and 6.22 (right)), the histogram distance is constant for all network distances, since there is no grouping of similar peers.

Next we show the clustering quality as network scales (Fig. 6.25). We vary the number of peers in the network from 500 to 1500. We use two short links and a radius equal to two. Also for multi-attribute histograms, we notice that the clustering remains unaffected as the number of peers increases, which means that *logM* number of visited peers when a peer joins the system are enough to achieve good grouping of the peers as network scales.

6.2.2 Query Routing

Similarly to the simple histograms, we measure *PeerRecall* to evaluate the performance of query routing. We compare the constructed clustered network

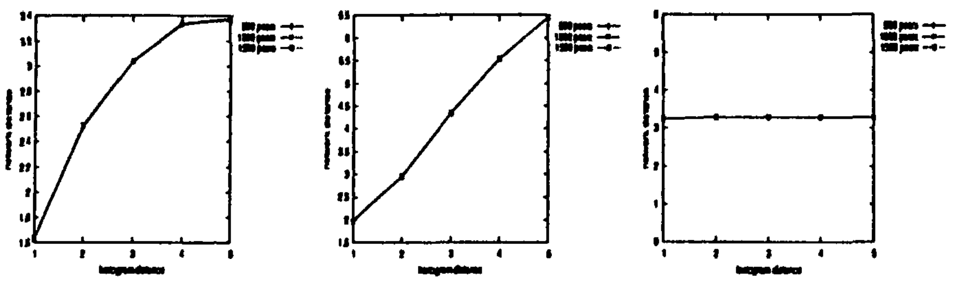

Figure 6.25: Clustering quality for different number of peers for networks built using L_1 (left) and edit (center) distance and for the *random* network (right), when $SL = 2$

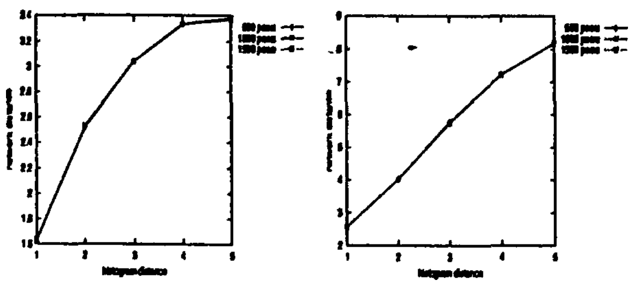

Figure 6.26: Clustering quality for different number of peers for networks built using workload-aware edit distance for $range = 0$ (left) and $range = 3$ (right), when $SL = 2$

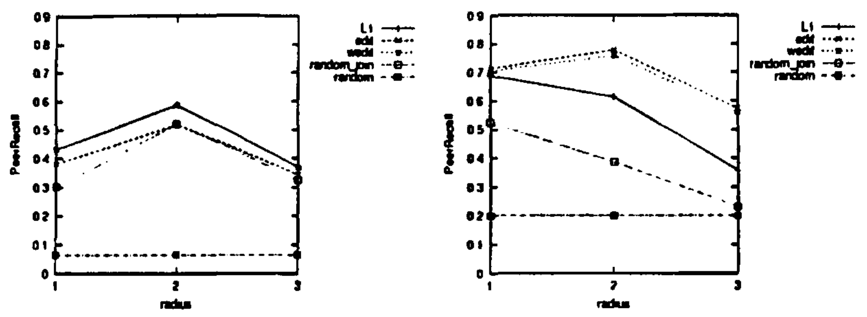

Figure 6.27: Performance of routing for different values of the radius when *range* = 0 (left) and *range* = 3 (right), when $SL = 2$

