Efficient Query Answering in Peer Data Management Systems

DISSERTATION

zur Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
Dipl.-Ing. Armin Roth

Gutachter:
1. Prof. Dr. Ulf Leser
2. Prof. Dr. Felix Naumann
3. Prof. Dr. Kai-Uwe Sattler

eingereicht am: 22.11.2010
Tag der mündlichen Prüfung: 01.04.2011

This work is dedicated to my parents and
to all humans which intelligently employ information systems
to make the world a better place.

Abstract

Peer data management systems (PDMS) consist of a highly dynamic set of autonomous, heterogeneous peers connected with schema mappings. Queries submitted at a peer are answered with data residing at that peer and by passing the queries to neighboring peers. PDMS are the most general architecture for distributed integrated information systems. With no need for central coordination, PDMS are highly flexible. However, due to the typical massive redundancy in mapping paths, PDMS tend to be very inefficient in computing the complete query result as the number of peers increases. Additionally, information loss is cumulated along mapping paths due to selections and projections in the mappings.

Users usually accept concessions on the completeness of query answers in large-scale data sharing settings. Our approach turns completeness into an optimization goal and thus trades off benefit and cost of query answering. To this end, we propose several strategies that guide peers in their decision to which neighbors rewritten queries should be sent. In effect, the peers prune mappings that are expected to contribute few data. We propose a query optimization strategy that limits resource consumption and show that it can drastically increase efficiency while still yielding satisfying completeness of the query result.

To estimate the potential data contribution of mappings, we adopted self-tuning histograms for cardinality estimation. We developed techniques that ensure sufficient query feedback to adapt these statistics to massive changes in a PDMS. Additionally, histograms can serve to maintain statistics on data overlap between alternative mapping paths. Building on them, redundant query processing is reduced by avoiding overlapping areas of the multi-dimensional data space.

Zusammenfassung

Peer-Daten-Management-Systeme (PDMS) bestehen aus einer hochdynamischen Menge heterogener, autonomer Peers. Die Peers beantworten Anfragen einerseits gegen lokal gespeicherte Daten und reichen sie andererseits nach einer Umschreibung anhand von Schema-Mappings an benachbarte Peers weiter.

Insofern sind PDMS die allgemeinste Architektur für verteilte, integrierte Informationssysteme. Solche aufgrund fehlender zentraler Komponenten eigentlich hochflexiblen Systeme leiden bei zunehmender Anzahl von Peers unter erheblichen Effizienzproblemen. Die Gründe hierfür liegen in der massiven Redundanz der Pfade im Netzwerk der Peers und im Informationsverlust aufgrund von Projektionen entlang von Mapping-Pfaden.

Anwender akzeptieren in hochskalierten Umgebungen zum Datenaustausch in vielen Anwendungsszenarien Konzessionen an die Vollständigkeit der Anfrageergebnisse. Unser Ansatz sieht in der Vollständigkeit ein Optimierungsziel und verfolgt einen Kompromiß zwischen Nutzen und Kosten der Anfragebearbeitung. Hierzu schlagen wir mehrere Strategien für Peers vor, um zu entscheiden, an welche Nachbar-Peers Anfragen weitergeleitet werden. Peers schließen dabei Mappings von der Anfragebearbeitung aus, von denen sie ein geringes Verhältnis von Ergebnisgröße zu Kosten, also geringe Effizienz erwarten.

Als Basis dieser Schätzungen wenden wir selbstadaptive Histogramme über die Ergebniskardinalität an und weisen nach, daß diese in dieser hochdynamischen Umgebung ausreichende Genauigkeit aufweisen. Wir schlagen einen Kompromiß zwischen der Nutzung von Anfrageergebnissen zur Anpassung dieser Metadaten-Statistiken und der Beschneidung von Anfrageplänen vor, um den entsprechenden Zielkonflikt aufzulösen. Für eine Optimierungsstrategie, die das für die Anfragebearbeitung verwendete Zeit-Budget limitiert, untersuchen wir mehrere Varianten hinsichtlich des Effizienzsteigerungspotentials. Darüber hinaus nutzen wir mehrdimensionale Histogramme über die Überlappung zweier Datenquellen zur gezielten Verminderung der Redundanz in der Anfragebearbeitung.

Contents

Part I.

Large-scale Data Sharing using PDMS

1. Introduction

Interconnecting computers both in enterprises and the Internet created the opportunity to widen the horizon of humans by exploiting widespread information. To achieve added value by describing complex facts, information from many sources can be combined. Usually, the interests of consumers of integrated information differ to a certain extent from those of the information providers. Consumers desire to quickly access comprehensible and stable data sources returning correct and complete results. The owners of the sources focus on high autonomy in the type, amount, and structure of data they offer. And finally, the owners of data sources often want to flexibly decide about which parties they directly cooperate with.

The main objective of integrating information from distributed information sources is to gain a view on the world that is as complete as possible. So the more sources can be combined to form a query answer, the higher is the potential completeness of the results with respect to the information need of the user. However, scaling up the number of contributing sources faces some challenges. In many scenarios, the number and heterogeneity of data sources makes it difficult to establish a *central* facility to access all the data. All stakeholders would have to agree to a common structuring of the data. As new sources enter the system, this global agreement possibly has to be revised.

These organizational obstacles as well as the requirements of autonomy of the data providers are addressed by a decentrally structured architecture for information integration. *Peer data management systems* (PDMS) are the most general architecture for such a purpose. Recently, many works related to PDMS emerged in the literature, e.g. [ACMH03, CGL$^+$05, FKLS03, HIST03, HLS06, LNWS03, RN05]. PDMS consist of an arbitrary network of peers that act both as data storage and as a mediator. The peers answer queries using own data and by forwarding the queries to their direct neighbors. The network of peers is made up of schema mappings that bridge the heterogeneous conceptual structures of the peers, Fig. 1.1. The schema mappings between peers are used to reformulate queries before passing them to a another peer. Please note that we assume that each peer only knows about its direct neighbors. Consequently, every peer performs query planning and data processing in isolation.

In PDMS, peers are highly autonomous, e.g., they can select any set of peers to which they establish schema mappings or they can expose only a part of their data. Local changes can happen independently from the rest of the system. Due to this decentral nature, PDMS show high flexibility. This is why this architecture is well suited for ad hoc integration scenarios such as disaster data management or virtual enterprises. In such situations, many systems have to be integrated in a short time which does not allow figuring out a centralized data architecture. A PDMS can be rapidly established without a central authority or time-consuming negotiations between all stakeholders.

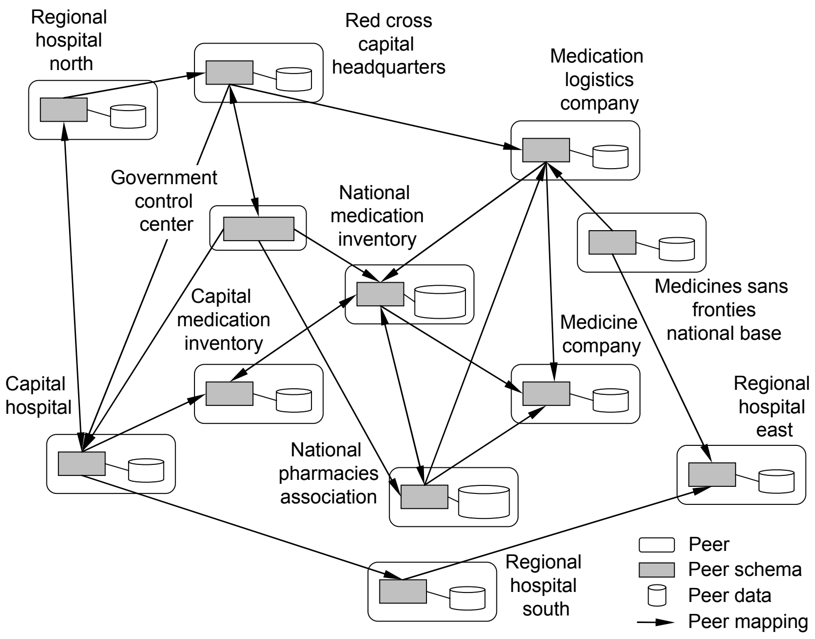

Regional hospital north
Red cross capital headquarters
Medication logistics company
Government control center
National medication inventory
Capital medication inventory
Medicine company
Medicines sans fronties national base
Capital hospital
Regional hospital east
National pharmacies association
Regional hospital south
Peer
Peer schema
Peer data
Peer mapping

Figure 1.1.: Example of a PDMS for medication logistics during a nationwide disaster.

However, the advantages of flexibility and high peer autonomy come at a cost. In general, a certain data item in a PDMS can be reached from a particular peer along *several* mapping paths. Hence, the same data might be transported back to the querying peer on redundant mapping paths which unnecessarily consumes time.

If the mappings between peers relate only a subset of the attributes available at one end to the other, information is lost that possibly has been transported between several peers before. Additionally, selections in mapping paths can reduce the amount of data returned by mapping paths compared to redundant paths. The obstacles of high redundancy and information loss hamper PDMS to scale up to a large number of peers although this would be possible from a conceptual perspective of schemas and mappings.

Despite of these drawbacks which will be adressed in this thesis, PDMS are suitable in several application scenarios:

– *Disaster data management* supports the recovery from a regional, nationwide (Fig. 1.1) or even more widespread crisis like the 2004 Asian tsunami or the 2005 hurricane Katrina [NR06]. It aims at rapidly integrating diverse and heterogenous data sources that were not meant to cooperate before. Due to lack of time and the high complexity, an appropriate solution for information sharing has to be built and scaled up on-the-fly. In such cases, the number of data sources is very high and the resulting network of peers usually evolves over time. Therefore, the architecure of PDMS are very well suited in this application field [NR06]. However, computing and transporting *all* result data is often not possible due to the exponential nature of the PDMS query answering problem. Additionally, in such situations a distributed information sharing system often faces low network bandwidth between the peers as well as long mapping paths due to damaged connections. Therefore, it is crucial to optimize the consumption of computing and network ressources in PDMS resulting in a best-effort approach returning incomplete query answers.

– *Fusions* between companies or building virtual organizations usually involve distributed and integrated information systems already being in place at the partners. A complete redesign of the overall data management is often impossible, because daily business has to continue. So the new organization is forced to quickly connect the existing system landscapes, which in general results in an irregular network of peers, i.e., a PDMS, which then can evolve to a target state. As large enterprises often have hundreds of heterogeneous information systems in place, obtaining the full set of query answers by exploiting all of the redundant mapping paths here also can be impossible due to limited computing ressources or the requirement of quick response time.

– *Sharing of scientific data*, for instance in astronomy or life sciences, is often characterized by a large and dynamic set of stakeholders that have specific interests. Here, the amount of data being queried is both huge and heterogeneous. Again, this work shows how PDMS can meet these requirements if efficiency of query answering is drastically increased compared to retrieving all query answers.

– *Personal information management* means that each employee in an organization structures the information on his or her's computer desktop in a personal way. These islands of personal information can be leaveraged by interconnecting them [ANR07] and providing overall query answering services. Due to the high number of participants and high volatility in the system, a distributed and flexible architecture of a PDMS is ideally suited for that purpose. Because of the system size and limitations in computing resources, efficiency is an important issue here, too.

– The *Semantic Web* is the most obvious application scenario for distributed query optimization [HIMT03]. Since the semantics of conceptual models and query answering can be captured much better in a decentral fashion and since the amount of data is huge, efficient query processing should also be organized in a distributed manner. This can be realized appropriately by a PDMS architecture. Furthermore, in the Web as large-scale information sharing environment best-effort approaches as discussed in this work are usually satisfying for the users.

For PDMS being scaled up, overcoming inefficient query answering due to the high redundancy in the network of peers is crucial. To preserve the important advantage of high flexibility, each peer should retain high autonomy.

This thesis contributes a novel solution to increase efficiency of PDMS query answering with a decentral optimization approach estimating result size and overlap between alternative mapping paths. This is comparably more difficult in PDMS than in centralized DBMS, because the former underlies higher volatility that massively impacts the maintenance of completeness-related metadata. Based on this information, each peer independently decides on pruning its local search space leading to best-effort optimization approach.

Our solution for efficient query answering in Pdms lets users access relevant query answers in environments that remain flexible and thus makes large-scale information sharing feasible.

1.1. Challenges for Flexible Large-scale Data Sharing

Pdms are the generalization of mediator-based or federated information systems and promise to be both more flexible and better scalable. However, compared to these centralized approaches to data integration, Pdms face two main difficulties in query answering: first redundancy both in mapping paths and data stored at the different peers and second massive changes in the data distribution as perceived at the peers as a consequence of changes in the peer network.

Redundancy in query answering. A user query issued at a peer is answered based on data stored locally at the peer and based on the results of queries posed to the immediate neighbor peers. These in turn perform the same procedure. So as a result, queries are reformulated and passed through the system recursively. Clearly, as a whole this is a process of exponential runtime complexity.

Since the topology of the peer graph can be arbitrary, different mapping paths starting at the same peer can end at another same peer. This means that data from the end of the mapping path is transported redundantly along both mapping paths. Consequently, the peer at the beginning of these alternative mapping paths possibly faces *overlapping* query results, i.e., tuples returned by one mapping path also are in the result of the other mapping path. Additionally, data about a particular real-world entity can be stored at several peers in the system. So even without any redundant mapping paths, there can exist data overlap from the perspective of a particular peer.

Another source of inefficiency is information loss *along* mapping paths. If an attribute of a relation at one mapping end is not present at the other end, the mapping contains a projection. Due to these projections attribute values can be eliminated from intermediate results. Then, those attribute values have unnecessarily been transported between several peers before.

The high degree of redundancy together with considerable branching as query processing advances through the peer graph as well as information loss give rise to massive runtime problems in Pdms query answering. Even Pdms instances with tens of peers tend to be intractable when the query answer is required to be complete with respect to the data stored at *all* peers.

Users often are not interested in all results of a query, possibly because they cannot inspect all of them in detail or they do not want to apply aggregation functions. So for our applications, we can assume that users are satisfied with *incomplete* query answers. This means that to make query answering in large-scale Pdms feasible, we can turn to a best-effort approach. This thesis will show that allowing query answers that are "sub-complete", i.e., that achieve at least 90 percent completeness, incur drastically less

cost than fully complete query results.
Flexibility in the system. From the perspective of a particular peer, an important advantage of the PDMS architecture is that changes can happen decentrally. A peer is free to leave or join the network at any time and to establish and remove mappings to other peers. For scenarios involving PDMS we can assume that there happen hundreds of queries between two changes in the system, because new peers have to find appropriate neighbors and build schema mappings to them. Since the topology of the peer graph is not constrained, even a single peer can massively influence the structure of the PDMS. If a peer represents a bottleneck in the peer graph, the data accessible from many other peers in the system is highly sensitive on the behavior of that particular peer. If it goes offline, the peer graph falls apart. As a consequence, all peers in the one part cannot reach peers in the other part any more; so the data distribution as perceived from the local perspective of the peers drastically changes. Observe that this mechanism can influence the available data much more than in central databases or mediator-based integrated information systems. Therefore, this thesis puts special emphasis on this issue.

1.2. Preserving Peer Autonomy

Being autonomous is an important requirement for data owners in practice. Often, the profit for the owner is independent of the access rates to their data. Rather, the main motivation for maintaining their data store is often given by their own business goals. For instance, to support enterprise-wide intelligence, the owner of the peers have the obligation to provide their data to others. However, acting with high autonomy often saves cost for peer administrators concerning following issues:

– *Ensuring operation without interruption.* Guaranteeing continious operation of a service can be much more cost intensive than allowing small interruptions from time to time. This is why peers desire to go offline from time to time, often without notifying its data consumers.

– *Changes in the schema of the data.* This can break the mappings neighboring peers have established. So they are forced to repeat the time-consuming and error prone procedure of creating schema mappings. Note that if the peer would cooperate closer with its neighbors, it would notice them about its schema changes and they possibly can adapt their mappings with less effort. But this notification would decrease its autonomy.

– *Changes in the data offering.* As peers in a PDMS return data they have in turn received from other peers, changing the set of neighbors can heavily influence the amount of data a peer returns for a certain query. Principally, peers want to be free to decide from where they get the data they return as query answers.

– *Service offering.* Each peer desires to decide on its own about the language of the query interface publicly exposed. To leave as much autonomy as possible at the peer, we assume that only conjunctive queries are supported. It will be shown in this thesis how peers can assess the amount of data available at their neighbors without being notified about updates.

Taken together, the simpler the assumptions on the query and maintenance services offered by the peers, the easier it is to extend the system with high autonomy for the peers.

1.3. Problem Definition

This thesis addresses the problem of efficiently answering queries with a satisfying size of the query answer in a volatile peer data management system. Given a network of autonomous and heterogeneous peers acting as data integration systems and given a query workload, we develop techniques to (i) *locally* gather metadata to identify promising peers, (ii) select a good combination of peers for a plan following a cost constraint, (iii) efficiently retrieve results for the plan. We divide this problem in several subproblems:

Description of semantic relationships between peers. The data exchanged by different peers must be related by describing their semantic correspondences. Thus translation of user queries between the peers is enabled. The language of the schema mappings needs to be expressive enough to capture differences between the peer's relational schemas. However, the mapping language also should be simple to meet even very restricted query capabilities of practical peers. Respecting their autonomy in what query interface they offer is a very important requirement in practice.

Constructing a query plan. A query plan describes how to obtain the query answer based on the preliminary answers returned from the neighbors. This plan should return all possible result tuples, independently from how many attributes carry `NULL` values. An incomplete result is always better than no result. A mechanism to determine a query plan has to deal with the specifications of semantic relationships between the peers.

Estimation of query answer size. For trading off query answer size and resource consumption at query planning time, the size of the query answer must be estimated. These assessments need to be accurate enough to distinguish peers promising many data from those potentially returning only few resulting tuples.

Maintaining metadata for size of query answers. To enable query answer size estimation, statistical metadata has to be collected. Changes in the topology of the peer graph can have major impact on the accuracy of these statistics. So the metadata have to adapt quickly to changes in the system.

Assessment of overlap between peers. Redundant mapping paths in the PDMS lead to massive overlap between subquery answers returned by alternative mapping paths. This overlap should be assessed in such a way that the information can be used to optimize efficiency of query answering as described below.

Effective query planning. To answer user queries, building on cardinality estimates on answers from neighboring peers a query plan has to be determined. The plan should optimally exploit the schema mappings to all neighboring peers. Moreover, it can be necessary that the resulting plan obeys a cost constraint. The creation of an plan maximizing the completeness of its answer should be efficient, because otherwise PDMS cannot scale up well.

Optimized retrieval of answers. Query plans can be further optimized by eliminating data overlap between answers from alternative mapping paths to a maximum extent. But this optimization should preserve the completeness of those query answers. Additionally, query planning decisions should be revised if they turn out to be inappropriate during query execution. Then, query planning has to be adapted.

1.4. Thesis Outline

The main contribution of this thesis is a completeness-oriented optimization of queries using adaptive statistics in a peer data management system. Part I introduces query answering in PDMS. Part II presents metadata to be employed for the query optimization approaches proposed in Part III. In the following, we provide an outline for the thesis that summarizes each chapter. Related work is discussed within the chapters.

Part I – Large-scale Data Sharing using PDMS

Chapter 1: Introduction. This chapter has motivated the problem of efficiently answering queries with a satisfying result size in a large, volatile PDMS. Due to many alternative mapping paths, query answering in PDMS is highly inefficient and tends to be intractable. Additionally, we claim that preserving the autonomy of peers is an important practical requirement. Therefore, approaches to prune the search space that are solely based on local reasoning are crucial to make PDMS query answering feasible. We divided this problem into several subproblems that are addressed throughout this thesis. An overview on the thesis was published in [Rot07].

Chapter 2: Query Answering in PDMS. We introduce our completeness-driven approach to answer user queries. We describe the language for specifying semantic relationships between neighboring peers. Based on that, a two-phase query planning algorithm is presented. Information loss accumulates along mapping paths in PDMS and leads to unnecessary transport of data. We propose to address this problem by pruning locally at a peer and point out several difficulties when trading off completeness of results and cost.

Chapter 3: Humboldt Peers: Building a PDMS. *Humboldt Peers* is the name of our experimental testbed of which we give an overview in this chapter. The effectivity of our approaches depend on many parameters of PDMS instances. *Humboldt Peers* automatically creates such instances and measures a variety of parameters during the experiments. We briefly describe its functionality and its architecture in this chapter. Humboldt Peers also was presented in [RNHS06].

Part II – Completeness-related Metadata in PDMS

Chapter 4: Benefit and Cost of Query Answering. This chapter models the completeness of query results, data overlap between mapping paths, and query execution cost. We show that the completeness of results heavily depends on projections in the mapping paths, which induce information loss. For data overlap, we illustrate opportunities to save transportation cost and show how this problem interacts with the completeness of the query answer.

Chapter 5: Metadata Statistics. Result cardinalities and overlap need to be captured by multi-dimensional histograms. We compare different approaches for gathering and maintaining statistics in distributed information systems, and we propose to use query feedback for building statistics on result cardinalities and overlap. Therefore, we adapt a technique for self-tuning histograms to our volatile setting. The size of histograms is optimized in the context of our complex mapping language by choosing the minimal set of dimensions.

Part III – Efficient Query Answering in PDMS

Chapter 6: Completeness-Driven Query Planning. In this chapter it is shown how the cardinality statistics can be applied to prune non-promising mapping paths from further query processing. To this end, we describe how our completeness model is employed to valuate the utility of incomplete subplans at a peer. Pruning candidates are identified by regarding their potential data contribution in the context of a complete plan. This information is exploited in our threshold-based pruning approach, which was published in [RN05]. An extensive experimental study reveals that this simple, yet effective approach increases efficiency of query answering considerably in many situations.

Chapter 7: Maintaining Metadata Statistics. Since there can be massive changes in the data distribution across a PDMS our mechanism for gathering statistics must detect such events and adapt statistics accordingly. We show that even small changes in the peer graph topology can drastically influence data distributions as perceived locally at a peer. Pruning cuts off query feedback from our histograms. So we develop a technique to trade off between those two issues. We conduct several experiments to find out the necessary level of accuracy in cardinality estimation to achieve a satisfying increase in query answering efficiency.

Chapter 8: Query Optimization under Limited Resources. Scaling up PDMS requires effective query answering in presence of bounded resource consumption, which is covered by this chapter. We propose a budget-driven strategy to optimize the completeness of query answers. Several alternative algorithms for spending and refunding a budget along with queries as well as their results are presented. This approach was published in [RNHS06].

Chapter 9: Query Optimization using Information Overlap. We focus on further optimizing query plans by eliminating data overlap while preserving completeness of the results. Using overlap histograms, we identify areas with high overlap density within the multi-dimensional data distribution for each pair of peer mappings. This information is then exploited for rewriting query plans such that overlapping data is queried only once. This approach also makes use of the cardinality statistics, because it must preserve as much of the initial plan completeness as possible.

Chapter 10: Conclusion. This chapter finally summarizes and discusses the contributions of this thesis. We end by pointing to future research directions in the area of PDMS query answering, which in parts were published in [HHNR05] and [HRZ$^+$08].

2. Query Answering in PDMS

This chapter prepares the remainder of this work by introducing PDMS query answering in the complex context of two types of schema mappings for mediators, namely global-as-view and local-as-view. To this end, we first present our system model. Then, we examine how PDMS query planning can be extended to deal with projections in mappings between peers. They are a major source of incompleteness in such systems.

We explain how a first formulation of a query plan, the rule-goal tree, is transformed into an directly executable query plan made up by join- and union-like operators. This plan serves as input for our pruning techniques discussed throughout our work. The chapter closes by focussing on the trade-off between the completeness of the query result and the query processing cost.

2.1. Decentralized Data Sharing Using PDMS

Sharing information between a large amount of participants has to deal with several problems:

– Data sources are heterogeneous with respect to their syntax, structuring of data, and most importantly their semantics. A system for information sharing must relate these heterogeneous data sets with each other.

– There are frequent changes in the set of data sources and how they expose information to others. In practice, the stakeholders want to autonomously decide about their data sharing services.

– Users desire to pose queries against familiar interfaces that cover their domain of interests.

– Queries over semantically heterogeneous data sources should retrieve as many answers as possible to achieve a complete view of the world.

– Efficient query answering becomes increasingly difficult to achieve as the number of participants scales up.

In the research literature, these challenges have often been addressed by so-called integrated information systems [Nau02]. They provide a query interface with a unifying view to the user, which hides heterogeneity of the data sources. Such a system is directly connected with all data sources and thus must map between the unified view and all the sources. Due to its nature of central coordination, this approach does not scale well, especially in dynamic environments. The common representation of data, in which all the sources have to be mapped into, often makes it difficult to expose peculiarities of certain sources in the unified view. In other words, this uniform interface acts as a semantic bottleneck in accessing the variety of underlying data sources. If changes happen in the set of data sources, a possibly time-consuming process of adaption of the integration information system has to be conducted, possibly involving a large number of stakeholders.

Peer data management systems have a novel architecture to approach the challenges of large-scale information sharing listed above. They are a natural extension to integrated information systems. PDMS abolish the distinction of passive data sources and an active integrated information system. Rather, in a PDMS every participant can act as a data source or an integrated information system. This achieves much higher flexibility to adapt both to the information demand of users and an infrastructure that is already in place. And most prominently, the architecture of a PDMS proves to be well adaptive to changes among the participating units.

2.1.1. The Architecture of PDMS

Autonomy, heterogeneity, and distribution are the main dimensions to classify integrated information systems [OV11]. These dimensions are depicted in Fig. 2.1. DBMS, for instance, show neither heterogeneity, nor autonomy, nor distribution. Distributed DBMS, data warehouses, federated DBMS (FDBMS), and mediator-based information systems [Wie92, Les00] are characterized by higher degrees of distribution and heterogeneity or autonomy.

The peer-to-peer paradigm was first realized in so-called P2P systems [SW05]. They are heavily used in the Internet to share content, mostly in the granularity of files. Search requests in P2P systems usually are semantic-free [GHI^{+}01]. They refer objects that are specified by their identifier, which can consist of multiple attributes, thus enabling multi-dimensional range queries [SSR07]. Hence, in the classification in Fig. 2.1, P2P systems are characterized as homogeneous, since all peers share a common schema. Such systems are highly distributed and the peers have high autonomy in sharing their computing resources with other peers.

As can be seen in Fig. 2.1, PDMS extend P2P systems in the dimension of heterogeneity and thus are at the far end of all three dimensions. PDMS consist of a set of heterogeneous and autonomous peers, which usually are distributed in wide-area networks [NR07, HIST03]. Due to their nature, PDMS are the most general abstraction for any kind of integrated information system encountered in practice.

Each peer in a PDMS can act as a mediator or as a passive data source. As a mediator, a peer receives queries, reformulates them, and passes them to its neighboring peers. Rewriting of the queries is guided by schema mappings between the peers. To reflect practical situations, it is important that the topology of the network of peers and schema mappings can be arbitrary. However, this has an impact on decidability of query answering under certain semantics for a PDMS as is discussed in Sec. 2.4.

A PDMS is a set of peers, each of which may comprise local data sources and mappings both to the local sources and to other peers, Fig. 2.2. A single peer acts as a mediator

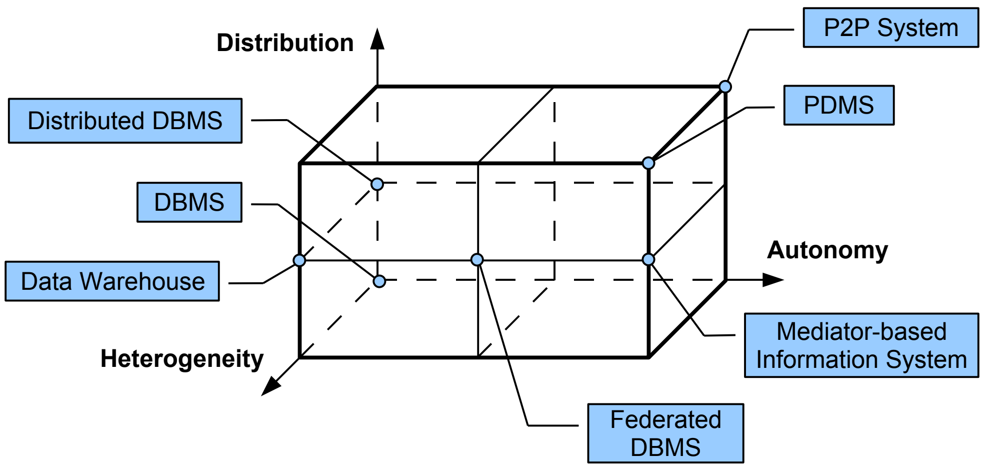


Figure 2.1.: Classification of integrated information systems over the dimensions autonomy, distribution, and heterogeneity [OV11, NL06].

consisting of a *peer schema* and possibly multiple local data sources. The peer schema describes data the peer offers. Local data sources are specified by *local schemas* and are connected to the peer schema by so-called *local mappings*. Peer schemas of neighboring peers are related by *peer mappings*. This is formalized in the following description:

Definition 1 (Peer Data Management System). *A PDMS is a set of peers* $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ *each of which is represented by a tuple* $P_i = (S, \mathcal{L}, \mathcal{M}_L, \mathcal{M}_P)$. *The peer schema* S *comprises a set of relations. To denote a relation* R *of the peer schema* S *at a peer* P *we write* $P.R$. *The set* $\mathcal{L}$ *consists of a relational schema for each of the local data sources of* P_i. *The set* $\mathcal{M}_L$ *of local schema mappings relates the schemas in* $\mathcal{L}$ *to the peer schema* S. *The set* $\mathcal{M}_P$ *of peer mappings maps* S *to the peer schemas of other peers.*

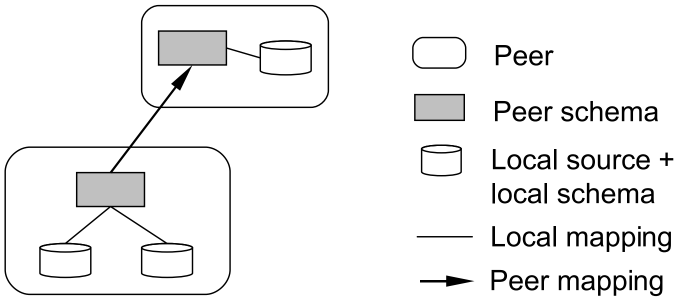


Figure 2.2.: Structure of a peer.

Similar to [CCGL04], we denote a foreign-key relationship between two relations R_1 and R_2 by $R_1[Key(R_1)] \supseteq R_2[f]$ with f being the foreign key attribute at R_2 pointing to the key $Key(R_1)$ of R_1.

Next, we describe the language for queries and mappings in our PDMS instances.

2.1.2. Queries and Mappings

In this work, we assume that peers accept a restricted version of the general form of conjunctive queries [Ull88]. Conjunctive queries (CQ) are the class of select-project-join queries without other operators, such as aggregation, grouping, or sorting. In the Datalog query language [Ull88, AHV95], a CQ has the following form

$$q(\bar{X}) :- s_1(\bar{X}_1), s_2(\bar{X}_2), \ldots, s_2(\bar{X}_n).$$

The atom $q(\bar{X})$ represents the answer relation and is called the *head* of the query. The predicates $s_i(\bar{X}_i)$ refer to relations of a underlying schema and are called query *subgoals*. Taken together, they make up the *body* of the query. The tuples $\bar{X}, \bar{X}_1, \bar{X}_2, \ldots, \bar{X}_n$ comprise variables and constants. The queries are required to be safe, i.e., $\bar{X} \subset \bar{X}_1 \cup \bar{X}_2 \cup \ldots \cup \bar{X}_n$. The means that to answer the query no information is needed other than that returned by the body of the query [PFPG02]. The variables in $\bar{X}$ are distinguished variables, all others are called existential. The complete set of variables of a query Q is denoted by $Vars(Q)$. In Datalog, a join between subgoals is expressed by the same variable occurring in these different subgoals.

Queries can also contain subgoals that are selection predicates. We consider semi-interval constraints of the form $v \otimes c$ made up by a variable v, a constant c, and a relation symbol $\otimes \in \{=, <, >, \leq, \geq\}$. We require that every variable v that occurs in a selection predicate also is included in any subgoal of the query. We refer to the set of selection predicates of a query Q by $Sel(Q)$.

High autonomy of peers among other criteria also refers to the query services they provide. In practice, projections in queries often are not possible when querying foreign peers. Hence, the result set comprises a minimal set of attributes, which for every result tuple have to be transferred from a remote peer. Therefore, we assume that the peers throughout this work accept only a restricted form of conjunctive queries that *excludes* projections. We denote this class of queries with $CQ_{\backslash P}$.

We extend the so-called Global-Local-as-View mappings, or GLaV mappings [Ull97, Len02] by projections:

Definition 2 (Extensionally Sound GLaV Mapping)**.** *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two distinct sets of peers. An extensionally sound GLaV mapping with projections is given by the logical formula*

$$\forall \bar{X} \forall \bar{Y} (Q_{\mathcal{P}_1}(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z} Q_{\mathcal{P}_2}(\bar{X}, \bar{Z})). \tag{2.1}$$

The queries $Q_{\mathcal{P}_1}$ and $Q_{\mathcal{P}_2}$ over the the peer sets are from $CQ_{\backslash P}$. The formula in Equation (2.1) is satisfied by database instances at $\mathcal{P}_1$ and $\mathcal{P}_2$ if

$$Q_1(\mathcal{P}_1) \subseteq Q_2(\mathcal{P}_2). \tag{2.2}$$

Intuitively, this definition means that any result tuple t of the query $Q_1(\mathcal{P}_1)$ is also an answer to the query $Q_2(\mathcal{P}_2)$, although t comprises different attributes at $\mathcal{P}_1$ than at $\mathcal{P}_2$. This means that in query processing this mapping can be used to rewrite queries over the peer set $\mathcal{P}_2$ to yield queries over $\mathcal{P}_1$, but *not* vice versa. Hence, these mappings are *directed* because of their inclusion dependency [AHV95]. In the following, we denote extensionally sound GLaV mappings with projections simply as GLaV mappings. Local mappings are of the same form as the peer mapping in Eqn. (2.2) except that Q_1 is defined over the schemas of a set of local sources.

Similarly to Datalog rules, we refer to the left side of the mapping formula

$$m : Q_h(\bar{X}) \subseteq Q_t(\bar{Y}) \tag{2.3}$$

as *head* and to the right side as *tail* and denote them by $Head(m)$ and $Tail(m)$, respectively. The set of variables $\bar{X}$ occurring in the head is called head variables and all others, $\bar{Y} \setminus \bar{X}$, appearing only in the tail are tail variables. For a more general introduction to mappings between domain models we refer the reader to [MBDH02].

In contrast to our approach, most important works on PDMS assume the result of the queries in the head and tail of a mapping being of the *same* arity [HIST03, CGLR04, Hos09, FKLS03]. We argue that projections in mappings are quite natural, since in practice the sets of attributes of different peers for the same real-world entity may be different [FHP$^+$02].

Please note that the restriction of the class of queries accepted by the peers to $CQ_{\setminus P}$ also applies to the individual queries occurring in peer mappings. This means that both the query in the head and the tail of a mapping are not allowed to contain projections with respect to the underlying peer schema. Rather, projections in mappings come into existence by the different *arities* of head and tail queries.

Building on the fact that our mappings are directed, we introduce the following notion for mappings that can be used for query rewriting between peers:

Definition 3 (Outgoing Mapping)**.** *Let P be a peer. A peer mapping m that contains any predicate over a relation of the peer schema of P in its tail, i.e., $P \in Tail(m)$, is called an outgoing mapping from P.*

Selection predicates in mappings express implicit knowledge about peer schemas. So in general, our GLaV mappings are of the form

$$Q_h(\bar{X}) \subseteq Q_t(\bar{Y}), Sel(\bar{Z}) \tag{2.4}$$

with $Sel(\bar{Z})$ being a conjunct of semi-interval constraints $v \otimes c$ made up by a variable v, a constant c, and a relation symbol $\otimes \in \{=, <, >, \leq, \geq\}$. The variables involved in selection predicates can be any variable of the query: $\bar{Z} \subseteq \bar{X} \cup \bar{Y}$. Observe that in particular there can be selections over variables that are *not* in the head.

For instance, in writing a mapping to a peer of a publisher *AW* and its relation *Book* the fact can be expressed that the peer models books of this publisher *only.* This is reflected in the following mapping from a peer of the online book shop *Amazon*:

$$\begin{gathered} AW.Book(Title, \underline{ISBN}) \subseteq \\ Amazon.Book(Title, \underline{ISBN}, Publisher), Publisher = 'AW'. \end{gathered}$$

As mentioned above, *projections* may appear in local and peer mappings, because different peers may comprise different attributes about a certain of real-world entity.

In Sec. 5.2, we discuss projections in the context of two important specializations of GLaV mappings, namely global-as-view (GaV) and local-as-view (LaV) mappings. To illustrate a simple case, consider a mapping from the *Book* relation of the publisher *AW*, which includes the *ISBN* number, to the peer *Library*. Because the *Publication* relation of *Library* does *not* offer this information and rather has its own identification system based on a *Signature*, the attribute *ISBN* is projected out by the mapping:

$$\begin{aligned}&\mathit{Library.Publication}(\mathit{Title}, \underline{\mathit{Signature}}, \mathit{Year}) \subseteq\\ &\quad \mathit{AW.Book}(\mathit{Title}, \underline{\mathit{ISBN}}, \mathit{Year}).\end{aligned}$$

Observe that although *ISBN* is a key attribute for *AW.Book* and although *Signature* is a *different* key for *Library.Publication*, the mapping can be used to transfer tuples from *AW* to *Library*, because they can be identified by the compound key ($\mathit{Title}, \mathit{Year}$).

Based on the system model described in this section, we now turn our attention towards usage of the mappings for query answering. The following section introduces our notion of weak query containment that plays an important role in the algorithms for query planning in our PDMS setting.

2.2. Weak Query Containment

When rewriting queries the notion of query containment is an important means to compare between the original query and its rewriting. A query Q_1 is said to be contained in another query Q_2, denoted by $Q_1 \subseteq Q_2$, if the answer to Q_1 is a subset of the answer to Q_2 for any underlying database instance.

Chandra and Merlin [CM77] proved that the existence of a so-called containment mapping between Q_2 and Q_1 is necessary and sufficient for $Q_1 \subseteq Q_2$. A containment mapping τ relates symbols of Q_2 and Q_1 as follows [Les00]:

CM.1 To every distinguished variable of Q_2 a distinguished variable of Q_1 is assigned by τ.

CM.2 Every constant in Q_2 is mapped to a constant in Q_1.

CM.3 Each subgoal symbol of Q_2 is mapped to a subgoal in Q_1.

CM.4 The selection predicates of Q_1 imply the ones of Q_2: $\mathit{Sel}(Q_1) \Rightarrow \mathit{Sel}(Q_2)$.

Since in our setting we allow projections in peer mappings, the set of distinguished variables can differ between the query over a peer's schema and its rewriting over a neighboring peer's schemas. So traditional query containment mappings cannot be applied to our setting. In general, it is not always possible to find a containment mapping that maps *every* distinguished variable of the original query to a variable of the rewriting as the following example demonstrates.

Example 1. *In practice, it is possible that tuples returned from a rewriting have a different key as the corresponding result of the original query. Consider the following peer schemas on books, their authors, and the year of publication:*

$$P_1.BookOfAuthor(Author, Title, Year, Publisher)$$

$$P_2.Book(Author, Title, \underline{ISBN})$$
$$P_2.Published(\underline{ISBN}, Year)$$

Observe that the relation $BookOfAuthor(Author, Title, Year, Publisher)$ has the compound key $\{Author, Title, Year\}$. Using the GaV mapping

$$\begin{array}{lll} P_2.Book(Author, Title, ISBN), & & \\ P_2.Published(ISBN, Year) & \subseteq & P_1.BookOfAuthor(Author, Title, \\ & & Year, Publisher) \end{array}$$

to reformulate the query

$$q(Author, Title, Year) :- \quad P_1.BookOfAuthor(Author, Title, Year, Publisher)$$

results in the rewriting

$$\begin{array}{lll} q'(Author, Title, Year, ISBN) & :- & P_2.Book(Author, Title, ISBN), \\ & & P_2.Published(ISBN, Year). \end{array}$$

This mapping contains a projection both in the direction from P_1 to P_2 and in the inverse direction. The former refers to the variable Publisher at P_1 and the latter projects out the variable $ISBN$ at P_2. Observe that the variable $ISBN$ is necessarily a distinguished variable of q', because it maps to the only key both of P_2.Book and P_2.Published. Please note that $\{Author, Title, Year\}$ cannot be used to uniquely identify the tuples returned by the rewriting, because each of these variables can be `NULL`.

However, due to the fact that Publisher is projected out in the mapping, a classical containment mapping cannot be established between the original query and its rewriting. In particular, the condition CM.1 from above is violated, because Publisher cannot be mapped to any distinguished variable of the rewriting. □

To accommodate deficiencies of traditional containment mappings, Grahne and Kiricenko introduced the notion of p-containment [GK03, Kir03]. It requires (1) that there is an inclusion dependency between the contained query, in our case the rewriting, and the containing query. Additionally, (2) for every distinguished variable in the rewriting a corresponding distinguished variable must in the original query. Therefore, the missing mapping for the variable *Publisher* in our example above would be tolerated by p-containment.

However, that approach would rule out the above mapping, because the missing mapping for the variable $ISBN$ of the rewriting violates condition (2) of p-containment. We argue that p-containment can be even more relaxed by dropping condition (2) and only requiring mappings between variables occuring in selection predicates of the query. We adopted the notion of weak containment introduced in [HÖ6]. This diploma thesis has been conducted in the context of our research. Within this chapter we show how an existing query rewriting algorithm can be extended to consider weak containment thus become applicable for mappings containing projections.

Definition 4 (Weak Containment Mapping). *Let Q_1 and Q_2 be two queries. $Sel(Q_1)$ and $Sel(Q_2)$ denote the selection predicates of Q_1 and Q_2, respectively. A weak containment mapping ω between Q_2 and Q_1 has following properties:*

WCM.1 *Every variable $v \in Sel(Q_1)$ is assigned by ω to a variable $w \in Sel(Q_2)$.*

WCM.2 *Every constant $c_1 \in Sel(Q_1)$ is assigned to a constant $c_2 \in Sel(Q_2)$.*

WCM.3 *Each subgoal symbol of Q_2 is mapped to a subgoal in Q_1.*

WCM.4 *The selection predicates of Q_1 imply the ones of Q_2: $Sel(Q_1) \Rightarrow Sel(Q_2)$.*

Rather than *every* distinguished variable as demanded in CM.1 above, a weak containment mapping *only* requires the variables occurring in the selection predicates of the containing query to be assigned to counterparts in the contained query. The same holds for constants. This is the minimum set of variables and constants that have to be mapped to guarantee the tuples returned by the rewriting be correct with respect to the query.

Our weak form of query containment can be applied to the schema mappings that mediate between the schema of the original query and the schema of its rewriting. Hence, we enclose these mappings in the new concept of containment as expressed by the following definition.

Definition 5 (Weak Containment of Rewritings). *Let Q be a query over schema S_1. Let Q' be a rewriting of Q over a schema S_2. Let further $\mathcal{M}$ be a set of GLaV mappings that were used to rewrite Q into Q' and that may contain projections. Then the rewriting Q' is weakly contained in Q with respect to $\mathcal{M}$, if a weak containment mapping $\omega_{\mathcal{M}}$ exists between Q' and Q. We denote this by $Q' \subseteq_{\omega_{\mathcal{M}}} Q$.*

Building on this basis, we now turn our attention to techniques for query planning in the context of PDMS dealing both with GaV and LaV mappings at the same time.

2.3. Query planning

In a PDMS, a query may be posed to the peer schema of any peer. This query may consist of relational and comparison subgoals. To translate it, the subgoals are reformulated and passed along mappings to other peers, which in turn recursively send the query to their neighboring peers, etc. Reformulation terminates when all branches of recursion have reached local data sources, where the queries can be evaluated on stored data, or when a query is about to use the same peer schema relation or the same mapping a second time, i.e., a cycle has been completed.

The query results make their way back from the local sources to the peer that initiated the query. Winding their way back the results of the different alternative branches are combined. Clearly, this whole process can be performed fully decentrally. Each of the peers independently acts as a mediator. It receives a user or intermediate query and rewrites it using the schema mappings to neighboring peer schemas as well as those to its local data sources.

For query planning we are given a query over a peer schema as well as a set of mappings to local data sources and peer schemas of remote peers. Our goal is to find a query plan that returns all possible answers to the query at hand using the given mappings. Our process of local query planning proceeds in two steps:

1. *Expanding query subgoals.* We build a so-called rule-goal tree by applying the set of available LaV mappings in isolation from the set of GaV mappings. The LaV mappings are involved in an algorithm for answering queries using views [Hal01]. The result of this step are a sort of *partial* rewritings that will be specified in more detail in the following section. These rewritings are partial, because the set of LaV mappings not necessarily covers the whole query. Each of the GaV mappings is applied by view unfolding and in general also results in a partial rewriting. The rule-goal tree described in the following section nicely reflects these rewriting steps and the resulting partial rewritings.

2. *Forming rewritings.* This phase exploits the partial rewritings in the rule-goal tree to form full rewritings. This results in a query plan comprising solely rewritings whose parts can be sent to neighboring peers and or can be applied to local data sources. We also present the structure of this query plan in the following.

The approach to deal LaV and GaV mappings with different techniques has been introduced in the Piazza PDMS project [HIST03]. There, the peer employs the MiniCon algorithm [PL00] for answering queries using views, i.e., the LaV peer mappings, whereas GaV mappings are expended separately. They also perform the two steps to build a rule-goal tree and then create a query plan from that.

Important works on PDMS *exclude* projections from mappings between the peers, e.g., [HIST03, CGLR04]. Since we believe that this kind of intentional incompleteness often can be encountered in practice, we propose how to modify algorithms for PDMS query planning to consider peer mappings with projections as well. These changes build on our notion of weak containment in the context of query rewritings.