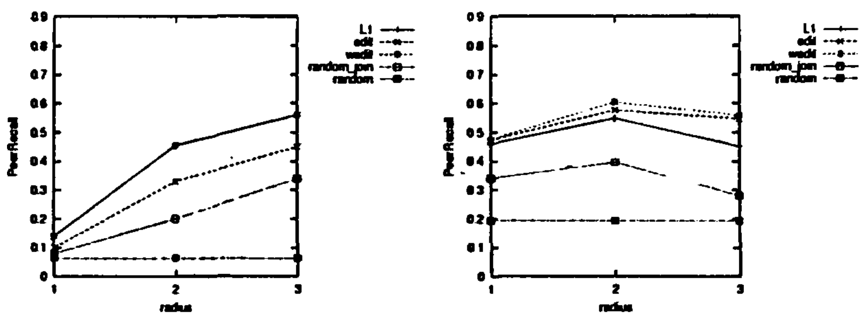

Figure 6.28: Performance of routing for different values of the radius when *range* = 0 (left) and *range* = 3 (right), when $SL = 1$

with a randomly constructed p2p, that is a p2p system in which each new peer connects randomly to existing peers (random construction and routing) (*random*). We also consider a randomly constructed p2p system that uses histograms for query routing only (*random_join*).

We use a network of 500 peers and examine the influence of the horizon in the query routing performance for query ranges 0 and 3 and for $SL = 1$ (Fig. 6.28) and $SL = 2$ (Fig. 6.27). The radius varies from 1 to 3.

The results are similar to the simple histograms. Using histograms for both construction and query routing results in much better performance than using histograms only for routing or not using histograms at all. For 2 short the network performs better, due to the stronger grouping of similar peers. Also, the best performance is achieved for radius equal to two (when $SL = 2$), except from L_1 distance. The reason is that since we deal with more attributes, there is larger relevance between the data stored by each peer. For example, two peers may have many tuples with similar values for the one particular attribute and different values for the other. Thus, except from the peers that have most of their tuples within the same square, there are also peers that have most of the tuples within squares of the same row or the same column in the two-dimensional

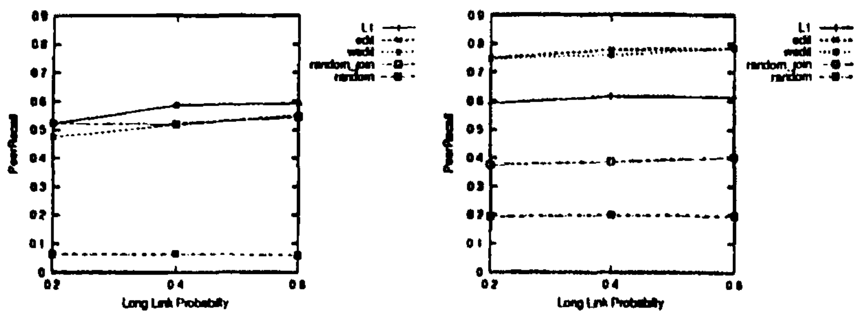

Figure 6.29: Performance of routing for different values of P_l when *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ and $R = 2$

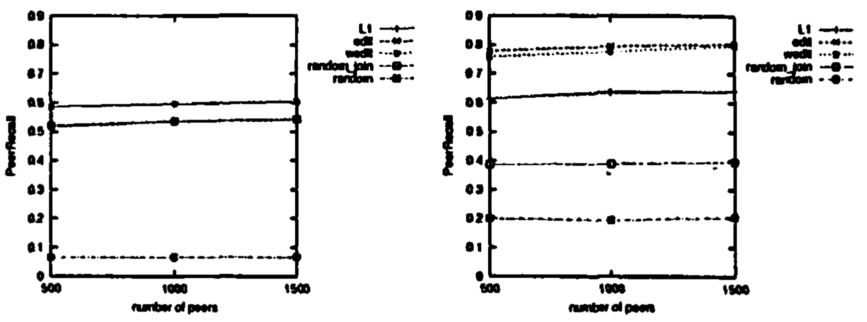

Figure 6.30: PeerRecall when varying the number of peers when *range* $= 0$ (left) and *range* $= 3$ (right), when $SL = 2$ and $R = 2$

space. That means that there are more peers that can give many results for each query, and thus, smaller value of radius is required to locate them. Using radius greater than two decreases the performance for all distance metrics due to the great loss of information. For one short link, the performance drops for radius 3 (in contrast to the simple histograms) for the same reason.

For queries of range 0, the performance for L_1 and workload-aware edit distance is exactly the same, since for this range these distance metrics are the same. Edit distance has much worst performance since for range 0 the difference in the number of results between two histograms is independent from their distance.

For queries of range 3, edit and workload-aware edit distances have nearly similar performance, since both of them make ordering of the peers based on the positions of the buckets with large number of results. L_1 is much worst, since it does not make this ordering, and thus, peers that have the large number of tuples within squares that are nearby, will not be nearby in the overlay network).