Interestingly, the MiniCon algorithm for answering queries using views [PL00] also first applies the mappings to find constituents of the rewritings, the so-called MiniCon descriptions, in the first step. The second phase combines these MiniCon descriptions in an efficient manner. Before we explain how we modify MiniCon by introducing weak containment, we describe the structure of a rule-goal tree.

2.3.1. Rule-goal Tree

The result of applying the mappings to rewrite the subgoals of the query at hand can be comprehensively represented as a rule-goal tree, Fig. 2.3. This form of an intermediate query plan was introduced to the context of PDMS in [HIST03]. In contrast to Piazza, our query planning approach is strictly local, hence the rule-goal tree only creates a relationship between a query received at a peer and the subgoals of rewritings over neighboring peers. Put together, the local rule-goal trees from a global, yet virtual query plan. We divide the tree into five levels:

1 Query Result The top level is the goal node of the predicate of the query result.

2 Query The rule node below the query result represents the query. In this work it is expressed as a Datalog rule.

3 Query Subgoals The goal nodes on this level are the subgoals of the query. The arc below the above rule node on Level 2 indicates the join between all the query subgoals forming a conjunctive query. We do not explicitly model the selection predicates related with the query subgoals on Level 3. Rather, we assume a function $Sel(s)$ for a subgoal s that returns all selection predicates of the query referring to any variable in s.

4 Mappings The rule nodes on this level represent the mappings used for reformulation of the query subgoals one level above. Again, an arc below a rule node denotes a join of the subgoals below.

5 Resulting Subgoals This level contains the subgoals resulting from applying the mappings in Level 4 to the query subgoals in Level 3. The subgoals of Level 5 are the constituents of the query rewritings. If a subgoal r of Level 5 returns attribute values for a query subgoal g on Level 3, we say that r covers g. Subgoals created by a local-as-view rewriting (LaV) are connected via a rule node to *any* of the query subgoals covered by the resulting subgoal. The dashed lines from the subgoals on Level 5 to query subgoals on Level 3 identify all other subgoals, so-called uncle nodes [HIST03], which are covered by that resulting subgoal.

Since the resulting subgoals actually are queries over the peer schema of a *neighboring* peer or a local source schema, they can be related by foreign-key relationships that have no counterpart in the peer schema the query is formulated over. Therefore, foreign-key relationships on Level 5 are modeled explicitly. Observe that joins between resulting subgoals can also be represented by an arc below a rule node in Level 4, i.e., a global-as-view rewriting (GaV).

We return to the foreign-key relationships on Level 5 when we describe the algorithm to transform a rule-goal tree into a local query plan consisting of join and union operators in Sec. 2.3.4.

2.3.2. Expanding Query Subgoals

In the first phase of query planning a peer computes constituents of the final rewritings for a query it has received. All of them are collected in a rule-goal tree described above. We now take a closer look into the algorithmic details of query rewriting in the PDMS context: We first focus on how well the parts of the rewritings produced in this step fit together. This aspect considerably influences the second part of the process that assembles the fragments together to obtain executable rewritings. A second requirement

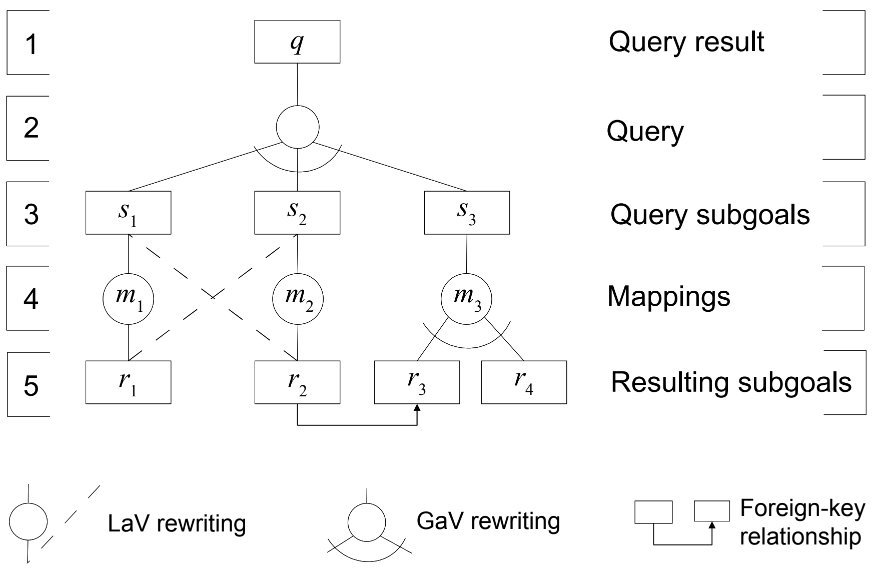


Figure 2.3.: Rule-goal tree created by query reformulation.

is to change existing approaches for PDMS query reformulation to cope with mappings containing projections.

Answering Queries using Views with Projections. Some important algorithms for answering queries using views, such as the BUCKET algorithm [LRO96], first consider each mapping in isolation and check which subgoals of the query to be answered can be covered by tuples resulting from that view predicate. In the BUCKET algorithm, each subgoal of the query collects the views that potentially deliver data for that subgoal. However, to find the query rewritings in the next step, this approach requires containment checks to filter actual rewritings from the Cartesian product of the contents of all the subgoal buckets.

The MINICON algorithm of Pottinger and Levy is considerably more efficient than the BUCKET algorithm [PL00], because it overcomes the enumeration of this Cartesian product. When looking at a view, MINICON also examines the join conditions of the query subgoal covered by that view. By doing this, the algorithm discovers additional query subgoals the view contributes data to. The view and the query subgoals that it covers are recorded in a so-called MiniCon description (MCD). Additionally, this artifact contains a partial containment mapping between the view and the corresponding query subgoals. Note that the term "partial" refers to the subgoals of the query rather than to the set of variables in the containment mapping. So it may not be interchanged with our notion of a weak containment mapping.

Performing the above kind of chase [AHV95] in the query can be used to decide whether a view can be combined with other views in a rewriting at all. In this way, MINICON can drop useless views in an early stage of the procedure rather than involving a view repeatedly in expensive containment checks many times as it is done by the BUCKET algorithm [PL00].

For applying both LaV and GaV mappings in our context, we start from the approach of the Piazza PDMS [HIST03] and show how to *relax* it by weak containment to better fit to our system setting. For a query at hand, we involve all LaV mappings into the first phase of the MiniCon algorithm, i.e., creating the MiniCon descriptions. GaV mappings are used to expand appropriate subgoals of the query by view unfolding. For this task, we propose to apply the same properties as for the views in our relaxed version of MiniCon, denoted by MiniCon^r. Next, we describe these properties together with our changes.

Our *relaxed* MiniCon description D^r for a view V applied to a query Q is a triple $\langle V, \omega, \mathcal{G}_D \rangle$ that contains

– the view V, which is part of the final rewriting,

– a partial mapping ω from $Vars(Q)$ to $Vars(V)$ relating query and view subgoals, and

– the set $\mathcal{G}_D$ of subgoals of Q that are covered by the subgoals in the view definition of V.

Next, we describe the original MiniCon algorithm and list its properties below. Briefly, MiniCon checks for each query subgoal g_i and each view subgoal v_j whether there exists a mapping φ from $Vars(g_i)$ to $Vars(v_j)$: $\varphi(g_i) = v_j$. Intuitively, the variable mapping φ is the containment mapping from the subgoals of the original query covered by V to the rewriting subgoal V. The original MiniCon and Bucket algorithms require that all distinguished variables of the query are supplied by the rewriting. This is the intuitive meaning of property M.1 below, which can also be applied for the Bucket algorithm.

Property M.2 introduces the improvement of MiniCon over Bucket. When MiniCon detects a mapping between a query subgoal and a view subgoal it changes the perspective and follows the join condition of the query subgoal. By this chase step, the algorithm can find other query subgoals that are covered by the view subgoal at hand.

The properties M.3 and M.4 of MiniCon refer to the treatment of selection predicates in the query and the mappings in our setting of semi-interval constraints. Given a set $\bar{X}$ of query variables, $Sel_{\bar{X}}(Q)$ denotes the set of selection predicates that only include variables in $\bar{X}$ and that at least contains one existential variable of Q. Intuitively, M.3 requires that the view must satisfy all selection predicates that refer to the set of variables in the domain of the corresponding variable mapping. To achieve the condition in M.4, selection predicates of Q not included in the preliminary rewriting obtained by combining the MCDs have to be added to form the final rewriting.

M.1 For each distinguished variable x of the query Q in the domain of φ, $\varphi(x)$ is a distinguished variable in V.

M.2 If φ(x) is an existential variable in V, following holds: For every subgoal g of the query Q that includes x, (1) all the variables of g are in the domain of φ, and (2) all variables of g are mapped to V: $\varphi(g) \in V$.

M.3 If $\bar{X}$ is the set of variables in the domain of φ of a MCD, then it is required for the MCD that $Sel(V) \supseteq Sel_{\bar{X}}(Q)$. In particular, a MCD must not contradict the selection predicates of the query in its view.

M.4 A final rewriting satisfies *all* predicates from $Sel(Q)$.

For our setting with *projections* in the mappings we argue that some of these properties are too restrictive and can be relaxed without losing their utility. Intuitively, we substitute the fragments φ of containment mappings from the original query to the rewritings by *weak* containment mappings. This means that in our relaxed MiniCon^r we do no longer require that *all* distinguished variables are mapped to distinguished variables in the view (M.1). This is illustrated in the following example.

Example 2. *Regard the rule-goal tree in Fig. 2.4. Assume that we are given extracts of the schemas of the emergency hospitalization of three hospitals J, K, and L that model patients and their admission differently:*

$$\begin{array}{l} J.Patient(\underline{SSN}, Name, Date) \\ K.Patient(\underline{SSN}, Name), K.Admission(\underline{AdmID}, Address, Date) \\ L.Patient(\underline{SSN}, Address, InID), L.Intake(\underline{InID}, Name, Date) \end{array}$$

The following LaV peer mappings are defined between these schemas:

$$\begin{array}{lll} m_1: & J.Patient(SSN, Name, Date) & \subseteq K.Patient(SSN, Name, AdmID), \\ & & K.Admission(AdmID, Address, Date), \\ m_2: & L.Patient(SSN, Address, InID) & \subseteq K.Patient(SSN, Name, AdmID), \\ & & K.Admission(AdmID, Address, Date), \\ m_3: & L.Intake(InID, Name, Date) & \subseteq K.Patient(SSN, Name, AdmID), \\ & & K.Admission(AdmID, Address, Date). \end{array}$$

As depicted in the rule-goal tree in Fig. 2.4, each of these mappings covers both *subgoals of the query*

$$\begin{array}{lll} q(SSN, Name, Address, Date) & :- & K.Patient(SSN, Name, AdmID), \\ & & K.Admission(AdmID, Address, Date). \end{array}$$

To understand why MiniCon^r *should obtain the depicted rule-goal tree as a result, we take a closer look at the variable projections in the mappings. They are listed in Table 2.1. The subgoal J.Patient resulting from applying m_1 does not return values for the K.Admission.AdmID and K.Admission.Address. Consequently, this mapping returns incomplete, yet still useful tuples for our query. If this result were to be materialized in the schema K, an AdmID has to be chosen using a Skolem function [FHP$^+$02]. That would provide a unique value for the key attribute AdmID of K.Admission.*

Mappings m_2 and m_3 are more subtle. While L.Patient in the head of m_2 comprises values for SSN and Address only, L.Intake only returns the "complementary" attributes Name and Date. Observe that although both m_2 and m_3 cover all *subgoals of our query in isolation, the query result would be more complete if a join between them is included*

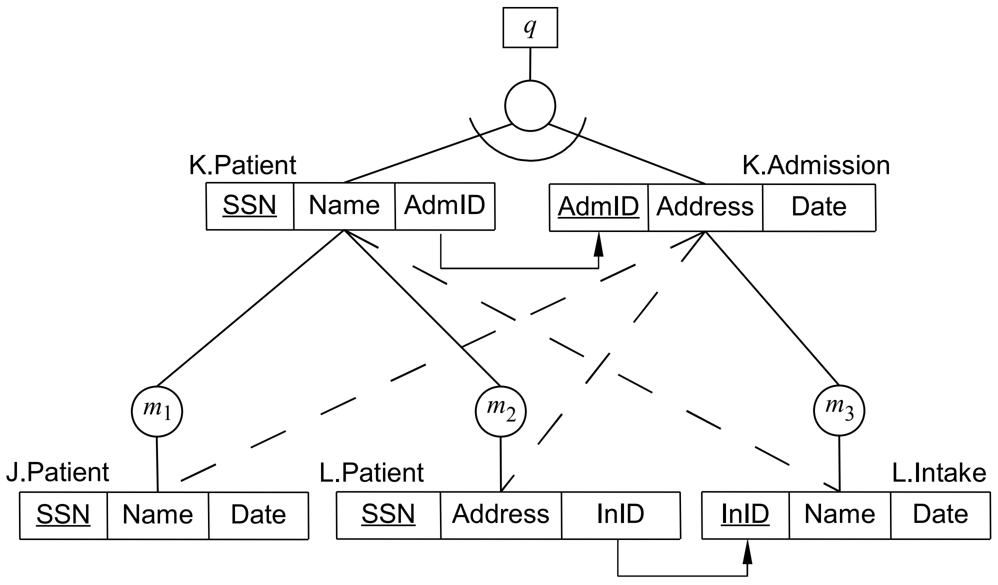


Figure 2.4.: Example for a rule-goal tree with two LaV expansions involving three mappings.

Mapping	Variables only in head	Variables in head and tail	Variables only in tail
m_1		*SSN*, *Name*, *Date*	*AdmID*, *Address*
m_2	*InID*	*SSN*, *Address*	*Name*, *AdmID*, *Date*
m_3	*InID*	*Name*, *Date*	*AdmID*, *Address*

Table 2.1.: Overview on the variable projections in our example on emergency hospitalization.

in the rewriting. This join returns tuples that cover all *attributes of the query subgoals. Instead, using m_2 and m_3 in isolation would only result in tuples for K.Patient and K.Admission that could not be joined, because they do not provide a value for the variable AdmID. The above join indicated by the foreign-key relationship between L.Patient and L.Intake on Level 5 of the rule-goal tree.*

The MiniCon *algorithm as published in [PL00] would rule out* each *of the resulting goal nodes on Level 5 in the rule-goal tree in Fig. 2.4. For instance, the variable Address is in φ and is a distinguished variable in the query. Since $\varphi(Address)$ is not distinguished in m_1, condition M.1 is violated and hence* MiniCon *will not create a MCD for it. Similarly, m_2 and m_3 are dropped. Since condition M.1 also holds for the* Bucket *algorithm, projections in mappings also enforce a relaxation for* Bucket. □

In our relaxed MiniCon^r we accept that a view does *not* return a variable that is assigned by φ to a distinguished variable x of the query, if and only if x does not occur in any selection predicate of the query. So we substitute properties M.1 and M.2 by relaxed conditions that consider selection predicates as well:

R.1 For each distinguished variable x of the query Q in the domain of φ that *also occurs in the selection predicates* $Sel(Q)$ of Q, $\varphi(x)$ is a distinguished variable in V.

R.2 If φ(x) is an existential variable in V, the following holds: For every subgoal g of the query Q that includes x, (1) all the variables of g *that are involved in selection predicates* are in the domain of φ, and (2) *all of these variables* are mapped to V: $\varphi(g) \in V$.

Condition $M.3$ remains the same for MiniCon^r. In Sec. 2.3.4 we describe a second relaxation of the MiniCon algorithm that applies to the phase that combines the MCDs to form the rewritings.

Creating the Rule-Goal Tree. In contrast to algorithms for answering queries using views, in our Pdms context rewritings can be created by exploiting both LaV and GaV mappings at the same time. In the following, we describe the creation of the rule-goal tree informally.

Similar to [HIST03], all LaV mappings to local sources or foreign peers are involved in the first phase of MiniCon^r for answering queries using views. Every MCD is inserted into the rule-goal tree as a goal node on Level 5 (see Fig. 2.3 on Page 23). Such a goal node is connected by a rule node to one of the goal nodes on Level 3 that are covered by this LaV mapping. All the other goal nodes from that set are connected to the Level 5 goal node by dashed lines indicating an "uncle" relationship [HIST03]. Observe that the foreign-key relationships between the Level 5 goal nodes resulting from the MCDs are a by-product of MiniCon^r. In contrast to former work, we annotate foreign-key relationships in the rule-goal tree to facilitate its transformation into an executable query plan as is described in the following two sections.

Since there is no need for coordination, every GaV mapping is applied to the peer query in isolation. It expands an appropriate goal node of Level 3 by traditional view unfolding [HIST03].

Example 3. *In the GaV expansion in Fig. 2.5 (a), the query subgoal g is expanded into a join of the subgoals $h_1, h_2, \ldots, h_l$ using the mapping m_1. A GaV mapping can be used only if the variables in the comparison predicates associated with the subgoal to be reformulated occur in the mapping head. For instance, this means that in the situation in Fig. 2.5 (a) it holds that*

$$Vars(Sel(g)) \subseteq \bigcup_i Vars(Sel(h_i)).$$

The LaV expansion in Fig. 2.5 (b) uses the mappings m_2, m_3, and possibly others to cover the subgoals of the query to be reformulated, namely $g_1, g_2, \ldots, g_k$. There, the dashed line indicates that the resulting goal node h covers both g_1 and g_2. □

The MiniCon algorithm published in [PL00] aims to combine the MCDs seamlessly to form *complete* rewritings. This is no longer necessary in the Pdms context. Here, goal

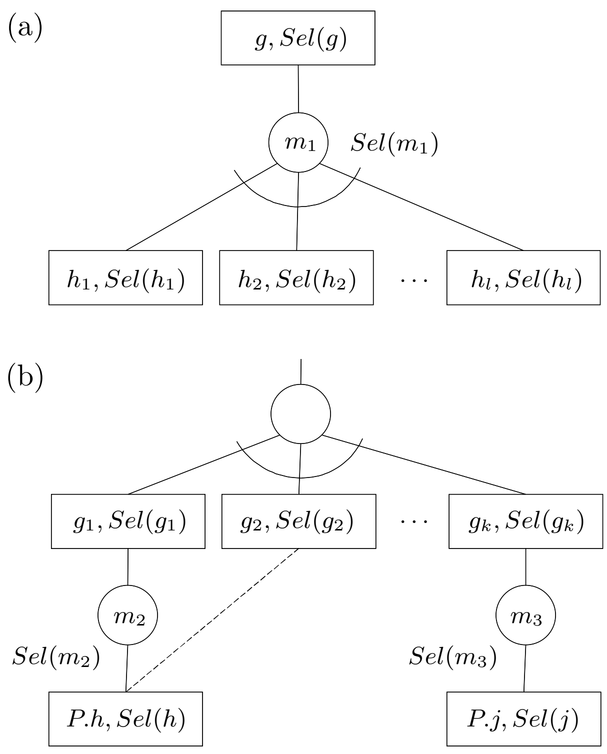


Figure 2.5.: Selection predicates (*Sel*) in the rule-goal trees of a GaV (a) and a LaV expansion (b). Rectangles represent goal nodes (subgoals of queries). Mappings are displayed as circles.

nodes created from MCDs can be combined with goal nodes resulting from applying GaV mappings as well. To this end, foreign-key relationships are required from the goal nodes resulting from MCDs and GaV expansions as indicated between the goal nodes below the LaV mapping m_2 and the GaV mapping m_3 in Fig. 2.3 on Page 23. These foreign-key relationships can be drawn from the peer schema if all involved goal nodes are instances of the *same* peer schema. If the goal nodes resulting from MCDs and those created by GaV expansions are related to different peer schemas, the foreign-key relationships between them could be discovered by data profiling techniques [BH03, IMH$^+$04].

Before we describe how to transform a rule-goal tree into an executable query plan in Sec. 2.3.4, the following section presents the structure of such a query plan along with some preliminaries.

2.3.3. Query Plan

While the rule-goal tree is advantageous to discuss the results of query reformulation at a peer, we need to transform it to a query plan to compute the query result. Such a query plan is made up by the join-merge operations, denoted by $\sqcap$, and full outerjoin-merge operations, denoted by $\sqcup$ [Nau02]. The operator $\sqcap$ performs join-like and $\sqcup$ union-like operations between data sets with possibly conflicting data values for the same real-world entity. We recall their definitions from [Nau02] in the following and use these symbols throughout this thesis to denote join respectively union operators.

Definition 6 (Join-Merge Operator $\sqcap$)**.** *Let R_1 and R_2 be two relations with the attribute sets $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Let $a_k \in \mathcal{A}_1 \cap \mathcal{A}_2$ be the foreign key that joins R_1 and R_2.*

$$\begin{aligned} R_1 \sqcap_{a_k} R_2 := \{\text{tuple } t \mid\ & \exists r \in R_1,\ s \in R_2 \text{ with} \\ & t[a_k] = r[a_k] = s[a_k], \\ & t[a] = r[a],\ \forall a \in \mathcal{A}_1 \setminus \mathcal{A}_2, \\ & t[a] = s[a],\ \forall a \in \mathcal{A}_2 \setminus \mathcal{A}_1, \\ & t[a] = f(r[a], s[a]),\ \forall a \in \mathcal{A}_1 \cap \mathcal{A}_2\}, \end{aligned}$$

where f is a resolution function that computes a value for each attribute of the resulting tuple t based on the given tuples r and s.

The definition of the full outerjoin merge operator $\sqcup$ builds on the outer-union operator $\uplus$ that performs a union between relations with differing attribute sets [Cod79]. The result of $\uplus$ comprises the union of the attribute sets of the input relations.

Definition 7 (Full Outerjoin-Merge Operator $\sqcup$)**.** *Let R_1 and R_2 be two relations with the attribute sets $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Let $a_k \in \mathcal{A}_1 \cap \mathcal{A}_2$.*

$$R_1 \sqcup_{a_k} R_2 := (R_1 \sqcap_{a_k} R_2) \uplus (R_1 \setminus (R_1 \sqcap_{a_k} R_2)[\mathcal{A}_1]) \uplus (R_2 \setminus (R_1 \sqcap_{a_k} R_2)[\mathcal{A}_2]).$$

The result of the full outerjoin-merge operator comprises every tuple of the given relations. Missing attribute values of a tuple that does not have a matching tuple in the other relation are padded with `NULL` values.

Before the structure of a query plan is explained, we introduce some additional preliminary notions.

Definition 8 (Result of a Mapping)**.** *A mapping m is used in isolation for query rewriting of a query Q. Then the tuple set returned by the rewriting of Q using m is called the result of the mapping m.*

When a mapping is used in isolation for query rewriting, the result of the mapping is transferred over the corresponding mapping path. If the mapping is involved in a join pushdown, Sec. 5.2.2, or in answering queries using views [Hal01], then the result of the mapping has to be joined with other conjuncts before it can be returned to the peer that performed the query rewriting. In the rule-goal tree the result of a mapping can be identified as all Level 5 goal nodes that are connected to the rule node of Level 4 that represents the mapping. The notion for the result of an individual mapping can naturally be extended to a set of mappings:

Definition 9 (Result of a Mapping Set)**.** *Let $\mathcal{M} = \{m_1, m_2, \ldots, m_n\}$ be a set of n mappings with their respective results $R(m_i)$. The result $R(\mathcal{M})$ is given by the conjunction of the results of all mappings in $\mathcal{M}$:*

$$R(\mathcal{M}) = R(m_1) \sqcap \ldots \sqcap R(m_n).$$

Generally, the results of mapping sets are parts of the query rewritings as produced by algorithms for answering queries using views [Hal01, LMSS95], for instance the MiniCon algorithm [PL00] or our relaxed form MiniCon^r, or finally by join pushdown to neighboring peers, Sec. 5.2.2. For individual mappings and sets of mappings containing projections we examine which attributes of the involved peer schemas and local source schemas are covered by the their results in Sec. 5.2 and Sec. 5.3.

Building on these concepts we are now ready to define the structure of the constituents of a query plan, i.e., the query rewritings:

Definition 10 (Query Rewriting)**.** *Let Q be a query over a peer schema. Let $Sel(Q)$ be the set of selection predicates of the query. A query rewriting is a multi-way join-merge*

$$Q' = C_1 \sqcap C_2 \sqcap \ldots \sqcap C_k \tag{2.5}$$

with $Sel(Q') \subseteq Sel(Q)$. Different subgoals $C_i \in Q'$ and $C_j \in Q'$ cover possibly overlapping subsets of subgoals of Q. Each subgoal C_i groups all alternative contributions of mapping sets with a full outerjoin-merge that contribute data to the query subgoals covered by C_i:

$$C_i = \bigsqcup_l (R(\mathcal{M}_l), Sel(\mathcal{M}_l))\,. \tag{2.6}$$

A rewriting returns a correct but possibly incomplete query result, i.e., can contain `NULL` *values in variables not involved in any selection predicate.*

The rewriting subgoals C_i group the results of different sets of mappings that cover the same query subgoals. These mapping sets can be of different size. For instance, certain query subgoals can be covered by two LaV mappings or three LaV mappings. The rewriting subgoals also are a group of *minimal* size of these mapping sets. This means that it is not possible to take a mapping away from a mapping set and still yield a correct rewriting due to the selection predicates that are associated with that mapping.

Previous works on query planning in data integration, e.g., the traditional MiniCon algorithm [PL00] or [Nau02] *exclude* that different subgoals of a rewriting cover overlapping sets of the query subgoals. In particular, MiniCon combines the MCDs such that they non-redundantly cover the query subgoals. In contrast, we argue that a higher completeness of the query result can be achieved if we allow that the subgoals of a rewriting may overlap in the sets of query subgoals they cover. Such a situation is mentioned in Example 2 on Page 25. We will take up this case in Sec. 2.3.4.

An example of a query plan is depicted in Fig. 2.6 and its general structure is defined as follows:

Definition 11 (Query Plan)**.** *Let Q be a query posed to a certain peer. Let $\mathcal{M}_1, \mathcal{M}_2, \ldots$ be sets of outgoing mappings of type local-as-view or individual global-as-view mappings. A query plan $P(Q)$ is a full outerjoin-merge of alternative query rewritings Q'_i that have the form $C_{i1} \sqcap C_{i2} \sqcap \ldots \sqcap C_{ik}$:*

$$P(Q) = \bigsqcup_i (C_{i1} \sqcap C_{i2} \sqcap \ldots \sqcap C_{ik})\,. \tag{2.7}$$

Each rewriting in $P(Q)$ has a unique assignment of the C_{ij} to the subgoals of Q. A particular conjunct C_{ij} can occur in different query rewritings and always covers the same set of subgoals of Q.

Even when the sets of query subgoals covered by different subgoals of a rewriting overlap it is important that the results of the mapping sets below each rewriting subgoal can be combined with those of the other rewriting subgoals using foreign-key relationships.

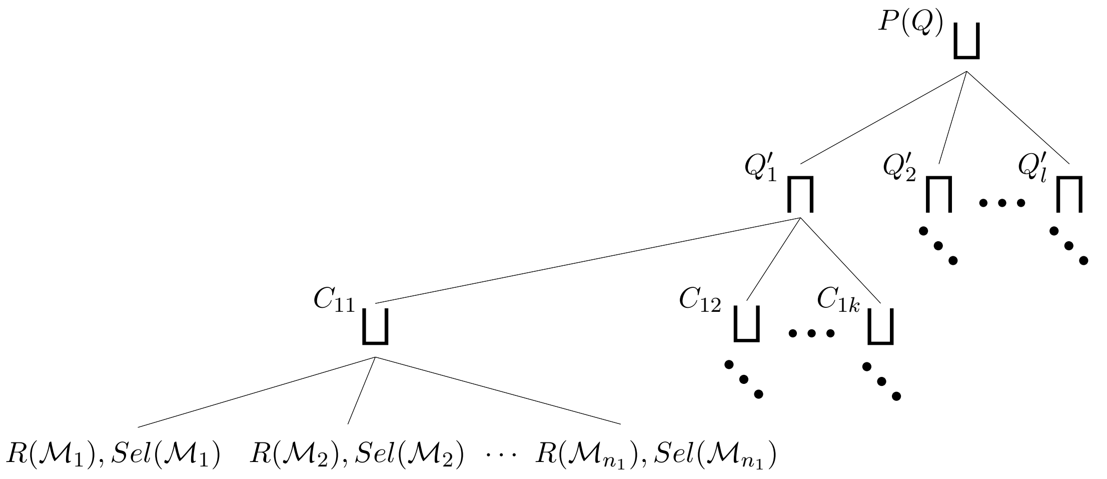

$P(Q)$
$\sqcup$
Q'_1
$\sqcap$
Q'_2
$\sqcap$
Q'_l
$\sqcap$
C_{11}
$\sqcup$
C_{12}
$\sqcup$
C_{1k}
$\sqcup$
$R(\mathcal{M}_1), Sel(\mathcal{M}_1)$
$R(\mathcal{M}_2), Sel(\mathcal{M}_2)$
$\cdots$
$R(\mathcal{M}_{n_1}), Sel(\mathcal{M}_{n_1})$

Figure 2.6.: General form of a query plan $P(Q)$ for a query Q at a peer consisting of join-merge ($\sqcap$) and full outerjoin-merge ($\sqcup$) operator nodes introduced in [Nau02].

For instance, assume that the query Q has three subgoals: $q :- s_1, s_2, s_3$. We present some possible assignments of rewriting subgoals to these query subgoals:

– $Q'_1 : \{R(m_1) \rightarrow \{s_1, s_2, s_3\}\}$: means that the LaV mapping m_1 covers all query subgoals. So the rewriting Q'_1 only comprises a single subgoal for m_1.

– $Q'_2 : \{R(m_2) \rightarrow \{s_1, s_2\}, R(m_3) \rightarrow \{s_3\}\}$: here the LaV mapping m_2 returns tuples for one part and the GaV mapping m_3 for the complementary part of the query subgoals.

– $Q'_3 : \{R(m_4) \rightarrow \{s_1\}, R(m_5) \rightarrow \{s_2, s_3\}\}$: the GaV mapping m_4 only covers the subgoal s_1 whereas the LaV mapping m_5 supplies results for the rest of the query subgoals.

In general, a particular rewriting subgoal can occur in *different* rewritings.

Moreover, note that there can be different rewriting subgoals C_a and C_b occurring in *different* rewritings that cover the *same* set of query subgoals. They cannot be substituted by each other since in general they can have different foreign-key relationships.

2.3.4. Transforming a Rule-Goal tree into a Query Plan

A query plan as described in the previous section is better suited for performing optimizations than the rule-goal tree from Sec. 2.3.1. The reason is that in a query plan the sub-queries to neighboring peers are grouped by their sets of query subgoals they cover. Additionally, from the structure of the query plan as introduced in the above section it is clear where joins and unions have to be performed.

In this section, we present an algorithm to transform a rule-goal tree into a query plan. This algorithm has to combine Level-5 goal nodes created by both LaV and GaV expansions. The main idea of that algorithm that forms the second phase of our relaxed MiniCon^r is to exploit all the foreign-key relationships between the goal nodes on Level 5 of rule-goal tree to exhaustively form all possible rewriting subgoals. These rewriting subgoals are then combined to form rewritings whose selection predicates satisfy those of the query at hand.

MiniCon^r extends the original MiniCon algorithm [PL00] in the way the MCDs are combined. As already described, each MCD maintains the set of query subgoals it covers. When the original MiniCon creates the rewritings, following additional property is considered:

M.5 A rewriting Q' can be only a combination of MCDs that fulfills following conditions:

(i) The union of all query subgoals covered by the MCDs in Q' contains *all* query subgoals.

(ii) The MCDs in Q' do *not overlap* with respect to the query subgoals they cover.

We argue that this property has to be relaxed to obtain maximally complete query answers in presence of projections in the peer mappings. Recall Example 2 from Page 22. There, the rule-goal tree contains two MCDs, namely the Level 4 rule nodes m_2 and m_3, that joined together in a rewriting yield a larger result for the given conjunctive query than applying them in isolation. Since foreign peer schema of hospital L has a different foreign-key relationship between the results of the mappings m_2 and m_3 than the local peer schema at hospital K the query is posed to, using the mappings in isolation would result in incomplete relations that cannot be joined to form an answer to the query at hand.

To provide a maximally complete query answer in such situations, we propose the following additional relaxation properties that complete the specification of our relaxed MiniCon^r:

R.3 A rewriting Q' can yield incomplete query answers with respect to the subgoals and the variables of the query. Variables for which Q' does not return any values are padded with `NULL`.

R.4 Different rewriting subgoals can overlap in the sets of query subgoals they cover.

R.5 The selection predicates of the rewriting entail those of the query: $Sel(Q') \supseteq Sel(Q)$.

We would like to emphasize again that MiniCon^r yields a higher completeness of the query answers than the original MiniCon algorithm, because it returns incomplete answers that are excluded by MiniCon. The algorithm to transform a given rule-goal tree into a query plan containing rewritings that follow the properties R.3 to R.5 is documented in Algorithm 2.3.1 on Page 34.

There, the set of goal nodes of the Level 5 of the given rule-goal tree T is denoted by $Level5(T)$. In Lines 2-4, the algorithm performs an exhaustive chase [FHP$^+$02, AHV95] to find all possible incomplete rewritings that follow the properties R.3 to R.5 above. They consist of the goal nodes of Level 5 of the given rule-goal tree (Line 2) connected by the foreign-key relationships between these goal nodes (Line 3) and are represented in by the set $\mathcal{R}$ (Line 4). For every subset $\mathcal{S}_i$ of the query subgoals, one potential rewriting subgoal $\langle C_j, \mathcal{F}_{u_i}, \mathcal{S}_j \rangle$ is created in Line 5. It contains a set $\mathcal{F}_{u_i}$ of unbound foreign-key relationships and the set of query subgoals covered by the rewriting subgoal C_j. Next, for each of the potential rewritings $\langle \mathcal{G}_i, \mathcal{F}_i \rangle$ a check for join pushdown is performed (Lines 7-15) and results of mappings belonging to the same remote peer are assigned to the rewriting subgoal that covers the same query subgoals as the result of the mapping set (as determined by the function *Covered*) and has the same unbound foreign-key relationships (returned by $UnboundFKs$). The latter is necessary that all contributions below a rewriting subgoal actually can be joined with their counterparts in other rewriting subgoals. Observe that in the mapping sets involved in join pushdown can be both LaV and GaV mappings. Next, all Level-5 goal nodes not involved in join pushdown are assigned to the rewriting subgoals where applicable with respect to the corresponding query subgoals and the foreign-key relationships (Lines 16-25). Finally, the algorithm creates the rewritings under consideration of the property R.5 for the selection predicates (Lines 26-31).

Since we believe that finding the fully expanded query plan at a peer is not performance-critical due to the typically small number of neighboring peers, this algorithm performs an exhaustive, non-optimized enumeration of rewriting subgoals. Rather, as it is more promising, the focus of this work is to exclude entire mapping paths from query processing.

Kiricenko also applies a her limited form of query containment to relax the MiniCon [Kir03]. Since our concept of weak query containment is more general and we consider foreign-key relationships within the schema of the rewritings, our version MiniCon^r yields results of higher completeness.

2.4. Semantics of Query Answering

The common approach to define semantics for PDMS is based on the notion of certain answers [AD98], which immediately can be applied to an individual peer [CGLR04, FKLS03, HIST03]. For a local-as-view mapping, a relation of a local schema is defined as a view on the peer schema. Hence, answering a query against the peer schema amounts to solving the “answering queries using views” problem [Hal01]. So a certain answer is defined to be part of *any* instance of the peer schema that is consistent with the above

```
   Input  : Query $Q$ with selection predicates $Sel(Q)$, rule-goal tree $T$
   Output: fully expanded query plan $\hat{P}(Q)$
1  $\hat{P} \leftarrow \bot$
2  $\mathcal{Q} \leftarrow \{s_1, s_2, \ldots, s_n\}, s_i \in Q$                         {query subgoals}
3  $\mathcal{G} \leftarrow \{g_1, g_2, \ldots, g_m\}, g_i \in Level5(T)$                {Level-5 goal nodes}
4  $\mathcal{F} \leftarrow \{g_i[Key(g_i)] \supseteq g_j[f_k]\},\ g_i, g_j \in \mathcal{G}$         {Level-5 foreign-key rel.}
5  $\mathcal{R} \leftarrow \{\langle \mathcal{G}_i, \mathcal{F}_i \rangle\},\ \mathcal{G}_i \in \mathcal{G},\ \mathcal{F}_i \in \mathcal{F}$ such that R.3, R.4, R.5 hold     {rewritings}
6  $\mathcal{C} \leftarrow \{\langle C_j, \mathcal{F}_{u_j}, \mathcal{S}_j \rangle\},\ \mathcal{S}_j \in \mathcal{Q},\ \mathcal{S}_k \neq \mathcal{S}_l$ if $k \neq l$    {rewriting subg., $\mathcal{F}_{u_i}$ unbound}
7  foreach $\langle \mathcal{G}_i, \mathcal{F}_i \rangle \in \mathcal{R}$ do                              {join pushdown}
8      group subgoals in $\mathcal{G}_i$ by peers into sets $\mathcal{G}_{P_r}$
9      foreach $\mathcal{G}_{P_r}$ do
10         repeat
11             $\langle C, \mathcal{F}_u, \mathcal{S} \rangle \leftarrow$ next $\langle C_j, \mathcal{F}_{u_j}, \mathcal{S}_j \rangle \in \mathcal{C}$ such that $\mathcal{S}_j = Covered(R(\mathcal{M}_{P_r}))$
12         until $\mathcal{F}_u = UnboundFKs(R(\mathcal{M}_{P_r}))$
13         add $R(\mathcal{M}_{P_r})$ and $Sel(\mathcal{M}_{P_r})$ to $C$
14     end
15 end
16 foreach $\langle \mathcal{G}_i, \mathcal{F}_i \rangle \in \mathcal{R}$ do                 {assignment of other goal nodes}
17     foreach $g \in \langle \mathcal{G}_i, \mathcal{F}_i \rangle \mid g \notin \mathcal{C}$ do
18         create $R(m)$, $Sel(m)$ using $g$ and the corresponding mapping $m$
19         foreach $\langle C_j, \mathcal{F}_{u_j}, \mathcal{S}_j \rangle \in \mathcal{C}$ do
20             if $\mathcal{S}_j = Covered(R(\mathcal{M}_{P_r}))$ and $\mathcal{F}_{u_j} = UnboundFKs(R(\mathcal{M}_{P_r}))$ then
21                 add $R(m)$ and $Sel(m)$ to $C$
22             end
23         end
24     end
25 end
26 foreach $\langle \mathcal{G}_i, \mathcal{F}_i \rangle \in \mathcal{R}$ with $C_1, C_2, \ldots, C_k \in \mathcal{C}$ do            {create rewritings}
27     $Q' \leftarrow C_1 \sqcap C_2 \sqcap \ldots \sqcap C_k$
28     if $Sel(Q') \supseteq Sel(Q)$ then                                     {property R.5}
29         add $Q'$ to $\hat{P}$
30     end
31 end
32 return $\hat{P}$
```

Algorithm 2.3.1: Transforming a fully expanded rule-goal tree into a query plan (Transform step of MiniCon^r). *Covered* returns the set of subgoals covered by a result of a mapping set and *UnboundFKs* the foreign-key relationships that are not bound within this result.

mapping from the peer schema to the local schema and the database instance of the local source.

Now we can extend this understanding to the whole PDMS and give an intuitive description of the *semantics* of PDMS query answering: Let a (global) source database be the assumed outer-union [Cod79] of the local source database instances of *all* peers, i.e., the database state of the PDMS. Additionally, let a global database of a PDMS be an assignment of attribute values to *all* peer schemas, i.e., the assumed union of all peer schema instances mentioned above [CGLR04, HIST03]. Please note that an instance of a particular peer schema is a so-called solution to the data exchange problem with the local schema as source and the peer schema as target [GLLR07]. The semantics of a PDMS is the set of *all* possible global databases. Given a (global) source database, the certain answers to a query is the set of tuples being part of *all* the global databases in the above semantics.

Of course, the global databases mentioned above depend on how the mappings between the peers are interpreted [CGLR04]. The approach implemented in the Piazza PDMS [HIST03] employs global first order logic semantics, which interprets the mappings as material logical implications. This enables global reasoning to obtain query answers and thus amounts to the open-world assumption with respect to a particular peer. In contrast, the approach of Calvanese et al. [CGLR04] considers mappings to be specifications of the exchange of certain answers [GLLR07], which actually means a closed-world assumption from the perspective of a particular peer. The authors argue that their epistemic logic based semantics better represents modularity of the peers, i.e., better preserves peer autonomy as discussed in Section 1.2. Therefore, our system also follows the closed-world assumption for a particular peer.

The problem of finding the certain answers to a query in a PDMS under globally defined semantics is undecidable [CGLR04, HIST03]. Cycles in mapping paths can lead to non-terminating query reformulation. Even if cycles are recognized and query reformulation is terminated according to a certain criterion [HIST03], there may be situations, in which correct tuples are missed. Approaches based on local semantics show decidable query answering even in PDMS with cycles in the network of mappings [CGLR04]. However, under this semantics some answers are missed that are found under global semantics, because it does not employ global reasoning.

There are several works that also examine semantics of query answering in data integration [Gra02, Kha08] or data exchange [Lib07] the context of incomplete information. In contrast to them, this work resorts to a more intuitive and experimental consideration of this topic.

2.5. Information Loss in PDMS

A consequence of our approach of local reasoning is information loss during the process of distributed query processing in a PDMS. Next, we describe this problem, which is one source of inefficient query answering.

In many practical situations, queries to foreign peers cannot contain projections to keep the query service simple. In particular, for heterogeneous peer schemas with a complicated peer mapping it is not immediately clear how to "translate" projections from one peer schema into another. A peer sending a query to a neighbor is then forced to transport the complete result rather than being able to specify which portion of resulting relation is desired. To reflect such situations in this thesis, we exclude projections from our queries.

In contrast to queries, peer mappings contain projections to specify that heterogeneous peer schemas provide different attributes of a real-world entity. Since peers are not able to adapt their queries to projections in peer mappings, they act as filters in the flow of data along mapping paths. This means that data provided by peers "behind" a mapping to a neighboring peer can be dropped because of projections in the mapping to that peer. As is discussed in detail in Sec. 4, projecting out even a single attribute of a relation significantly decreases the completeness of that intermediate query result. So actually valuable information can be lost. This is inefficient, because these data may have been transmitted over several peer mappings before and may have induced considerable load in several database computations.

Clearly, the effect of information loss along mappings is a small result size at the end of a mapping path. One of the main ideas of this thesis is to prune mapping paths that promise comparably few result data from further reformulation to reduce cost of query answering. Essentially, by these pruning decisions peers aim to find a trade-off between completeness and cost of the query result.

2.6. Trade-off between Completeness and Cost

This section characterizes our solution as a best-effort approach and introduces our main ideas. The data complexity of PDMS query answering can be measured by the size of the virtual overall query plan. This plan can be created by building the union of all local query plans actually computed at all peers involved in answering a particular query. In general, the size of the overall query plan is exponential in the number of mappings to neighboring peers and the size of the query in terms of relations. So even in PDMS with tens of peers, there is an astonishingly high number of redundant mapping paths between two peers. For instance, in an example with about 30 peers we observed an overall query plan with about 34,000 union and about 17,000 join operators.

In the spirit of [NFL04] we argue that in large-scale PDMS users do not demand complete query answers. Similar to information retrieval with search engines, users of PDMS with a large number of peers cannot inspect all query answers in detail. On the other hand, it is important to have reasonable response times in our scenario to satisfy users even with incomplete query answers.

To achieve such a satisfying trade-off between completeness and cost of query answers we propose to make concessions to the completeness of query answers – turning it into an optimization goal rather than requiring a complete answer. The main idea of our approach is to exploit these concessions to increase efficiency of query answering. To this end, we prune mappings promising less data than others, Chapter 6 [RN05].

Additionally, resources in large PDMS may be limited, i.e., users require a query answer within a *maximum* response time. In Chapter 8 we propose solutions to this problem.

They spent a limited budget intelligently [RNHS06].

2.7. Applications

At this point we continue the discussion of application scenarios for our approach of local query optimization for large-scale from the introduction. We focus on the issues of peer autonomy and the scale of the PDMS and explain why they are important in the following application fields.

Virtual organizations, both in industry and science, spanning multiple stakeholders often emphasize the autonomy of their information resources. Usually, data stores and applications are at the core of the respective business of the individual organizations. Therefore, it is this business that drives the structure and usage of these resources. Exchanging data with other organizations is often of lower priority and the corresponding interfaces are kept simple and have to adapt to the requirements of the organization's purposes rather than to an optimized data flow within the virtual organization. To meet this practice, we assume highly autonomous peers that need to provide only simple interfaces for information sharing, e.g., a simple query language and exclusively local optimization approaches.

Disaster Data Management [NR06, HIST03] is an area where both highest flexibility and large-scale are crucial. In case of a nationwide crisis, systems might have to be connected that were never supposed to cooperate before. Since the disaster is spread over large regions, the resulting integrated data management system is also complex. These requirements ideally fit to the characteristics of PDMS. It is moreover realistic that during a serious crisis computing resources are limited due to broken networks or lack of electricity. So the question arises which level of query answering service can be achieved under these circumstances. This problem is studied in Chapter 8 of this thesis.

The *Semantic Web* [SS06, HIMT03] comprises such a huge variety of information that it seems impossible to establish centralized query answering services even for particular subareas. Offerings in the Web pop up and get out of date frequently. So an extremely flexible approach like PDMS is best suited for information sharing in this highly volatile environment. With today's search facilities, it is commonly accepted that search results in the Web are incomplete and even of low relevancy with respect to the search query. PDMS for the Semantic Web promise to return answers of much higher relevancy than current search engines. But this can only work with an effective trade-off between the result quality and the cost of query answering in such an infrastructure. Exactly this topic is addressed by this thesis.

Networked Personal Information Management Systems (PIM) [ANR07] can help organizations save much of the effort their employees spend for searching useful information. Estimations for the amount of time employees use for searching information differ, but many of these studies claim that these activities take more than half of the whole time. PIM are an approach to address this enormous potential. However, as organizations are complex corresponding PIM are complex as well. In [ANR07] we proposed to use PDMS to interconnect PIM environments of the individual employees. Again, as organizations desire to be agile, corresponding information systems also face this challenge.