Next, we examine how our algorithms scales. We vary the size of the network from 500 to 1500. Radius is set to two. We use two short links and queries with range 0 and 3. As shown in Fig. 6.30, the network scales well and the performance increases a little as the number of peers increase, for all histogram distance metrics, due to the improvement in the network structure as more peers join the system.

Finally, we vary the probability of creating a long link for each peer that joins the system between 0.2 and 0.6 (Fig. 6.29). The performance is a little improved as the number of long links increases for all the distance metrics, since we can find the appropriate groups with less hops.

6.2.3 Summary of Experiments Using Multi-Attribute Histograms

To conclude, we notice that in general the behavior of the network is nearly the same with simple histograms for all the distance metrics. Thus, the multi-attribute histograms and the distance metrics used perform well also for range queries over multiple attributes. When extending each distance metric (L_1, edit and workload-aware edit) to the multi-attribute histograms, it inherits the properties of the distance metric over simple histograms.

Chapter 7

Conclusions and Future Work

In this thesis, we have proposed the use of histograms as routing indexes in peer-to-peer systems. We proposed building a small-world network in a fully decentralized procedure, where each peer uses only information stores locally. Routing indexes are stored for each link of a peer, summarizing the content of the peers within the horizon, reachable through this link. These indexes are used in order to route a query or join message, through the link that we expect to find more results for the query or to find peers that store data similar to the new peer correspondingly.

We proposed using histograms as routing indexes, that are appropriate for answering range queries. In order to construct a small-world network based on the content of the peers, the distance between the histograms, that represent the content of different peers, should be calculated (peers with small distance will be grouped together). The criterion based on which the peers will be grouped together is the number of results they return for each query. Thus we introduced a workload-aware distance metric that takes into account the query workload in the calculation of the distance between two histograms. For this distance metric, it is shown that for a particular workload the distance between two histograms, in the average case, is analogous to the difference in the number of results they store. Thus, peers that are more likely to answer the same set of queries will be grouped together.

A similar approach for building and querying a small-world network was first presented in [22]. Bloom filters are used as routing indexes, and exact-match queries are posed in the network. In [10], also path queries over XML documents are studied using Multi-Level Bloom Filters as indexes and horizons for the query routing.

We have extended our system to support also range queries over multiple attributes, by introducing a multi-attribute histogram. We also extended the distance metrics used for simple histograms to support also the multi-attribute histograms.

We compared experimentally workload-aware distance metric with L_1 and edit distance. The results show that our distance metric performs much better (the histogram distance increases analogous to the difference in the number of results).

Our experimental results on the construction of the network and the routing of the queries have shown that our small-world construction procedure is effective, since in the constructed peer-to-peer system peers with small histogram distance have also small network distance. Routing is also very efficient, since using histograms increases the number of results returned for a given number of peers visited. Furthermore, the network scales very well when increasing the number of peers, since only $logM$ number of peers need to be visited (during the join procedure) as the number M of peers increases, in order to achieve the same performance and to leave the network properties unaffected. Finally, our workload-aware edit distance is very effective for the performance of the network. Instead of L_1 and edit distance metrics, when workload-aware distance is used, the network performs well for all the query ranges.

This work is a first step towards leveraging the power of histograms in peer-to-peer systems. There are many issues that need further investigation.

So far we used equi-width histograms. An interesting issue would be to use other types of histograms, for example equi-depth histograms. In this occasion there are several things to be studied. New distance metrics should be introduced that are appropriate for the new type of histograms. Also new procedures need to be defined for constructing the routing indexes by aggregating local indexes and updating them.

An issue that need to be studied further is finding the appropriate number of buckets thus achieving the best performance. In this work we assume that this number is predefined and does not change. Based on the content of the peers and the query workload, there may exist a more appropriate number that leads to a more accurate representation of the data by the histograms. For example, if a particular range of values is queried more, it may be effective to distribute the tuples within this range into more buckets, in order to achieve less loss of information.

We can extend our work on the histograms by using them in structured p2p systems, where the location of an object in the DHT overlay depends on its histogram (that can be seen as a vector of values). Objects with small histogram distance will be placed at peers that correspond to regions nearby in the $n-$dimensional space.