2.8. Related Work

In his work on mediators [Wie92], Wiederhold mentions that mediators may use other mediators to answers queries. Since peers in our sense act as mediators, this was the first proposal for the system architecture of PDMS, i.e., an arbitrary network of mediators. The work in [Les00] introduced a mapping language of similar expressivity as GLaV mappings and studied semantics and efficiency of query answering. In some sense, it can be seen a basis for query answering in mediators in the context of PDMS. Building on a large body of literature in the field of peer-to-peer file sharing systems, Halevy et al. identified several research questions for peer-to-peer systems employing data integration techniques [GHI+01]. In [BGK+02], Bernstein et al. formulate their vision on peer-based data management.

In the following, we provide an overview of the main types of PDMS approaches. In [HRZ+08], we surveyed these system proposals by identifying a set of characteristics of PDMS. We found a set of different approaches that are roughly decribed in Table 2.2 on Page 40. As explained in Sec. 2.4, the main difference between different variants of PDMS is whether query planning is performed on a global level or in an isolated manner at each peer. The Piazza system [HIST03, TH04, Tat04] employs global information about the whole PDMS and computes *all* certain answers. However, this setting is only tractable under certain conditions of which an acyclic topology is the most important one. Note that this requirement is nearly never given in practice. Experiments with PDMS of any approach clearly show that the effort for query answering grows rapidly with the size of the system. So intractability can actually occur in practice. Another drawback of the Piazza architecture is that their pruning technique involves non-local information about the query plan as we discuss in Sec. 6.2.1.

The work of Calvanese et al. [CGLR04] follows a fully local approach for query planning. As mentioned in Sec. 2.4, local reasoning misses answers that are found under global first order semantics, because that actually amounts to an open-world assumption. In contrast, local query planning means a closed-world assumption. Our work also computes all query answers under their semantics but additionally performs query optimization in a best-effort manner.

There are several research prototypes that take a best effort approach, for instance our full-fledged PDMS *Humboldt Peers* [RNHS06], SmurfPDMS [HLS06, Hos09], or the approach of Petrakis et al. [PKP04]. They all have in common that they compromise the completeness of the query result. Besides that, every of these approaches has its own focus. SmurfPDMS performs skyline computation. The work reported in [PKP04] aims at discovering clusters of peers with similar data. *Humboldt Peers* puts special emphasis on optimization techniques that preserve the autonomy of the peers by exploiting query feedback for query planning purposes. In contrast to other approaches, *Humboldt Peers* is capable to deal with volatile PDMS instances independently where changes in the system happen. Moreover, our work is the first that reduces data overlap between alternative mapping paths to reduce cost, Sec. 9.

2.9. Summary

This chapter introduced query processing at a peer in the context of both global-as-view and local-as-view mappings. It presented an algorithm to transform a rule-goal tree into an executable query plan comprising joins and union operators.

We examined how information loss accumulates along mapping paths due to projections in the mappings. As a consequence, many attribute values are unnecessarily transported between several peers before a projection in a peer mapping eliminates them. We also discussed that selections in peer mappings differentiate redundant mapping paths. Both of these phenomenons can be exploited to prune the search space and thus increase efficiency of distributed query answering in PDMS. We argued that in many large-scale information sharing applications users are satisfied with incomplete query answers. Therefore, the main problem addressed in this thesis is to trade-off completeness of the query result and cost of query answering.

Category	**Alternatives**	Global planning + complete answers [HIST03]	Local planning + incomplete answers [RNHS06] [HLS06] [PKP04]	Local planning + complete answers [CGLR04]
System model				
Topology	Arbitrary		×	×
	Arbitrary without cycles	×		
Peer autonomy	Exchange queries + results only			×
	Exchange metadata with queries		×	
	Non-local containment checks	×		
Information loss	No projections in mappings	×	[HLS06, PKP04]	
	Projections in mappings		[RNHS06]	
Semantics				
Semantics of query answering	(Global) first-order logic	×		
	(Local) reasoning		×	×
Decidability	Decidable in general		×	×
	Decidable for acyclic topologies	×		
Query optimization				
Query planning	Global (OWA)	×		
	Local (CWA)		×	×
Complexity	co-NP complete	×		
	polynomial		×	×
Relaxation of completeness	Complete answer sets	×		×
	Incomplete answer sets		×	
Pruning	Optimizing cost for complete answers	×		
	Trading off cost and completeness		×	

Table 2.2.: Characteristics of PDMS and features of exemplary PDMS approaches [HRZ$^+$08].

3. Humboldt Peers: Building a PDMS

To investigate our query planning and pruning strategies experimentally, we have developed a full-fledged PDMS implementation called *Humboldt Peers*, which is able to generate PDMS instances [Sch06, HÖ6, Tie06, Tie09] This chapter gives an overview of its functionality and the system architecture

Although *Humboldt Peers* contains a variety of measurement facilities, it is a real PDMS rather than a simulation environment. In that, it nicely abstracts many practical scenarios in which a set of independent information systems is in place that only exchange a minimal set of queries and their results.

3.1. Features

As shown in Fig. 1.1 on page 4, peers in Humboldt Peers consist of

– a relational peer schema,

– a set of local sources connected to the peer schema via local mappings, and

– peer mappings from one peer schema to the schemas of other peers.

This metadata is stored at each peer and includes only information about the peer itself and the mappings to the direct neighbors. As depicted in Fig. 3.1 on Page 43, peers only hold metadata that can be gathered locally (local metadata). This reflects usual practical scenarios where no global catalog on all systems exists.

Query planning is *fully decentralized* and implemented locally at the peers. Based on the given local-as-view (LaV) and global-as-view (GaV) peer mappings, a local rule-goal tree (Sec. 2.3.1) is created at the peer receiving the original query as well as at every peer that is contacted during query processing. After local optimization and possibly pruning, the resulting local query plan (Sec. 2.3.3) is executed. This process requires no global knowledge at all; query planning, optimization, and pruning strategies are based on local peer information only.

For experimental purposes, *Humboldt Peers* features a specialized *Monitor peer* (see Fig. 3.2 on Page 3.2 for its GUI) to which the regular peers report their activities. This peer can be used to create, visualize, manipulate, and query a PDMS. During query execution this peer can perform a live-visualization of the mapping paths already used for query reformulation and passing back results. The monitor peer is also able to collect numerous measurement data that can be used for query execution analysis and statistics, for instance execution time, number of database operations, number of peer queries, number of peers used for query answering and local mappings, completeness and density (Sec. 4.1) of result sets, virtual global query plan (composed of local query plans). Please note that the measurement facilities of the monitor peer do not influence local reasoning for query answering. In particular, the *Monitor* peer does not provide any data about other peers to any of the peers in the PDMS. The Monitor peer also contains a workload generator that generates, persistently stores, and executes a random query workload over all peers in the PDMS. Additionally, there is a central facility for changing the topology of a PDMS instance by removing certain peers from the network and possibly activating them again at a later point in time.

Each peer comprises extensive logging services. Especially for the histogram statistics for cardinality and overlap (see Chapter 5) we implemented useful visualization and logging functionality. Of course, the measurements for the Monitor peer and the logging functionality mean overhead cost for query answering. However, since our focus is on relative comparisons of different approaches the absolute query answering time is less important.

Humboldt Peers serves as a testbed for the query optimization approaches examined throughout this thesis. Hence, most of the techniques discussed in the remainder of this work were extensively evaluated by experimenting with our system.

3.2. System Architecture

Humboldt Peers has been developed using Java and consists of three main subsystems that are depicted in Fig. 3.1:

– *Monitor Peer*: The purpose of this subsystem is to create PDMS instances and conduct the experiments with them. It contains the graphical user interface (GUI) that displays the peer graph, the list of peers as well as detailed information on them such a peer schemas, mappings, peer activity status, or estimation accuracy of the histogram statistics. The PDMS generator [HÖ6] is a powerful facility to automatically create PDMS instances of arbitrary size. This process can be controlled by a rich set of parameters, Sec. 3.3.

  The *Deployer* component assigns the PDMS model created by the generator to a set of computers in the network. On each of them, several so-called PDMS Peers can run within a single Java Virtual Machine (JVM). Their creation and registration at the JXTA network (see below) is supervised by the *Controller* component. Please note that the *Controller* is not directly involved in query processing during runtime of the PDMS. Rather, it acts as a supporting facility to put the PDMS instance into operation and control the system during runtime.

  The query processing statistics are measured orthogonally to the actual query processing functionality and are available afterwards at the *Monitor*. Collecting these statistics is completely independent from query processing. The latter does not use any intermediate non-local query processing statistics.

– PDMS *Peers*: There can be several PDMS peers running in a distributed mode. Each of them "models" a real-world peer with own query answering facility, cost model, and statistics. The PDMS peers are created by the *Deployer* and the *Controller* component. Several PDMS Peers can run within a single JVM on a computer in the network. The local metadata mentioned in Fig. 3.1 comprises the peer schema, peer mappings, local schemas and mappings, as well as the statistics on cost, cardinality and data overlap discussed throughout this thesis.

In Humboldt Peers, any existing RDBMS with a JDBC driver can be embedded as a local data source. If no DBMS is available, Humboldt Peers uses an internal HSQLDB[1] database for storing temporary data and to perform joins as well as unions.

– *JXTA Network*: This system is based on the JXTA framework[2], which supports communication between distributed peers, so-called nodes, or across the Internet. JXTA offers methods to establish a peer-to-peer system, offer services at the peers and for communication between the peers. For instance, to create a P2P system, peers can automatically find each other by a multicast technique. During our first experiments, it turned out that JXTA consumes considerable resources. This led to the design decision to run several PDMS peers within a single JVM that commonly use one instance of JXTA.

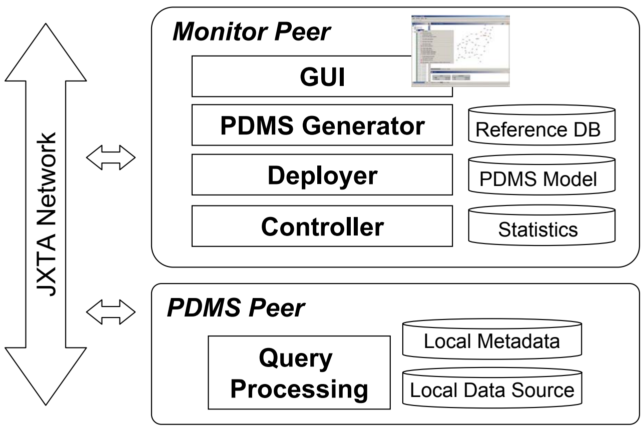


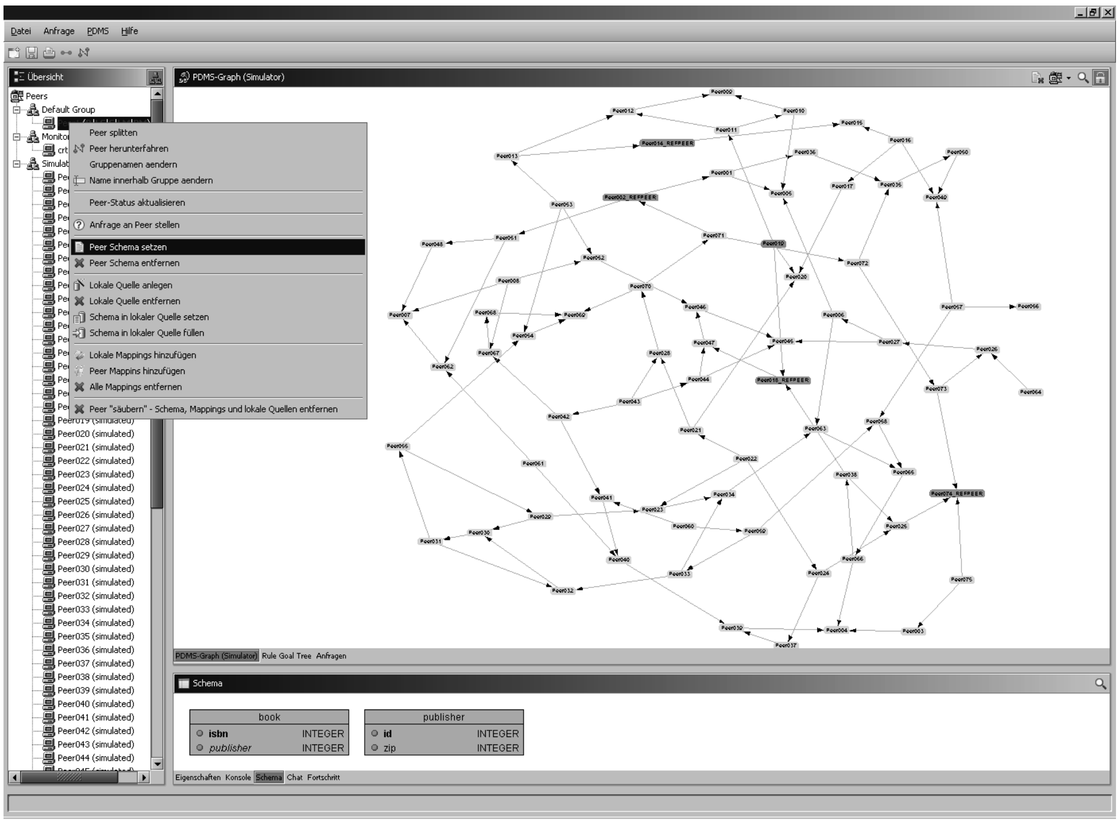
Figure 3.1.: Architecture of Humboldt Peers with monitor peer and an exemplary PDMS peer.

Query processing in *Humboldt Peers* can be done in sequential and parallel mode. Sequential means that all outgoing queries to neighboring peers are fired one after the other by a peer. The peer waits for the result of a query before sending out the next one. In the parallel query processing mode [Tie06], all outgoing queries at a peer are sent at the same time. Downstream peers can then process these queries in parallel to each other. Clearly, the parallel mode only leads to a speed up of the whole computation if there are several computers involved. If an experiment is performed on a single-core computer, sequential query processing is sufficient.

[1]`hsqldb.org`
[2]`www.jxta.org`

The cost model for query processing has to be specific for either sequential or parallel query processing. For simplicity, we implemented only a sequential cost model. To simulate different types of networks with different bandwidths, we introduced an artificial transport time into the cost model. Additionally to the actual time for transporting the result data back to the peer where the query originated, this artificial transport time is added. The bandwidth can be set for each peer mapping individually.

The source code of *Humboldt Peers* comprises about 650 classes and about 80,000 lines of code. We have tested the system with up to 50 peers across a network and were able to answer queries against data distributed across all peers.



book	
isbn	INTEGER
publisher	INTEGER

publisher	
id	INTEGER
zip	INTEGER

Figure 3.2.: Humboldt Peers with automatically generated PDMS. On the left, the peer structure of the PDMS is displayed. Below the peer graph, detailed information on selected peers is presented, e.g., a peer schema.

3.3. PDMS Instance Generation

To perform meaningful experiments, it is important to create numerous PDMS instances with different characteristics and execute queries on them. To simplify this step, *Humboldt Peers* includes a PDMS generator that was developed by Hübner [HÖ6]. This generator creates a given number of peers, local sources, various heterogeneous schemas, and mappings including selection predicates for the created peer graph. This task can be controlled by a set of parameters referring to the structure of the peer graph, the distribution of selections and projections in the peer mappings, and the data distribution across the peer graph. Note that the creation of a PDMS instance that meets a considerable set of constraints is difficult. To meet this challenge, Hübner examined several techniques for finding a satisfying approximation to this constraint solving problem.

The generator takes a reference schema as input and varies this schema by normalization and de-normalization to yield heterogeneous candidate schemas for the peers. As the relationships between the reference schema and these candidate schemas are known, the generator can create appropriate mappings between the derived peer schemas. Additionally, instances of local sources are created, again using data given with the reference schema or by automatically generating new data values. Finally, these peers are assigned to differently shaped PDMS graphs, such as simple chains, circles, trees, and completely random graphs, and then distributed across the network to the different peers by the *Deployer* component.

Alternatively, a PDMS instance can be modified manually or even built from scratch by creating peers, setting peer schemas and peer mappings, and assigning instance data to them. Using this functionality, any PDMS instance can be created to study certain scenarios.

3.4. Summary

This chapter presented our PDMS implementation *Humboldt Peers*. It is a full-fledged, fully-functional relational peer data management system whose peers can run distributed across a network. *Humboldt Peers* contains a powerful instance generator that greatly helps to create experiment configurations. Additionally, we described the system architecture and the measurement facilities of the system.

Part II.

Completeness-related Metadata in PDMS

4. Benefit and Cost of Query Answering

The main objective of information integration is to obtain a view on the world that is as complete as possible. The higher the number of data sources, the more information can be expected about a certain real-world entity. So intuitively, completeness of the query result is the most important benefit in information integration settings. A first dimension of this completeness is the cardinality of the result set. If projections are involved in mappings, it additionally has to be considered how many data values are returned in each result tuple. Besides completeness, Part II of this thesis describes why data overlap between alternative mapping paths is an important metadata input for query optimization.

Classical query optimization in centralized databases heavily depends on accurate estimations for sub-result cardinality in query execution plans. In this part we show why PDMS pose more difficult challenges when gathering the above completeness-related statistics, and we discuss several approaches to establishing metadata statistics in the context of a PDMS.

The following section defines the most important metadata for optimizing query answering towards high efficiency in terms of completeness and cost. First, we relate the completeness model for integrated information systems introduced by Naumann et al. [NLF99, Nau02, NFL04] to the PDMS context. Next, we go beyond this model and examine why data overlap between alternative mapping paths at a peer occurs and how it influences the efficiency of query answering. Finally, we review several works for cost models in data integration and develop our cost model on this basis.

4.1. Completeness of a Query Result

In many scenarios, users of large integrated information systems are not interested in *all* certain answers, Sec. 2.4, because they are not able to examine all of them in detail. Further, a constraint of large PDMS is the limitation of resources for query evaluation and transmission of query results. Facing these restrictions, a user may be satisfied with a fraction of all answers. This concession is related with the so-called extensional completeness, i.e., the number of tuples in a query result. For instance, in Web search users usually examine a very small fraction of the query answer. In such cases, it is desireable to rank the result tuples according to their relevancy. Next, we define different measures concerning completeness.

Coverage. Extensional completeness, also known as *coverage*, describes the proportion of the size of a tuple set to the number of *all* tuples accessible within a PDMS. The measure applies both to the data set a peer actually stores and to a query result.

To calculate coverage we make the closed world assumption. The *world* comprises all real-world entities represented *within* a PDMS. We assume to know the size $|w|$ of this world. In practice, however, knowing this number precisely is not necessary, because it only plays the role of a normalizing factor. We formalize this notion as follows:

Definition 12 (Coverage)**.** *Let $\mathcal{D}_Q$ be a set of tuples answering a query Q. The coverage of $\mathcal{D}_Q$ with respect to a world w is $c(\mathcal{D}_Q) := |\mathcal{D}_Q|/|w|$.*

Density. The *intension* queried by the user is the set of attributes $\mathcal{A}_Q$ asked for in the query. Intensional completeness of data sets, also known as density, first suffers from `null` values in data sources. Secondly, attributes that are asked for in the query may not be available at certain data sources in the PDMS. The user may be interested in having tuples in the query result even though they miss some of the desired attributes. In this way, the overall completeness of the query can be increased. Missing attribute values are filled with `null` values, thus creating incomplete tuples. As introduced in [Nau02], attribute density is used as a measure for this kind of completeness:

Definition 13 (Attribute density)**.** *Let a_R be an attribute of a relation R. A projection of a tuple t of this relation to a_R is denoted by $t[a_R]$. With $\perp$ denoting `null`, the attribute density of a tuple set $\mathcal{D}$ for R is defined as*

$$d(a_R) := \frac{|\{t \in \mathcal{D} \,|\, t[a_R] \neq \perp\}|}{|\mathcal{D}|}.$$

Similar to coverage, the density of a data set is also query dependent as expressed by the following definition:

Definition 14 (Query-dependent density)**.** *Let $\mathcal{A}_Q$ be the set of attributes asked for in a query Q. We define the query-dependent density of a tuple set $\mathcal{D}$ as the average attribute density of the attributes in $\mathcal{A}_Q$*

$$d_Q(\mathcal{D}) := \frac{1}{|\mathcal{A}_Q|} \sum_{a \in \mathcal{A}_Q} d(a).$$

Completeness. Intuitively, overall completeness can be regarded as an aggregate measure for the ratio of the amount of data in a certain data set, e.g., the result set of a query, to the amount of data in the world w. This means that it is a combination of coverage and density, which we aim to maximize in query answering:

Definition 15 (Query-dependent completeness)**.** *Let $\mathcal{A}_Q$ be the set of attributes asked for in a query Q. We denote the value of an attribute $a \in \mathcal{A}_Q$ by v_a. The query-dependent completeness for a tuple set $\mathcal{D}$ is*

$$C_Q(\mathcal{D}) := \frac{|\{v_a \in \mathcal{D} \,|\, v_a \neq \perp\}|}{|w| \cdot |\mathcal{A}_Q|}.$$

In [Nau02] it is shown that the completeness score of a data source S can be calculated as $C(S) = c(S) \cdot d(S)$. There, the query is posed to the universal relation, which comprises all attributes of all relations contained in the global schema of a data integration system. We can use these results in the context of a PDMS by regarding the peer schema the query is posed to as the (universal) relation, which includes the attribute set $\mathcal{A}_Q$ introduced above. So this equation naturally applies to an intermediate query result of a PDMS as well.

Since the overall completeness is a product of extensional and intensional completeness, it can increase by retrieving more but perhaps incomplete tuples compared to the usual requirement that the resulting tuples should be intensionally complete, i.e., having only non-`null` values in all attributes.

4.2. Completeness of mappings

To follow our main objective of efficient query answering in PDMS, our approach is based the concession of accepting incomplete query answers. This is realized by pruning decisions during query planning, Sec. 2.3.

Intuitively, query planning for a PDMS is a search problem. Its search space is made up of peer schemas and peer mappings. The goals of the search are the local data sources at the "border" of the search space. Using local mappings to reach them forms the final search step. During exploration of the search space, we lack information about the completeness contribution of local data sources not found yet. As a consequence, to intelligently explore the search space collecting query results, we must decide which *mappings* promise to be useful. This leads to the question how mappings and the data sources "behind them" contribute to the overall completeness of the query result.

In this work, we assume the mappings to include only select-project-join (SPJ) queries. In the following, we show how to calculate the influence of S, P, and J operations used in the mappings on the coverage and density of query results. Additionally, query plans contain union-type operators, which collect results returned by alternative mapping paths starting from a certain peer. Again we describe their influence on the completeness of the result.

4.2.1. Influence of selections and projections

Applying a selection σ to a tuple set of a relation R reduces the set of tuples by a selectivity factor s. Hence, we can calculate coverage of the selection result as

$$c(\sigma(R)) = s \cdot c(R). \tag{4.1}$$

Observe that in our definition the higher the selectivity s the *more* tuples are returned by the selection.

Assuming that `null`-values are distributed uniformly over all tuples, density is not affected by a selection:

$$d(\sigma(R)) = d(R). \tag{4.2}$$

We use this model with a mapping selectivity to study the influence of selections along mapping paths in PDMS. In further chapters of this work we introduce a more advanced approach, because the above selectivities would have to be query-dependent to be sufficiently accurate. In particular in Chapter 5 we describe our approach to use histograms to estimate the result size of mapping paths. For now, we assume the above selectivities to be given and use them to abstract from the histograms.

Note that these mapping selectivities are applied to the data of the target of a mapping, but also on all other sources reachable through that mapping. For instance, in the PDMS in Fig. 4.1 the selectivity annotated to the mapping from P_1 to P_2 applies to the data returned by P_2. For simplicity, we assume here that projections do not reduce coverage,

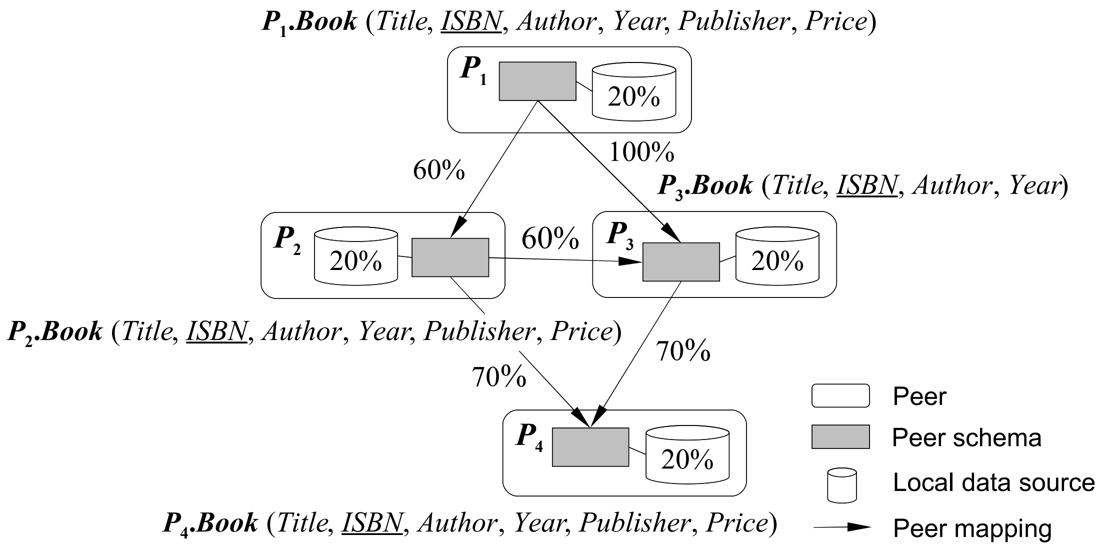

P1.Book (Title, ISBN, Author, Year, Publisher, Price)
P1
20%
60%
100%
P3.Book (Title, ISBN, Author, Year)
P2
20%
60%
P3
20%
P2.Book (Title, ISBN, Author, Year, Publisher, Price)
70%
70%
P4
20%
P4.Book (Title, ISBN, Author, Year, Publisher, Price)
Peer
Peer schema
Local data source
Peer mapping

Figure 4.1.: Example PDMS with coverage scores and mapping selectivities.

i.e., duplicates generated by projection are not eliminated. Additionally, observe that the query issued by the user also can contain selections. Their selectivity has to be "concatenated" with the mapping selectivities. That happens in the same way as for mapping selectivites along a mapping path as it is shown in the following example.

Example 4. *Regard the* PDMS *in Fig. 4.1. Suppose P_1 receives a query Q. The data at P_4, for instance, are then filtered by the selective mappings $P_1 \rightarrow P_2$ and $P_2 \rightarrow P_4$. If we assume statistical independence between the selections in the mappings, they together yield a selectivity of* $60\% \cdot 70\% = 42\%$. *If the tuples from P_4 are transported via P_3 to P_1, the selectivity of the mapping path is* $100\% \cdot 70\% = 70\%$.

Without concessions to the completeness of query answers, *projections* would not be allowed in mappings. Using projections in mappings means to reduce the intensional completeness of the resulting tuples. Attributes projected out have to be padded with `null`-values. But, as stated above, by allowing projections in our mappings we can explore a larger part of the query-dependent world w.

Projections in mappings massively influence the density of the result of a mapping (Def. 8). To understand this, we quantify this influence and find that the *query-dependent* density value has be to recalculated subtracting the density of the set of attributes projected out:

Theorem 1. *Let $\mathcal{A}_R$ be the attribute set of a relation R. Let $\mathcal{A}_P$ be the set of attributes R is projected to. Then the density of the projection $R[\mathcal{A}_P]$ of R to $\mathcal{A}_P$ is calculated from the given density of R as*

$$d_Q(R[\mathcal{A}_P]) = d_Q(R) - d_Q(R[\mathcal{A}_R \setminus \mathcal{A}_P]). \tag{4.3}$$

Proof. Let the attribute set of the relation R be $\mathcal{A}_R = \{a_1, \ldots, a_n\}$. Without loss of generality, suppose the attribute set, on which R is projected, to be $\mathcal{A}_P = \{a_1, \ldots, a_i\}$. We use the definition of the query-dependent density from Def. 14 with $\mathcal{A}_Q$ being the attribute set of the query:

$$\begin{aligned}
&d_Q(R) - d_Q(R[\mathcal{A}_R \setminus \mathcal{A}_P]) = \\
&\qquad \frac{1}{|\mathcal{A}_Q|}(d(a_1) + \ldots + d(a_i) + d(a_{i+1}) + \ldots + d(a_n)) \\
&\qquad - \frac{1}{|\mathcal{A}_Q|}(d(a_{i+1}) + \ldots + d(a_n)) = \\
&\qquad \frac{1}{|\mathcal{A}_Q|}(d(a_1) + \ldots + d(a_i)) \\
&\qquad + \frac{1}{|\mathcal{A}_Q|}(d(a_{i+1}) + \ldots + d(a_n)) \\
&\qquad - \frac{1}{|\mathcal{A}_Q|}(d(a_{i+1}) + \ldots + d(a_n)) = \\
&\qquad \frac{1}{|\mathcal{A}_Q|}(d(a_1) + \ldots + d(a_i)) = d_Q(R[\mathcal{A}_P]).
\end{aligned}$$

Observe that all these transformations still hold if some of the attributes of R do not occur in the query Q. Their densities are set to 0. □

Intuitively, this means that the influence of the projection on the query-dependent density is proportional to the fraction of attributes that is projected out. However, this only holds exactly, if the density of all attributes of R is the same and the remaining set $\mathcal{A}_P$ of attributes and the attributes $\mathcal{A}_R \setminus \mathcal{A}_P$ projected out include the same number, but not necessarily the same set, of query attributes.

In other words, since query-dependent density is always based on the size $|\mathcal{A}_Q|$ of the attribute set of the query at hand, the influence of the projection can be measured by counting non-`null` attribute values of the tuples of a relation. As a projection removes a vertical part of a relation, the difference in the query-dependent density can be obtained by simply subtracting the number of non-`null` attribute values removed.

As a consequence, according to Equation (4.3) the part of the information loss caused by a projection in a mapping can be quantified by calculating *only* the query-dependent density of the attributes projected out. We use this observation in our heuristic to decide which mappings suffer from loss of information [RN05], Sec. 6.

Since the projection $R[\mathcal{A}_P]$ leaves the number of tuples of R unchanged, the coverage of the result is not affected:

$$c(R[\mathcal{A}_P]) = c(R) \tag{4.4}$$

Example 5. *Consider the relation P_4.Book from Fig. 4.1. The following table lists the attribute densities and the query-dependent density of P_4.Book as it is exported by peer P_4 and as it is perceived by P_3 over the mapping from P_3 to P_4, which projects out the attributes Publisher and Prize.*

a) First, we assume the query asks for all attributes. The following table lists the attribute densities and the query-dependent density at both peers:

a	*Title*	*ISBN*	*Author*	*Year*	*Publisher*	*Price*	d_Q
$d_{P_4}(a)$	*90%*	*100%*	*80%*	*60%*	*40%*	*70%*	*73%*
$d_{P_3}(a)$	*90%*	*100%*	*80%*	*60%*	*0*	*0*	*55%*

As can be seen, projecting out a third of the attributes reduces the query-dependent density by about a third in this example. If the coverage had not been affected (having a mapping selectivity of 1), the completeness of P_4.Book perceived at P_3 would have been decreased by a third.

b) Next, we assume that the query projects out the attributes ISBN and Price. Then, the result is as follows:

a	*Title*	*Author*	*Year*	*Publisher*	d_Q
$d_{P_4}(a)$	*90%*	*80%*	*60%*	*40%*	*68%*
$d_{P_3}(a)$	*90%*	*80%*	*60%*	*0*	*58%*

Now, the decrease in d_Q is only about 15%, because the attribute Price that is projected out in the mapping $P_3 \to P_4$ is also not considered in the query.

□

4.2.2. Influence of joins

Suppose we are given the tuple sets of the two relations R_1 and R_2 together with their respective coverage and density values. We aim to calculate completeness for the result of the join $R_1 \sqcap R_2$. In [Nau02] formulas for calculating the *expected* coverage and density for these join-merge operators are derived.

Both the way of calculation and the results of coverage and density for joined relations strongly depend on the information overlap, i.e., the number of real-world entities represented in both data sets. If we assume *independence* of the representation of objects, we can use the following formulas for the expected coverage and density for $R_1 \sqcap R_2$, where $\mathcal{A}$ denotes the union of the attribute sets of R_1 and R_2:

$$c(R_1 \sqcap R_2) = c(R_1) \cdot c(R_2) \tag{4.5}$$

$$d(R_1 \sqcap R_2) = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} (d_{R_1}(a) + d_{R_2}(a) - d_{R_1}(a) \cdot d_{R_2}(a)). \tag{4.6}$$

Intuitively, Equation (4.5) states that the probability of a real-world entity to be represented in both relations and thus enter the join result is the product of the probabilities of being in one of the given relations. This property is associative, so coverage can be calculated along paths of mappings. To determine the density of the joined relations, shown in Equation (4.6), we have to count the number of non-`null` values in both relations for each attribute and then subtract the corresponding number of tuples counted twice.

Please note that because the above equations calculate only the *expected* value for the coverage of $R_1 \sqcap R_2$, the actual overlap situation can lead to results that significantly differ. For very small overlaps the values may be much less and for large overlaps they are higher.

If the extensional overlap X between the joined relations is known from statistics about about data overlap, Sec. 4.4, a more accurate assessment is possible. With the size of the overlap $|X|$ and the size of the world $|w|$ we may use following equation from [Nau02]:

$$c(R_1 \sqcap R_2) = |X|/|w|. \tag{4.7}$$

Intuitively, the resulting coverage is the "coverage of the overlap".

4.2.3. Influence of unions

The answer to a query posed to a PDMS is usually made up of the union of many contributions from alternative mapping paths. As stated above, these contributions can comprise different sets of attributes. Hence, in general they are not union-compatible. To overcome this restriction, a more flexible union operator is needed. For instance, the "outer union" operator as suggested by Codd [Cod79] pads attributes not contained in both relations with `null`-values before performing a usual union operation. Additionally, it is mentioned in [EN00] to expect that the set of common attributes includes a common key. This can be used to eliminate duplicates by the "outer union" operator. This requires that tuples representing the same real-world entity have the same value in the key attribute, e.g., the *ISBN* attribute of a relation *Book*.

The full outerjoin-merge operator $\sqcup$ from [Nau02] is similar to the "outer union", but allows all common attributes but the key attribute to have *conflicting* data values. In this work we use the full outerjoin-merge operator to calculate the completeness of the above combinations of the results returned by alternative mapping paths, i.e., query plans, and thus may draw the corresponding equations from [Nau02]. The following equation provides the expected coverage of a full outerjoin-merge for the case of statistical independent data sets:

$$c(R_1 \sqcup R_2) = c(R_1) + c(R_2) - c(R_1) \cdot c(R_2). \tag{4.8}$$

For the density of a "union" realized by a full outerjoin-merge the equation includes the coverage of the join-merge operator $\sqcap$. First, the attribute density of an individual attribute a amounts to:

$$\begin{aligned} d_{R_1 \sqcup R_2}(a) = & \frac{d_{R_1}(a) \cdot c(R_1)}{c(R_1 \sqcup R_2)} + \frac{d_{R_2}(a) \cdot c(R_2)}{c(R_1 \sqcup R_2)} - \\ & \frac{d_{R_1}(a) \cdot d_{R_2}(a) \cdot c(R_1 \sqcap R_2)}{c(R_1 \sqcup R_2)}. \end{aligned} \tag{4.9}$$

Using this, the density of $R_1 \sqcup R_2$, which comprises the attribute set $\mathcal{A}$ is the arithmetic mean of the attribute densities:

$$d(R_1 \sqcup R_2) = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} d_{R_1 \sqcup R_2}(a). \tag{4.10}$$

Again, if we have more information about overlap, Equations (4.8) – (4.10) can be refined. There, also associativity of the coverage criterion for $\sqcap$ and $\sqcup$ is shown. However, density is proven to be associative only for independent data sets [Nau02]. We take up this result in the next section.

Having the basic means to calculate the expected completeness scores for the most important relational operators occurring in mappings and query plans, the next section shows how to apply them to calculate the completeness of partial and entire query plans.

4.3. Completeness of query plans

In this section we apply the completeness model to calculate the completeness of an entire query plan as it was presented in Sec. 2.3.3. The completeness of a query plan can be used to decide about pruning of parts of a fully expanded query plan at a peer as discussed throughout the Sections 6 - 9. To illustrate the structure of a PDMS query plan, we return to our running example.

Example 6. *The virtual global query plan of our example introduced above (see Fig. 4.1 on page 52) is displayed in Fig. 4.2 below. The root of the tree is built by the query predicate. Here, we assume a user at peer P_1 wants to retrieve all books in the* PDMS. *Full outerjoin-merge operators are used to combine contributions from alternative mapping paths. The respective peer is annotated next to the operator symbol. We assumed the mappings between the local data sources and their peer schema to have no information loss. All the mapping selectivities from the peer mappings are annotated at the corresponding edge. Even this simple* PDMS *shows a considerable redundancy within the query plan, e.g., the local data source of peer P_4 occurs three times. Also note that because this is a virtual* global *query plan, it does not correspond to the structure of a local query* *plan presented in Sec. 2.3.3.* □

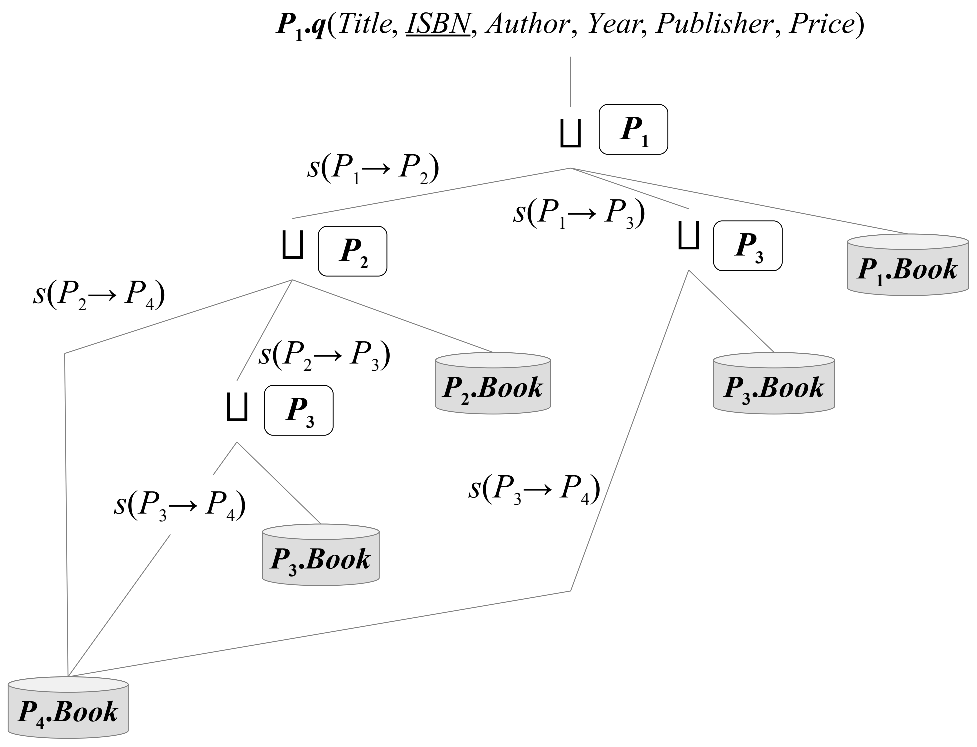


Figure 4.2.: Overall query plan of our example PDMS.

To calculate the completeness of the query plan in Fig. 4.2, we traverse it recursively. For each node containing a full outerjoin merge ⊔, the following is performed:

1. Before combining results from local data sources, they are transformed to the peer performing the full outerjoin merge. This is achieved by multiplying the coverage of the local data sources with the mapping selectivities annotated in the query plan according to Eqn. (4.1).

2. Now the coverage and density scores of the data sets can be combined making use of Equations (4.8)–(4.10).

Notice that, at some nodes of the query plan we have to combine more than two local data sources. This is only possible if we can assume associativity of the full outerjoin-merge operator ⊔ in our setting. But because of the redundancy of local data sources, some data sets to be combined are *not* independent from each other. For instance, $P_4.Book$ and $P_4.Book \sqcup P_3.Book$ in the left of Fig. 4.2 have the "known" overlap of $P_4.Book$. So in general, we have a *mixed* overlap situation [Nau02] in a *global* PDMS query plan, in particular because peers can be reached through multiple paths and because peers can play different roles in a single path. Dealing with such settings is complex and there are no formulas available yet. In Chapter 9 we examine the overlap between alternative mapping paths in more detail. This helps assess the accuracy of the formulas for independent data sets when they are applied in situations where the overlap significantly differs from the case of statistical independence.

For now, we accept some error created by using the formulas for the case of independent data sets. To get an intuition about this error we use a simple example.

Example 7. *To quantify the error of using the formulas for independent data sets for full overlap, assume all attributes of $P_4.Book$ to have the same attribute density value $d_Q(a)$. Now we calculate the query-dependent density of the full outerjoin merge of $P_4.Book$ with itself. The result should have the same query-dependent density as $P_4.Book$, because it is actually the same data set. If we use Eqn. (4.9) for independent data sets, we get the values for $d_Q(P_4.Book \sqcup P_4.Book)$ listed in the following table for varying coverage and attribute density of $P_4.Book$:*

$c(P_4.Book)$	$d_Q(P_4.Book \sqcup P_4.Book)$		
	$d_Q(a) = 0.2$	$d_Q(a) = 0.5$	$d_Q(a) = 0.8$
0.2	*0.22*	*0.53*	*0.82*
0.5	*0.25*	*0.58*	*0.85*
0.8	*0.31*	*0.67*	*0.91*

For small coverage values, which are more likely to be encountered in practice, the deviation of $d_Q(P_4.Book \sqcup P_4.Book)$ from the uniform attribute density as the correct value is moderate (e.g., max. 10% for $c(P_4.Book) = 0.2$ and max. 25% for $c(P_4.Book) = 0.5$). As the coverage increases, the error rises as well (e.g., max. 55% for $c(P_4.Book) = 0.8$). □

So, even in case of one data set containing the other, i.e., full overlap, the formulas for independent data sets lead only to an acceptable error in most practical situations. Knowing this, we apply these formulas to our query plan. Additionally, the problem of the inaccurate independence assumption is addressed by removing data overlap from local query plans as shown in Chapter 9.

Example 8. *Using the above procedure we calculate the completeness score for the query plan in Fig. 4.2. Recall that each of the four local data sources has a coverage score of 20%. For the overall completeness of the plan we get $C(q) = 55\%$.*

Now we illustrate our heuristic to prune mappings providing poor information quality. Suppose we decide to prune the mappings from P_2 to P_3 and from P_3 to P_4, because they include both a reduced selectivity and loss of a third of the attributes asked for in the query due to projection. In this example, this leads to a query plan without redundancy. Although we save the expansion of two mappings, we nearly achieve the same completeness of $C_p(q) = 52\%$. If we use the number of peer mappings used as a rough cost measure, the pruned query plan needs only about a third of the cost of the fully expanded query plan. □

4.4. Information Overlap between Mappings

Information integration aims at obtaining a complete view on the universe of discourse. To this end, data from many sources are combined. Usually, information on a certain real-world entity can occur in *several* of these sources. Because different data sources in general contain different attributes of that particular real-world object, added value is created by merging this data into a single consistent and coherent result. To enable this, different tuples representing the same real-world entity that stem from different sources have to be identified. In the literature, this problem is called duplicate detection, object identification, object fusion, or record linkage [New88, PAGM96, WN05, DHM05]. In this work, we assume that two tuples representing the same real-world entity share a common key attribute a_k. If they have the same value for a_k, we conclude that the tuples belong to the same real-world entity. Algorithms for duplicate detection create such an identifying attribute, if it does not exist. As mentioned in [Nau02], this is the extensional dimension of overlap:

Definition 16 (Extensional Overlap)**.** *Let R and S be two relations with the attribute sets $\mathcal{A}_R$ and $\mathcal{A}_S$ respectively. $R[\mathcal{A}]$ denotes the projection of the data set R to the attribute set $\mathcal{A}$. Let $\mathcal{A}_k$ be a key for both R and S. Then, the set of tuples*

$$\{t \,|\, \exists\, r \in R, s \in S \text{ with } t[\mathcal{A}_k] = r[\mathcal{A}_k] = s[\mathcal{A}_k]\}$$

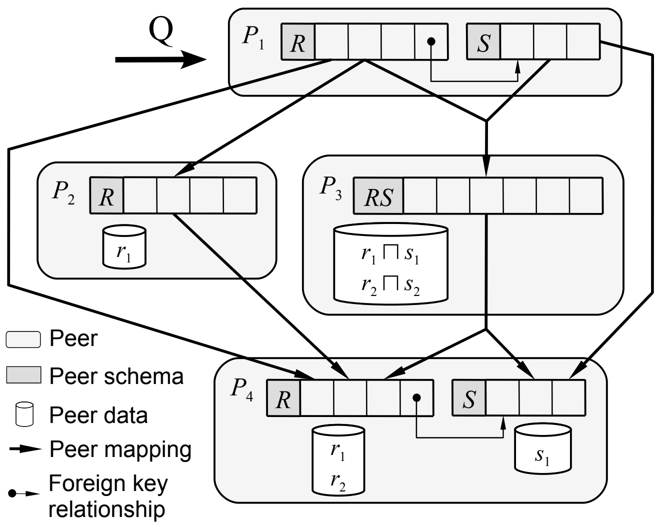
is called the extensional overlap of R and S.