Another interesting issue is how to select the peer to be linked through long link with the new peer during the join procedure. We currently select this peer randomly from the list of the visited peers, but more sophisticated methods can be used. These methods may take into account the distances between the histogram of the new peer and each of the visited peers and select one of them with a probability that depends on this histogram distance. This way we can affect the topology of the constructed network.

Bibliography

[1] Gnutella. http://www.gnutella.com.

[2] Kazaa. http://www.kazaa.com.

[3] Napster. http://www.napster.com.

[4] F. Banaei-Kashani and C. Shahabi. SWAM: A Family of Access Methods for Similarity-Search in Peer-to-Peer Data Networks. In *CIKM*, Washington, USA, 2004.

[5] F. Banaei-Kashani and C. Shahaby. Searchable Querical Data Networks. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing in conjunction with VLDB'03*, Berlin, Germany, 2003.

[6] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search Enhanced by Topic Segmentation. In *26th ACM SIGIR*, p 306-313, Toronto, Canada, 2003.

[7] S-H Cha and S. N. Sribari. On Measuring the Distance Between Histograms. *Patern Recognition*, 35:1355–1370, 2002.

[8] A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, 2002. Submitted for publication.

[9] A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *22nd ICDCS*, Vienna, Austria, 2002.

[10] Y. Petrakis G. Koloniari and E. Pitoura. Content-Based Overlay Networks of XML Peers Based on Multi-Level Bloom Filters. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, Berlin, Germany, 2003.

[11] A Guttman. R-Trees: A Dynannc Index Structure for Spatial Searching. In *ACM SIGMOD*, p 47-57, Boston, USA, 1984.

[12] R. Morris I. Stoica, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE/ACM Trans. on Networking*, 11(1):17–32, 2003.

[13] A. Iamnitchi, M. Ripeanu, and I. T. Foster. Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations. In *1st IPTPS*, p 232-241, Campridge, MA, USA, 2002.

[14] Y. Ioannidis. The History of Histograms. In *29th VLDB*, p 19-30, Berlin, Germany, 2003.

[15] J. Kleinburg. The Small-World Phenomenon: An Algorithm Perspective. In *thirty-second annual ACM symposium in Theory of computing*, Atlanta, USA, 1999.

[16] W. Lee M. Li and A. Sivasubramaniam. Constructing a Semantic Small World in P2P Systems. In *Proceedings of the International Conference on Network Protocols (to appear)*, 2004.

[17] S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed Hashing in a Small World. In *4th USENIX Symposium on Internet Technologies and Systems*, San Antonio, Texas, USA, 2003.

[18] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, 2002.

[19] A. Mohan and V. Kalogeraki. Speculative Routing and Update Propagation: A Kundali Centric Approach. In *ICC*, Alaska, USA, 2003.

[20] S. Chaudhuri N. Bruno and L. Gravano. STHoles: A Multidimensional Workload-Aware Histogram. In *ACM SIGMOD*, California, USA, 2001.

[21] B. Yang P. Ganesan and H. Garcia-Molina. One Torus to Rule them All: Multi-dimensional Queries in P2P Systems. In *7th international workshop on The web and databases*, Paris, France, 2004.

[22] Y. Petrakis and E. Pitoura. On Constructing Small Worlds in Unstructured Peer-to-Peer Systems. In *International Workshop on Peer-to-Peer Computing and DataBases*, Heraklion, Greece, 2004.

[23] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *VLDB*, p 486-495, Athens, Greece, 1997.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *SIGCOMM*, p 161-172, San Diego, CA, USA, 2001.

[25] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. A Peer-to-peer Framework for Caching Range Queries. In *20th ICDE*, Boston, USA, 2004.

[26] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *12th HPDC*, p 226-235, Washington, USA, 2003.

[27] C. Tang, Z. Xu, and S. Dwarkadâs. Peer-to-Peer Information retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM*, p 175-186, Karlsruhe, Germany, 2003.

[28] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. In *1st CIDR*, Asilomar, CA, 2003.

[29] D. J. Watts and S. H. Strogatz. Collective Dynamics of Small-World Networks. *Nature*, 393:440–442, 1998.