Extensionally overlapping tuples can differ in their attribute sets, i.e., their intension. So the second orthogonal dimension overlap is defined as follows:

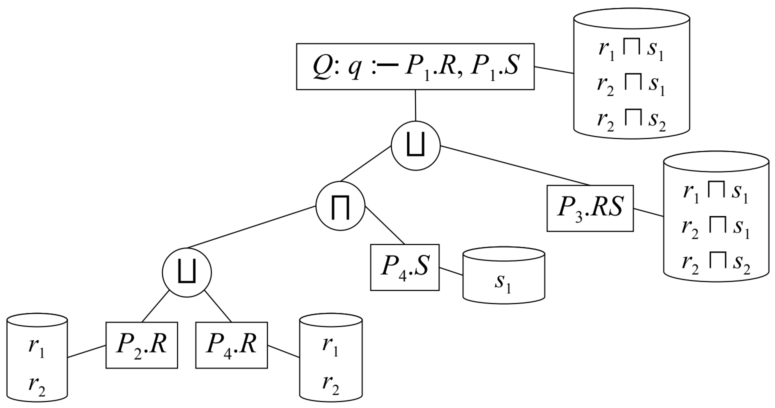
Definition 17 (Intensional Overlap)**.** *Let R and S be two relations with their attribute sets $\mathcal{A}_R$ and $\mathcal{A}_S$. The intensional overlap between R and S is given by the attribute set $\mathcal{A}_O = \mathcal{A}_R \cap \mathcal{A}_S$.*

Since peers are autonomous, the same real-world entity can be represented at several peers. A special characteristic of PDMS is that *redundant mapping paths* can lead to *extensional* overlap between result sets retrieved over these paths, even there is no extensional overlap among any sources. For instance, two mapping paths starting at the same peer P_i can meet at another peer P_j. If the data returned by P_j are not joined with other data on their way to P_i, the overlap visible at the peer P_i will comprise all tuples initially returned by the peer P_j. Recall that query planning extends in the direction of mapping paths, whereas the result data flows back in the opposite direction.

Example 9. *We introduce the simple illustrative example in Fig. 4.3(a) on Page 60 to study the potential savings of considering data overlap between neighboring peers in* PDMS *query planning. Each peer schema comprises at least one of the two relations R and S, which are connected by a foreign-key relationship (at peers P_1 and P_4) or occur in de-normalized form as a single relation RS at peer P_3. Peer P_1 has no local data. The relations at the other peers are populated by the tuples r_1, r_2, s_1, and s_2 as depicted*



(a) PDMS with peer schemas and mappings. The symbols in the peer data denote individual tuples.



(b) Query plan local at P_1 for the query Q:− $P_1.R, P_1.S$ of our illustrative example. Each relation is annotated with the result data it returns (to P_1) after evaluating this plan.

Figure 4.3.: Illustrative example for overlap-driven query answering.

in the database symbols. The relations in the peer schemas are related by peer mappings.

Suppose a user poses the query $Q\ :-P_1.R, P_1.S$ at P_1. Peer P_1 uses its outgoing mappings to create the query plan displayed in Fig. 4.3(b). The relation $P_1.R$ can be reformulated using two alternative mappings into $P_2.R$ and $P_4.R$. These contributions are merged by the union operator in the lower left of the query plan. All data retrieved for $P_1.R$ has to be joined with results for $P_1.S$, which can draw tuples from $P_4.S$. Finally, the result of this join-merge needs to be merged with the result returned by $P_3.RS$. Observe *that these tuples cover both $P_1.R$ and $P_1.S$, because of the local-as-view mapping between P_1 and P_3.*

Now we identify redundancies *in the data retrieved by the query plan in Fig. 4.3(b). It is easy to see that the result sets from $P_2.R$ and $P_4.R$ are equal. So one of the two mappings could have been pruned, if P_1 had known that their data contributions are mutually equal. In addition, such an overlap-based pruning can be combined with other techniques taking a cost model into account. In this case, assessing the cost for obtaining the result $\{r_1, r_2\}$ might lead to the decision that accessing P_4 is cheaper, because the data have to be transported only between P_4 and P_1. When the tuples $\{r_1, r_2\}$ are joined with $\{s_1\}$ coming from P_4, we maximally obtain $\{r_1 \sqcap s_1, r_2 \sqcap s_1\}$. So the union right below the query in Fig. 4.3(b) shows overlap as well (namely $\{r_1 \sqcap s_1\}$). However, pruning the result from $P_3.RS$* completely *would result in losing the tuple $\{r_2 \sqcap s_2\}$, which is not part of the overlap. So we have to make a trade-off between the completeness of the query result and cost savings achieved by removing data overlap from the plan, Sec. 4.5. In Chapter 9, we introduce a novel approach to prune mappings partially. Thus, we aim to eliminate overlap while preserving non-overlapping tuples in the query answer of a set of alternative mapping paths.*

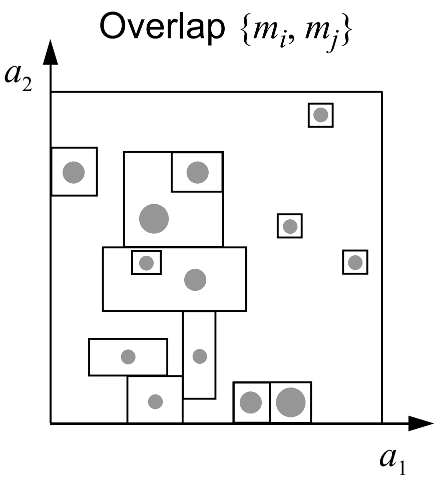
□

Overlap distribution To reduce data overlap during query planning, it is necessary to measure and describe extensional overlap in an aggregated form. To this end, it is interesting to look at the distribution of overlap with respect to the ranges of the attributes of a relation. Consider a multi-dimensional data space made up by the attributes of a relation as dimensions as depicted in Fig. 4.4 on Page 62. It makes a fundamental difference whether the overlapping tuples are scattered equally in this space or whether they occur with higher frequency in certain *regions*.

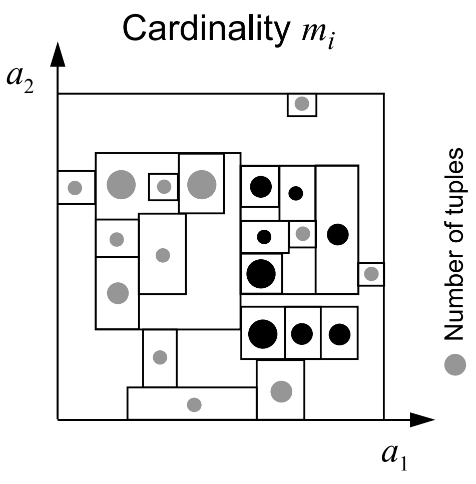
In the first case, the overlap can only be described on a tuple basis. As a consequence, it is very difficult to provide an aggregated description of the extensional overlap between the two relational extensions at hand. In the case that overlapping tuples occur at certain regions within the multi-dimensional space, we have a good chance to describe these regions and adjust further query processing to this information. In Chapter 9 we propose a technique to do so.

We believe that in practice both cases occur. However, observe that selections in mappings and in the user query restrict the corresponding result sets to be located in subareas of the multi-dimensional space. In effect, we can expect much of the overlap between these intermediate result sets being limited to a number of sub-regions that can be handled by our techniques presented in Chapter 9.

The degree of skew in the data distribution accessible over alternative outgoing mappings at a peer also influences the distribution of overlap at that peer. The higher the skew of the data distribution, the higher is the probability of regions with comparably high density of overlapping tuples. The reason for this are redundant mapping paths. As discussed above, data returned by a peer P_j may be transported to a peer P_i along different mapping paths. There, the datasets coming from the *same* originating peer



(a) Overlap distribution of data potentially returned by a pair $\{m_i, m_j\}$ of mappings. The multi-dimensional space is decomposed irregularly by rectangles derived from query feedback. The dots represent the size of the tuple sets in the corresponding cell.



(b) The cardinality distribution of mapping m_i shows that only a fraction of all data at that peer is part of the overlap. Some of these cells are highlighted with a black dot.

Figure 4.4.: Example of a 2-dimensional overlap distribution aligned with corresponding cardinality distribution.

P_j overlap completely. If the data distribution at P_j has a high skew, i.e., there are areas in the multi-dimensional space comprising much more tuples than other areas of the same size, then the overlap distribution at P_i is skewed in the same way as the data distribution at P_j. Observe that detecting and exploiting such situations is exactly our objective.

4.5. Combining Overlap and Completeness

In this section we motivate the trade-off between completeness and overlap minimization. This issue is then taken up in Chapter 9 and considered in our overlap-driven query optimization approach.

Briefly, our approach "removes" an overlapping area in the data space from all but one of a set of alternative mappings in the query plan. This is technically achieved by inserting appropriate selection predicates into the corresponding parts of the local query plan.

There can be situations in which an overlapping area of the data space *also* comprises a considerable part of the result size of that set of mappings, Fig. 4.4. Excluding this area from query reformulation also drops all non-overlapping result tuples in this area, which decreases the completeness of the answer. If this decrease is higher than the cost savings by avoiding the transport of overlapping tuples, the overall efficiency of query answering decreases.

So it becomes clear that we have to consider the completeness of the query plan and cost savings due to excluded overlap at the same time. And doing so is very well possible, because we can exploit the degree of freedom at which of a set of alternative mappings we reduce data overlap. For more details, we refer the reader to Chapter 9.

4.6. Query Execution Cost

To address the challenge of high peer autonomy, our cost model should work fully decentrally. This means that a peer can only use information available locally to estimate the cost of answering a query being sent to a neighboring peer.

Basically, referring to the result size we can identify two different types of cost in PDMS query answering:

– *Variable cost* depend on both the size of the transmitted data set *and* the charateristics of the mapping path they are transported on, such as its length and network bandwidth. Moreover, the result size influences the cost to be spent for computation of the query plan operations.

– *Fixed cost* represent the effort query planning and latency time for establishing network connections for transporting result data. Query planning cost depend on the size of the virtual overall query plan that can be thought of as the "union" of all local query plans at all subsequent peers.

  Latency [OV11] is independent from the size of the query result. Similar to query planning, it depends on the length of mapping paths in an virtual overall query plan.

Both types of cost are part of the query response time. Clearly, the total query response time can be measured locally at a peer. In Chapter 9, we only vary the *number* of queries sent to neighboring peers, for each of which we estimate the result size. So it is of secondary importance how the variable cost depend in detail on the size of the result, the mapping path, and the computation effort. The same holds for the structure of the fixed cost.

Since we want to perform a trade-off between the amount of transported data and the query planning effort influenced by the number of queries, we choose a linear regression model for the total query response time t_Q:

$$t_Q = \underbrace{v \cdot |R|_e}_{\text{variable}} + \underbrace{c \cdot \sum_i |Q_i|}_{fixed} \tag{4.11}$$

with the *estimated* result cardinality $|R|_e$, the sum of the number of predicates of outgoing queries in the plan, a parameter v for calculating the variable cost, and a parameter c for the fixed cost. To find the unknown parameters v and c we repeatedly perform a simple linear regression. In the literature, several much more elaborated cost models have been proposed, e.g., [GRZZ00, ZL96, ROH99, NK01, NGT98]. Since this topic is not in the focus of this thesis, we use a fairly simple cost model and make sure that it suffices for our experiments, i.e., the estimation error for the query execution cost should not lead to wrong decisions in our query optimization approaches.

4.7. Related Work

In this section we review related research in the areas of completeness modeling, data overlap and cost models, apart from the work in [Nau02, NFL04], which we have already discussed.

4.7.1. Data Overlap

The research from Nie et al. [NKN05, NK04, NKH03, NK01] considers overlap in query planning for data integration systems. However, we believe that due to the high redundancy of mapping paths in PDMS data overlap must be exploited more fine grained than Nie et al. do for data integration systems. There, redundancy only arises if data about a certain real-world entity is stored at different sources. Even in such cases it can be too coarse to prune an entire source as it is done in the work of Nie et al.

4.7.2. Cost Model

The work in [GRZZ00, ZRZB01, ZRV^{+}02] employs a multi-dimensional learning technique to predict query response times of Web data sources. Similar to the technique we describe in Chapter 5, their model uses query feedback and can adapt to changes in the behavior of the data sources. As in our work, one important dimension in the cost statistics in [GRZZ00] is the result size of queries. However, it remains unclear from the description in [GRZZ00] how the authors estimate the result size when the response time for a query is to be assessed. As we describe in the following chapter, cardinality estimation for conjunctive queries over skewed data sources is a difficult task.

Finding regression models in a multi-database environment is the subject of [ZL96]. This procedure could principally be applied to PDMS. However, the set of the known parameters would have to be adapted, e.g., the size of the query plan has to be considered when trying to find a trade-off between plan size and overlap removal.

The framework in [NGT98] assumes that a wrapper exports detailed information to facilitate cost estimation. This assumption is less applicable in our setting of high peer autonomy.

4.8. Summary

This chapter identified important measures to drive best-effort query optimization in PDMS. Completeness and overlap serve as the basis for query planning decisions in Part III of this thesis. We showed how the completeness model of Naumann et al. [Nau02, NFL04] can be applied to our PDMS setting.

We discussed why it is important to address data overlap between alternative mapping paths at a peer and proposed to include the overlap distribution over the data space queried into query planning strategies. Such approaches have to consider completeness and overlap in an integrated way. Finally, this chapter reviewed existing cost models for data integration and developed a coarse linear regression cost model used throughout this work.

5. Statistics on Metadata

The processing models presented in the previous section rely on information about completeness and data overlap of neighboring peers. Peers employ these scores to valuate full and pruned query plans and to rank them accordingly. In the following, we propose to gather statistics on completeness and data overlap over neighboring peers locally at each peer.

Density of query results is heavily affected by projections in the schema mappings between the peers, Sec. 4.2.1. Additionally, density suffers from `null`-values in the tuples returned from neighboring peers. In contrast, coverage of results returned from adjacent peers cannot be assessed by looking on the schema mappings. Rather, we need an approach to gather query-dependent information on the actual data stored in the part of the system accessible by a particular peer mapping. This section explains why using query feedback for this purpose is superior to other techniques. Particularly, the autonomy of the peers in the PDMS can be preserved, which is a major prerequisite for the flexibility of this kind of system architecture.

We show how actual query results are examined to build up query-dependent statistics on result cardinalities. To this end, we propose to use self-tuning histograms (ST-histograms[1]) for both cardinality and data overlap assessment. Since each overlap histogram involves a pair of peer mappings and since there can be an exponential number of such pairs at a peer, it is very important to minimize their size to save computing resources. This section describes how to build these histograms with minimal size in a complex query planning context employing both GaV and LaV style schema mappings. Please note that we examine the accuracy of our histograms in Sec. 7.

5.1. Histograms Preserving Peer Autonomy

In this section we first recall the problem of estimating result cardinalities and discuss some improvements to the STHoles histograms of Bruno et al. [BCG01]. Then, we compare several approaches to gather and maintain statistics in PDMS with special emphasis on the requirement of high peer autonomy.

5.1.1. Cardinality Estimation

Our approach of completeness-driven query planning heavily relies on precise and up-to-date statistics about cardinalities of results returned from neighboring peers. Real-world data is usually not uniformly distributed over the multi-dimensional *data space*,

[1]We use the term "ST-histogram" for the class of histograms using query feedback for their adaption and do not refer to the special approach documented in [AC99].

which is made up of the attributes of a relation as dimensions[2]. Instead, the tuples of a relation are distributed with more or less skew. To perform a query-dependent cardinality estimation, we are interested in the so-called *joint data distribution.*

We review some aspects of query result size estimation using multivariate statistics [AC99, GTK01, IMH$^+$04, MMK$^+$05, PI97] and relate them to our PDMS context. This problem deals with estimating result cardinalities for queries containing selection predicates over *different* variables. A common solution is to estimate the result cardinalities for the selections over single attributes separately and to assume attribute value independence. To assess the selectivity of a conjunctive selection predicate, all the individual attribute selectivities are multiplied.

However, it is well known that there may be statistical correlations between the values of different attributes of a relation, for instance functional dependencies. So in practice, the assumption of attribute value independence may lead to very inaccurate cardinality estimates, which can miss the true selectivity by orders of magnitude [FK97, DGR01, IMH$^+$04, MMK$^+$05]. Additionally, we point out that in PDMS it is not possible to apply the uniformity assumption for the distribution of attribute values for cardinality estimation, because a peer neither knows the overall number of tuples a neighbor potentially may return nor it does know the complete range of attribute values.

Since approximating the joint data distribution of a multi-dimensional data set is an important problem in query optimization and approximate query answering, various works proposed solutions to this problem using multi-variate statistics and in particular multi-dimensional histograms. A general survey about histograms can be found in [Ioa03].

To underline the difficulty of adequately estimating cardinalities in PDMS, recall Fig 2.5 on Page 28. The comparison predicates of the user query, the intermediate queries, and the mappings overlap and are accumulated. In PDMS, this effect *repeatedly* may happen along a path of peer mappings. As a result, downstream queries include complex conjunctive predicates. So we face high requirements with respect to accuracy in cardinality estimation. This can only can be addressed by multi-dimensional histograms.

Among other characteristics, approaches to multi-dimensional histograms differ in how they decompose the multi-dimensional data space and how they are set up and maintained. Maintenance can be done in three different ways:

– *Sampling.* A peer systematically poses queries to adjacent peers to explore data potentially accessible over them.

– *Update propagation.* The peers are expected to inform their respective neighbors about any update in their local data stores [HLS06].

– *Query feedback.* Intermediate answers to user queries are exploited to adapt the histograms to the actual data distribution that can be found in the query results [CR94, AC99, BCG01, SHM$^+$06].

[2]Note that this notion of data space is completely different from the more general dataspace introduced by Halevy et al. [HFM06].

These approaches are compared in Sec. 5.1.3. In this work we use the query feedback approach and propose several techniques to ensure accuracy and timeliness of our multi-dimensional histograms on cardinality and data overlap in Sec. 7. There, we provide a mechanism to detect "hidden" changes in the data distribution of a PDMS from a local perspective. Additionally, we describe and solve the problem of trading off the conflicting objectives of using query feedback and pruning the search space during query planning.

Another important facet of the cardinality estimation problem is assessing join result sizes. Instead of complex probabilistic models [GTK01], we use a formula from [NFL04] for estimating cardinality of join results, which is based on a simple probability assessment with the cardinalities of the joined data sets as input. It does not require any assumptions about the information overlap of the join partners. Adopting more accurate techniques for join selectivity estimation is subject to future work.

5.1.2. Self-tuning Histograms: Improved STHoles

We briefly review the multi-dimensional self-tuning *STHoles* histogram proposed by Bruno et al. [BCG01], which we use in an improved form for cardinality and data overlap estimation.

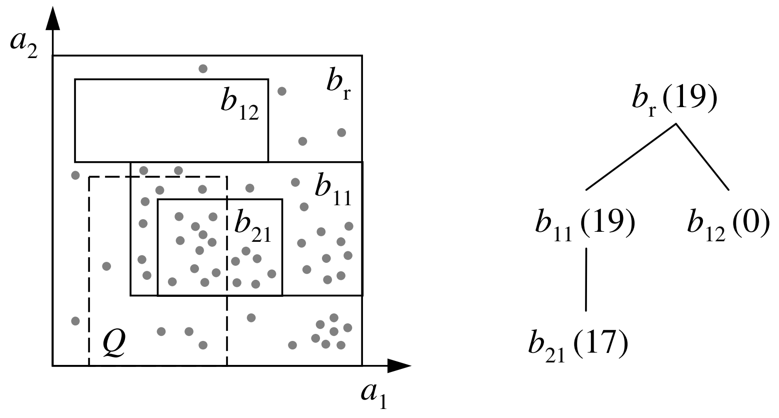


Figure 5.1.: Simple 2-dimensional STHoles histogram [BCG01] with tree structure of buckets and the actual data distribution. The number of tuples measured for each bucket is annotated in brackets.

STHoles consist of a nested structure of rectangular buckets covering subareas of the multi-dimensional data space, Fig. 5.1. Each bucket covers a certain range of attribute values in each dimension and maintains the number of tuples potentially returned for a query exactly selecting that bucket—the so-called frequency. The volume $v(c)$ of a bucket c nested in a particular bucket b is *not* included in b. Rather, the "hole" c maintains its own frequency f(c) and can in turn have own child holes. Regard the STHoles histogram in Fig. 5.1. The root bucket b_r only measures those tuples that are not within the bounding box of any of its children. Consequently, to estimate the result size of the query Q, the overlap of Q with each of the bucket b_i has to be computed and the frequency of each overlap is determined using a uniformity assumption across each intersection, i.e., the contribution of b_i to the cardinality estimation for Q is assumed to be proportional to the volume of $Q \cap b_i$.

This tree structure of buckets of a STHoles histogram is refined by drilling new holes where the actual tuple density obtained from the query answer significantly differs from other parts of a potential parent bucket. An overlap area between the query and a bucket b can have a non-rectangular shape in case it overlaps with any child of b. To obtain a rectangular form, the so-called candidate bucket has to be shrunken. For instance, for the histogram shown in Fig. 5.1 it has been determined that in the area covered by the bucket b_{12} the tuple density is 0, hence it differs from the parent b_r. So a new child bucket b_{12} has been drilled into b_r. The overlap of the query Q with b_{11} has an L shape, because of the child b_{21}. For a new child being drilled into b_{11}, $Q \cap b_{11}$ would have to be shrunken to a rectangular shape. One of our improvements of the STHoles histogram approach concerns which one out of several possible solutions to chose as shrunken candidate hole.

To bound memory consumption and computing resources for maintenance, the number of buckets is limited. A mechanism to merge buckets with similar frequency is used to keep the number of buckets within that limit.

Improvements. We have made several changes to the STHoles mechanism as it was introduced in [BCG01]:

– *Ranking buckets during shrinking.* When a candidate bucket c has to be shrunken to exclude overlap with any of its parent's children, several solutions s_i are possible in general. Bruno et al. propose to chose the shrunken bucket with the "smallest reduction of c" [BCG01]. However, there are cases in which the shrunken candidate bucket s_i with the smallest reduction of volume is not the best solution. Rather, it is more adequate to rank the different solutions according to their difference in tuple density Δd_t with the original candidate bucket c

$$\Delta d_t = \left| \frac{f(c)}{v(c)} - \frac{f(s_i)}{v(s_i)} \right| . \tag{5.1}$$

Observe that this approach requires to evaluate the query representing $v(s_i)$ over the query feedback to be used for updating the histogram. However, as we point out in the next bullet, this helps to further increase the accuracy of histogram updates.

– *Using actual frequency for holes.* As mentioned above, the original approach estimates the frequency $f(c)$ of a candidate hole c by relying on the uniformity assumptions. Especially in histograms with few but comparably large buckets, the tuple density can be quite different across the volume of a large potential parent bucket. Hence, the uniformity assumption can lead to considerable errors for $f(c)$.

Our solution employs query feedback to precisely determine the frequency of the new hole c. This improves consistency as well, because the merge procedure also has to assess the frequency of resulting buckets using the uniformity assumption. This leads to our last improvement.

– *Query feedback for bucket merge.* To overcome the drawbacks of applying the uniformity assumption for histogram update, we propose to maintain the last query feedback result $R(b)$ for each bucket b in the histogram. Thus, even merge procedures can exploit query feedback in computing the frequency of the resulting bucket and the change to the frequency of the parent based on the actual data distribution as gained by the corresponding query feedback. Drilling a new hole c into bucket b requires to "move" the corresponding tuples from $R(b)$ to the query feedback of c. In merging buckets, the query feedbacks of the buckets have to be adjusted analogously.