Appendix

False positive probability for a histogram. Let $H(n)$ be a histogram for an integer attribute $x \in [Dmin, Dmax]$ (x can take $d = Dmax - Dmin + 1$ distinct values). $H(n)$ has b buckets. Let a query $x = A$. We assume that each peer has t tuples that follow uniform distribution. Then in each bucket we have t/b tuples. The probability that we do not have a query match, that is, there does not exist a tuple with value $x = A$ in the data summarized by $H(n)$ is $P(query\_no\_match) = ((d-1)/d)^n$. The probability that the histogram indicates a match is: $P(hist\_match) = 1 - ((b-1)/b)^n$ (it is sufficient that one tuple falls into the bucket that A falls into as well). The range of each bucket is d/b. Thus the probability of having a $query\_no\_match$ while we had a histogram match is: $P_1 = ((d/b - 1)/(d/b))^{t/b} = ((d-b)/d)^{t/b}$. Thus, the false positive probability is according to the formula of Bayes:
$P(fp) = P(hist\_match \; / \; query\_no\_match) = P_1 * P(hist\_match)/P(query\_no\_match)$
$\Rightarrow P(fp) = (((d-b)/d)^{t/b} * (1 - ((b-1)/b)^t))/((d-1)/d)^t$.

Equivalence of the two workload histogram distances. The two workload-aware distances 6 and 5 are equivalent, that is $wd_{L_1}(H(n_1), H(n_2)) = wd_e(H(n_1), H(n_2))$.

Proof.

$wd_e(H(n_1), H(n_2)) = \Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj}|pref(j+k) - pref(j-1)| =$
$\Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj}|\Sigma_{i=0}^{j+k}(H_i(n_1) - H_i(n_2)) - \Sigma_{i=0}^{j-1}(H_i(n_1) - H_i(n_2))| =$
$\Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj}|\Sigma_{i=0}^{j-1}(H_i(n_1) - H_i(n_2) - H_(n_1) + H_i(n_2)) + \Sigma_{i=j}^{j+k}(H_i(n_1) - H_i(n_2))| =$
$\Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj}|\Sigma_{i=j}^{j+k}(H_i(n_1) - H_i(n_2))| = \Sigma_{k=0}^{b-1}\Sigma_{j=0}^{b-1} w_{kj}|\Sigma_{i=j}^{j+k}(H_i(n_1) - H_i(n_2))| =$
$wd_{L_1}(H(n_1), H(n_2))$.

Proof that the histogram distances are metrics. We show next that the workload-aware L_1 measure is a metric, by proving that it satisfies the metric properties (reflexivity, non-negativity, commutativity and the triangle inequality).

- reflexivity: $wd_{L_1}(H(n_1), H(n_1)) = 0$
  $L_1(H_i(n_1), H_i(n_1)) = 0, \forall i, 0 \le i \le b-1 \Rightarrow wd_{L_1}(H(n_1), H(n_1)) = 0$.
- non-negativity: $wd_{L_1}(H(n_1), H(n_2)) \ge 0$.
  Since $0 \le w_k$ and wd_{L_1} is the sum of absolute values, the property holds.
- commutativity: $wd_{L_1}(H(n_1), H(n_2)) = wd_{L_1}(H(n_2), H(n_1))$
  $wd_{L_1}(H(n_1), H(n_2)) = \Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k}(H_{i+j}(n_1) - H_{i+j}(n_2))| =$
  $\Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k} H_{i+j}(n_1) - \Sigma_{j=0}^{k} H_{i+j}(n_2)| =$

$\Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k} H_{i+j}(n_2) - \Sigma_{j=0}^{k} H_{i+j}(n_1)| =$
$wd_{L_1}(H(n_2), H(n_1))$

- triangle ineq.: $wd_{L_1}(H(n_1), H(n_3)) \leq wd_{L_1}(H(n_1), H(n_2)) + wd_{L_1}(H(n_2), H(n_3))$.
  $wd_{L_1}(H(n_1), H(n_3)) = \Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k}(H_{i+j}(n_1) - H_{i+j}(n_3))| =$
  $\Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k}(H_{i+j}(n_1) - \Sigma_{j=0}^{k} H_{i+j}(n_2) + \Sigma_{j=0}^{k} H_{i+j}(n_2) - \Sigma_{j=0}^{k} H_{i+j}(n_3))|$
  $\leq \Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k}(H_{i+j}(n_1) - H_{i+j}(n_2)| + \Sigma_{k=0}^{b-1}\Sigma_{i=0}^{b-1} w_{kj}|\Sigma_{j=0}^{k}(H_{i+j}(n_2) -$
  $H_{i+j}(n_3)| =$
  $wd_{L_1}(H(n_1), H(n_2)) + wd_{L_1}(H(n_2), H(n_3))$.

Since wd_e and wd_{L_1} are equivalent, wd_e is also a metric and the metric properties can be proved similarly to the respective properties of wd_{L_1}.

For two histograms $H(n_1)$, $H(n_2)$ and a query q_{kj} the difference in the number of results returned by the peers n_1 and n_2 is: $hdiffer(n_1, n_2, qkj) = \Sigma_{i=j}^{j+k}|L1(i)|$

Proof.

$\Sigma_{i=j}^{j+k}|L1(i)| = \Sigma_{i=j}^{j+k}|H_i(n_1) - H_i(n_2)| = hdiffer(n_1, n_2, qkj)$

For two histograms $H(n_1)$, $H(n_2)$ and a query q_{kj} the difference in the number of results returned by the peers n_1 and n_2 is: $hdiffer(n_1, n_2, qkj) = |pref(j+k) - pref(j-1)|$

Proof.

$|pref(j+k) - pref(j-1)| = |\Sigma_{i=0}^{j+k}(H_i(n_1) - H_i(n_2))$-$\Sigma_{i=0}^{j-1}(H_i(n_1) - H_i(n_2))| =$
$|\Sigma_{i=j}^{j+k}(H_i(n_1) - H_i(n_2))| = hdiffer(n_1, n_2, qkj)$

Proof that the multi-attribute histogram distance is metric. We show next that the multi-attribute histogram distance measure (Equation 5.2) is a metric, by proving that it satisfies the metric properties (reflexivity, non-negativity, commutativity and the triangle inequality). We assume that the distance measure d used is a metric.

- reflexivity: $D(H(n_1), H(n_1)) = 0$
  $D(H(n_1), H(n_1)) = \Sigma_{i=1}^{attr}\Sigma_{j=1}^{b_{i-1}^{i-1}} d(H_{ij}(n_1), H_{ij}(n_1))/b_{i-1}^{i-1} = 0 \Rightarrow$
  $D(H(n_1), H(n_1)) = 0$.
- non-negativity: $D(H(n_1), H(n_2)) \geq 0$.
  $d(H_{ij}(n_1), H_{ij}(n_2)) \geq 0, \forall 1 \leq i \leq attrand 1 \leq j \leq b_{i-1} \Rightarrow D(H(n_1), H(n_2)) \geq 0$
- commutativity: $D(H(n_1), H(n_2)) = D(H(n_2), H(n_1))$
  $D(H(n_1), H(n_2)) = \Sigma_{i=1}^{attr}\Sigma_{j=1}^{b_{i-1}^{i-1}} d(H_{ij}(n_1), H_{ij}(n_2))/b_{i-1}^{i-1} =$
  $\Sigma_{i=1}^{attr}\Sigma_{j=1}^{b_{i-1}^{i-1}} d(H_{ij}(n_2), H_{ij}(n_1))/b_{i-1}^{i-1} =$
  $D(H(n_2), H(n_1))$
- triangle ineq.: $D(H(n_1), H(n_3)) \leq D(H(n_1), H(n_2)) + D(H(n_2), H(n_3))$.

$$D(H(n_1), H(n_3)) = \Sigma_{i=1}^{attr} \Sigma_{j=1}^{b_{i-1}^{i-1}} d(H_{ij}(n_1), H_{ij}(n_3))/b_{i-1}^{i-1} \leq$$
$$\Sigma_{i=1}^{attr} \Sigma_{j=1}^{b_{i-1}^{i-1}} (d(H_{ij}(n_1), H_{ij}(n_2)) + d(H_{ij}(n_1), H_{ij}(n_2)))/b_{i-1}^{i-1} =$$
$$D(H(n_1), H(n_2)) + D(H(n_2), H(n_3))$$

Thus D is a distance metric.