Since the focus of this work is not on multi-dimensional histograms itself, we did not perform an in-depth analysis of these improvements.

5.1.3. Histogram Maintenance

As mentioned in Sec. 1.2, high autonomy of the peers is an important requirement for information sharing infrastructures. The flexibility of peer-to-peer architectures heavily depends on the autonomy of individual peers. In this section we examine several approaches to gather and maintain statistics as they are listed in Sec. 5.1.1. We focus on the following aspects:

– *Influence on peer autonomy in* PDMS*.* What obligations does the approach require from the peers? Each functional requirement reduces peer autonomy to a certain degree.

– *Adaptability to* PDMS *changes.* As we discuss in Sec. 7, the capability to adapt to changes in the PDMS structure and data distribution over the system is an important discriminator of corresponding solutions. There, we argue that possibly due to comparably simple events, for instance a particular peer goes offline, massive changes in the data distribution can be induced. A first challenge for statistics maintenance in PDMS is to be informed about changes that happen beyond the own neighborhood. To valuate solutions to this difficulty, we introduce the *horizon* of statistics for a distributed information system. It is defined as the maximum length of mapping paths to data the histogram captures.

  Second, the data distribution as it is perceived in a certain peer's statistics has to be updated quickly to reflect the change, although the peer has no direct connection to the part of the PDMS where the change happened.

– *Efficiency of maintenance.* How large is the overhead of a certain approach to update distributed statistics in case of significant changes in the PDMS? This depends on whether the solution requires additional exchange of information for updating statistics throughout the system.

Sampling Neighboring Peers. As we assume that a particular peer only knows its immediate neighbors and we believe that a histogram update should not compromise those peer's autonomy, sampling [LN90, PI97] involves additional queries, which are recursively passed through the system. This expensive processing of sampling queries would have to be done during normal operation time, thus decreasing the overall efficiency of the system. Moreover, the additional cost of sampling is very difficult to predict for an individual peer.

Another drawback is that sampling statistics can become outdated in case of changes in the PDMS. The question is how often to repeat sampling to capture changes in the system quickly. This cannot be answered from the perspective of an individual peer because, as already mentioned, changes can happen beyond its directly accessible neighborhood.

Update Propagation. A different approach to establish statistics about data accessible through peer mappings is the *propagation of updates* as proposed in [HLS06]. There, peers are expected to inform their neighbors about updates in their local data and update messages they receive from their respective neighbors. The peers use this information in their own statistics on the multi-dimensional data distribution perceived over a certain peer mapping.

We believe that in practice peers may not always be willing to pro-actively distribute their updates, i.e., peers likely deny such functionality. Moreover, the approach in [HLS06] is less accurate than using query feedback for histogram tuning, because a certain number of updates is buffered before propagating them. Our results presented in Sec. 7 show that information on changes in the PDMS needs to be processed quickly to adapt the distributed system to these events.

Propagating all updates through the complete system would also not be feasible, because changes involving whole peers would result in transforming and transporting large amounts of data. Therefore, Hose et al. propose to limit the propagation distance of updates by a hop count [HLS06]. This, again compromises peer autonomy in that peers have to attach lineage information on the query result as well as pass this information to other peers. Hence, they would uncover from where the result tuples come and thus impede "trading" data they received from other peers. Moreover, introducing such a limitation actually bounds the horizon of the histograms accordingly. Then, the difficulty arises at each peer how to choose this hop count in a PDMS of unknown extension.

Using Query Feedback. In contrast to the above techniques, using query feedback for gathering and maintaining statistics basically comes for free. Of course, in such an approach only those parts of the data space can be adapted that are covered by queries. However, the workload created by user queries usually suffices to adapt histograms to changes of the data distribution in the PDMS as we experimentally show in Sec. 7.2. Recall that due to the high redundancy of the mapping paths in PDMS every peer has to answer so many queries that we have enough query feedback to update the histograms in the frequently queried areas of the data space. This means that no further functionality is required from the peers, i.e., their autonomy is fully preserved. Compared to the two other approaches, the efficiency of histogram maintenance based on query feedback is also much higher.

The most important advantage is that query feedback reflects the current state of *all* the data accessible over a peer mapping, hence the horizon of the statistics covers all reachable peers. Consequently, if the mechanism for detecting changes beyond a peer's neighborhood is sensitive enough, it can notice modifications of the data distribution even over long mapping paths. To this end, we present a fine grained mechanism for detecting hidden changes in the PDMS in Sec. 7.2. It is based on the estimation error history of individual histogram buckets.

Pruning the search space in query answering as we propose in Sec. 6 raises the difficulty that first the amount of query feedback is drastically reduced and second that changes can be masked by examination of query feedback. We provide a solution for this conflict between maintenance of statistics and cutting down the effort for query answering in Sec. 7.1.

As histogram adaption is computed in a distributed fashion in a PDMS, we employ online refinement in *Humboldt Peers*, i.e., we adapt the histograms immediately after receiving the query feedback to achieve fast histogram adaption rather than collecting query feedback information and updating the histograms only from time to time [AC99].

Taken together, we agree to Aboulnaga and Chaudhuri who claimed in their paper introducing self-tuning histograms [AC99] that techniques using query feedback are especially suitable in settings with distributed data sets, where there is no maintenance window available, in which to perform sampling to update histograms. This especially applies to PDMS because there is no central unit of control.

The comparison of approaches to histogram maintenance in PDMS is summarized in Tab. 5.1.

	Peer Autonomy	Adaptability	Efficiency
Sampling Neighboring Peers	guaranteed	−	−−
Update Propagation	compromised	−	−
Query Feedback	guaranteed	++	++

Table 5.1.: Comparison of approaches for histogram maintenance in PDMS. The scale reaches from very poor (−−) to very good (++).

5.2. Histograms for GaV and LaV Mappings

Computational complexity for cardinality estimation and update of an individual histogram is indeed in linear time. However, the number of pairs of overlap histograms at a peer is exponential in the number of outgoing peer mappings. Therefore, we are interested in keeping the size of the histograms as small as possible to save computing resources and memory. There are several factors influencing the computational effort for histogram maintenance:

– Number of dimensions, i.e., the number of attributes considered in the histogram,

– Maximal number of histogram buckets for the STHoles histograms,

– Update frequency of the histograms.

The latter two of these factors also heavily impact estimation accuracy of the histogram. So we face a conflict of goals that is dealt with in Sec. 7. Here, we focus to limit the number of histogram dimensions to the ones that are absolutely necessary.

As discussed in Sec. 2.3, a local query plan of a peer consists of several layers involving relation predicates, comparison predicates, and the operators $\sqcap$ and $\sqcup$. For our goal to prune mapping paths that contribute only a small amount of data to the result of a user query cardinality statistics for individual mappings are required. Similarly, approaches to exclude overlapping tuples from being transferred unnecessarily need information on data overlap between pairs of mappings. In this section we give an answer to the question which attributes should be considered in building histograms for GaV as well as LaV mappings in the context of the complex query plans at the peers.

A first optimization opportunity is to take into account projections in mappings (Sec. 5.2.1). In some situations, there are further optimization opportunities by combining histograms for different relations into one. This happens if certain properties for the mappings hold. For instance, considering join pushdown (Sec. 5.2.2) can further decrease consumption of computing resources of histograms at a peer.

5.2.1. Mappings Containing Projections

In general, peer mappings include projections as the following example illustrates. To understand how histograms have to be initialized in such cases, we study the interference of queries and mappings. The following example describes a peer mapping containing projections.

Example 10. *Consider the following standard schema H describing hospitals in the disaster management domain:*

$$\begin{array}{l} H.Hospital(HID, Name, Zip) \\ H.Department(DID, HID, LeadingDoc, Specialization) \\ H.Bed(BID, Department, Patient, Status) \end{array}$$

A particular hospital L may have a relation

$$IntensiveBed(BID, Room, Patient, Status)$$

to offer information about beds in intensive care units. The following LaV-mapping relates the standard hospital schema to the above relation of our local hospital:

$$\begin{array}{rcl} L.IntensiveBed(BID, r, p, st) & \subseteq & H.Hospital(HID, n, Zip), \\ & & H.Department(DID, HID, ld, sp), \\ & & H.Bed(BID, Department, p, st), \\ & & HID = 89123, sp = 'IntensiveCare' \end{array}$$

This mapping contains projections in both directions: The variable r in the head of the mapping does not occur in the tail. Vice versa, the tail variables HID, n, Zip, DID, ld, sp, and Department do not appear in the head of the mapping. Observe that this (inclusion) mapping can be used only to reformulate queries over the standard hospital schema H into queries over the schema L. □

Additionally to selections and projections in peer mappings, the intermediate query reformulations may contain selection predicates as well. So, we face the problem to *minimize* the number of attributes considered in a histogram for a given mapping while being able to estimate the result cardinality for *all* possible queries. To find a solution, we investigate the set of possible queries, which may use the given mapping for reformulation. The set of attributes to be considered by a histogram depends on the variables of the selection predicates.

We distinguish GaV and LaV mappings: The single subgoal t in the tail of a GaV mapping $Q(\bar{X}) \subseteq t(\bar{Y}), Sel(\bar{Z})$ is unfolded into the subgoals of the query $Q(\bar{X})$ in the mapping head. The result of the reformulated query of a LaV mapping of the form $h(\bar{X}) \subseteq Q(\bar{Y}), Sel(\bar{Z})$ comprises the attributes $\bar{X}$ of the single head subgoal h. In both types of mappings the head variables are denoted by $\bar{X}$, and $\bar{Y}$ is the set of tail variables. Note that there is no restriction about the variable set of the comparison predicates: $\bar{Z} \subseteq \bar{X} \cup \bar{Y}$.

For setting up a histogram it suffices to consider only the intensional overlap between tail and head of the corresponding mapping (i.e., $\bar{X} \cap \bar{Y}$):

Theorem 2. *Let m be a GaV or LaV mapping of the form*

$$\begin{array}{lr} Q(\bar{X}) \subseteq t(\bar{Y}), Sel(\bar{Z}) & \textit{(GaV)} \\ h(\bar{X}) \subseteq Q(\bar{Y}), Sel(\bar{Z}) & \textit{(LaV).} \end{array}$$

Let H_m be a corresponding histogram that allows cardinality estimation of all possible queries that can be reformulated using m. Then the minimal set of attributes to be covered by H_m to enable estimates for all possible queries are the attributes referenced both in the head and the tail of m, i.e., the variable set $\bar{X} \cap \bar{Y}$.

Proof. In general, the head of m can contain fewer or more variables than the tail. We distinguish three different sets of variables: An individual variable can occur either

- only in the mapping head, $\bar{X} \setminus (\bar{X} \cap \bar{Y})$,
- both in the head and the tail, $\bar{X} \cap \bar{Y}$,
- or only in the tail, $\bar{Y} \setminus (\bar{X} \cap \bar{Y})$.

We show why the first and the last group of variables need not to be considered by a histogram for a GaV or a LaV mapping. Let ψ be a variable mapping from the mapping variables $\bar{X} \cup \bar{Y}$ to the query variables that assigns a particular query variable to each mapping variable.

Queries that have some variables from $\psi(\bar{Y} \setminus (\bar{X} \cap \bar{Y}))$ in their comparison predicates cannot be reformulated by the mapping, because the resulting reformulation would not contain any of these variables. So their corresponding comparison predicates could not be translated. Rather, using the mapping m we can reformulate only queries *maximally* having $\psi(\bar{X} \cap \bar{Y})$ in their comparison predicates.

Variables that are *not* part of the mapping tail, $\bar{X} \setminus (\bar{X} \cap \bar{Y})$, i.e., that occur only in the mapping head, cannot be part of any query to be reformulated using m. Simply, there are no attributes corresponding to these variables in the peer schema(s), to which the relations in the mapping tail belong.

The variables in $\bar{X} \cap \bar{Y}$ have counterparts in the both the head and the tail schema. So comparison predicates in the query and the mapping can be formulated in both schemas. Thus, it is guaranteed that only correct tuples are returned by the peer(s) involved in the mapping head. □

Taken together, the histogram for GaV, LaV, and in general for GLaV mappings is only required to calculate estimates for queries over the attributes corresponding to $\bar{X} \cap \bar{Y}$. These attributes can be found by establishing a mapping from the variables in $\bar{X} \cap \bar{Y}$ to the attributes of the peer schema relations corresponding to the tail subgoal t, in which all these variables occur. Note that Theorem 2 holds independently of any particular query reformulation algorithm.

In Example 10, the intensional overlap $\bar{X} \cap \bar{Y}$ between the mapping tail and head comprises the variables p and st corresponding to the attributes *Patient* and *Status*, respectively. All other variables in the example mapping need not to be considered in setting up the corresponding histogram. So a two-dimensional histogram suffices to cover all possible queries to be reformulated by the mapping given in that example.

Within the context of a query plan, the attribute set $\bar{X} \cap \bar{Y}$ of the intensional overlap between head and tail of a mapping is the schema of the query rewriting resulting from that individual mapping. So we can refine the corresponding Definition 8 from Sec. 2.3.3:

Definition 18 (Schema of the Result of a Mapping)**.** *Let m be a GLaV mapping of the form $Q_h(\bar{X}) \subseteq Q_t(\bar{Y})$. The result of the mapping m has the attribute set $\bar{X} \cap \bar{Y}$.*

5.2.2. Optimization for Join Pushdown

After minimizing the set of attributes for a single histogram we now widen our scope to all histograms at a particular peer. To identify additional opportunities for reduction of the size or the number of histograms, one has to regard the most general query plan for an individual peer. Here, we extend our PDMS model by the notion of a special kind of peer mapping:

Definition 19 (Incoming Mapping)**.** *Let m be a peer mapping of the general GLaV form $Q_h(\mathcal{P}_h) \subseteq Q_t(\mathcal{P}_t)$. Let P be a peer. The mapping m is called an incoming mapping with respect to P, if $P \in \mathcal{P}_h$.*

Intuitively, incoming mappings have the peer under consideration as target. Consequently, they can only be used for query planning by the neighbor peers occuring in the tail of the mapping.

In situations where reformulation using *several* mappings results in a join of relations that reside at the *same* remote peer, the join attribute needs to be reflected only once in the histograms. In fact, a *single* histogram for the join result would be sufficient. However, this optimization assumes that the peer *exclusively* receives such queries that require the join result rather than queries over a participating relation.

A sufficient condition that this condition is *always* fulfilled is that

(1) the peer knows about all of its incoming mappings and

(2) *all* of these mappings are GaV mappings such that the above join is part of their reformulation.

Even if the peer does not know about all of its incoming mappings, it can observe the query workload it receives. If the workload in the past has solely consisted of conjunctive queries, it is quite probable that this also will be the case in the future. Property (2) guarantees that none of the relations involved in the join is ever queried in isolation by another peer. Rather, the peer always receives conjunctive queries. Observe that checking property (2) before setting up the histograms only requires information locally available at a the peer. The following example illustrates this approach.

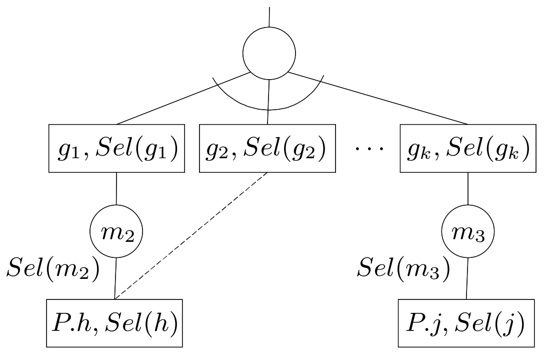


Figure 5.2.: Selection predicates (*Sel*) in the rule-goal tree of a a LaV expansion (extract from Fig. 2.5). Rectangles represent goal nodes (subgoals of queries). Mappings are displayed as circles .

Example 11. *Regard the local rule-goal tree for a LaV expansion shown in Fig. 5.2. The goal nodes $g_1, g_2, \ldots, g_k$ form the conjunctive query to be reformulated at the current peer. These goal nodes are joined as indicated by the arc below the top rule node. The* *goal nodes were created by using the mapping corresponding to this rule node in a GaV expansion. One level below, the LaV mapping m_2 covers g_1 and g_2 and results in a goal node P.h at another peer P. On the far right, the goal node g_k is reformulated into the goal node P.j at the same peer P. Observe that in general there can be more goal nodes in between g_2 and g_k. For now let the goal nodes g_1, g_2 and g_k be the only ones in the rule-goal tree.*

Then, the LaV mapping m_2 and the GaV mapping m_3 completely *reformulate the conjunctive query g_1, g_2, g_k. The result of the query reformulation step involving m_2 and m_3 is $P.h \sqcap P.j$, i.e., the join between the goal nodes P.h and P.j. Since both of these goal nodes represent queries over the* same *peer P, the join calculation can be pushed down to that peer. This means that it suffices to use a histogram for the result of $P.h \sqcap P.j$ as discussed in more detail below.*

Since the goal nodes g_1, g_2 and g_k from the above example are joined, their corresponding relations must pairwise share a join attribute. For instance, let the join attribute of g_2 and g_k be a_k. Then, the attribute a_k of the extension of the (natural) join $P.h \sqcap P.j$ only contains values that occur both in the extension of g_2 and the tuple set of g_k.

If we create a histogram for both m_2 and m_3, the join attribute a_k of $P.h \sqcap P.j$ would occur in both of these histograms. Moreover, some parts of the value ranges of a_k covered by these histograms never occurs in the extension of $P.h \sqcap P.j$. Hence, this approach unnecessarily occupies memory for the join attribute in both histograms and for covering value ranges of a_k, which are guaranteed not to be part of the final query answer.

Another drawback is that actually pushing down joins, e.g., $P.h \sqcap P.j$, also hides update information from the histograms of the individual mappings, here m_2 and m_3. Since each of those mappings can potentially return more tuples than $P.h \sqcap P.j$ if queried in isolation. Note that this can happen, if a user poses a query only over the relations g_1 and g_2 that is rewritten using only m_2 or a query over g_k that is reformulated using only m_3.

Usage of computing resources, memory consumption, and estimation accuracy can be improved by creating only a single histogram for the join $P.h \sqcap P.j$. In this approach, the join attribute a_k is represented only once. Additionally, the accuracy of the histogram is higher, because the same number of buckets as in the naive approach are used to exclusively cover the ranges of a_k, which are guaranteed to occur in the result of the join.
$\square$

Substituting histograms for single mappings by a single histogram for their reformulation result requires that it is guaranteed that the peer is expected to estimate cardinalities only for the conjunctive query completely covered by these mappings, instead of individual subgoals of that conjunctive query. For instance, the peer in the above example solely could perform cardinality estimation for the conjunctive query g_1, g_2, g_k if only a histogram for the result of the mappings m_2 and m_3 is available.

5.3. Histograms for Information Overlap

Our goal in this section is to build histograms for every situation in the local query plan at a peer where alternative subresults are combined. To this end, we first recall the notion of information overlap. Then, we examine the most general structure of the query plan with respect to overlap. Based on these findings, we are able to decide how to create the overlap histograms that can be used for all possible queries over a given peer schema. Finally, the usage of the overlap histograms in query answering is explained.

Information overlap has two dimensions as discussed in Sec. 2. Intensional overlap means that two data sets have at least one attribute in common. As [NFL04], we require that two overlapping data sets at least have one common attribute a_k. This attribute a_k can either be a key for both data sets or a foreign key for one data set and a key for the other. Extensional overlap requires that there are actually tuples in the different data sets that have an identical value for a_k.

5.3.1. Information Overlap in Query Plans

As discussed in Sec. 2.3, query plans reflect several alternatives to rewrite a query at hand. The following example serves as a starting point to study at which positions in the local query plan overlap has to be considered.

Example 12. *We take up Example 2 on the emergency hospitalization of three hospitals J, K, and L from page 25. The rule-goal tree in Fig. 5.3 displays the state after expanding the subgoals of the following query:*

$$q :- K.Patient(SSN, Name, AdmID), K.Admission(AdmID, Adress, Date).$$

This query can first be partially answered by using the mapping m_1 to J.Patient. Second, both m_2 and m_3 can individually be used to obtain further partial answers to the query. Finally, combining *the two mappings m_2 and m_3 in a join-merge $L.Patient \sqcap L.Intake$ also returns result tuples for the above query. In terms of Sec. 2.3, these tuples are the result $R(\{m_2, m_3\})$ of the mapping set $\{m_2, m_3\}$.* □

Now we widen our scope to the general query plan as it results from the transformation of the rule-goal tree. The local query plan P of a peer is a full outerjoin-merge $\sqcup$ of rewritings, each of which returns tuples that satisfy the query Q the peer is about to answer:

$$P(Q) = \bigsqcup_i (C_{i1} \sqcap C_{i2} \sqcap \ldots \sqcap C_{ik}) \,. \tag{5.2}$$

This is the first level on which the operator $\sqcup$ occurs in the query plan. So we have to consider overlap between tuples resulting from different query rewritings $Q'_i = C_{i1} \sqcap C_{i2} \sqcap \ldots \sqcap C_{ik}$.

Each of the terms C_{ij} in the rewritings can combine alternative contributions for the query subgoals covered by that rewriting subgoal. Each of these alternative contributions is a result of a *set* of mappings $\mathcal{M}_l$ along with the corresponding selection predicates $Sel(\mathcal{M}_l)$. All results of mapping sets that are in the same C_{ij}, have to be combined by a full outerjoin-merge $\sqcup$:

$$C_{ij} = \bigsqcup_l (R(\mathcal{M}_l), Sel(\mathcal{M}_l)) \,. \tag{5.3}$$

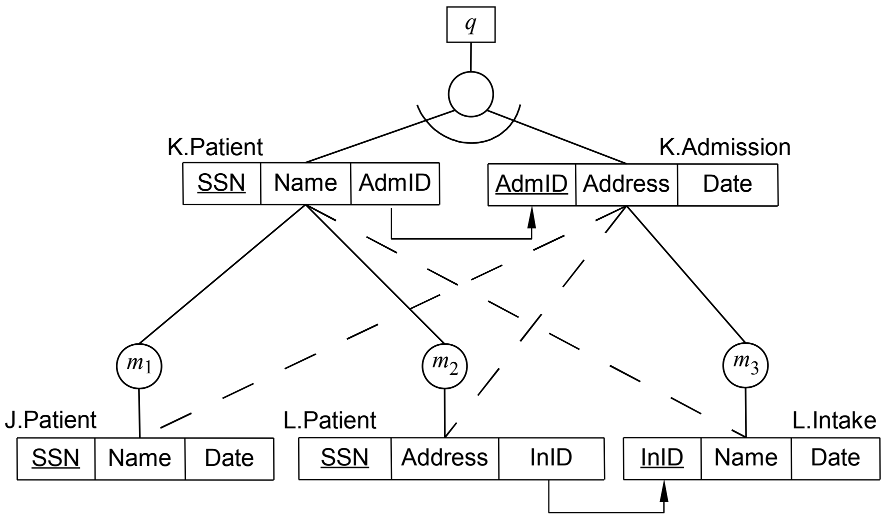


Figure 5.3.: Example for a rule-goal tree with two LaV expansions involving three mappings.

Hence, this is the second occurrence of ⊔ in the query plan. Observe that in many cases a mapping set consists only of an individual mapping.

In general, we aim to assess the size of the extensional overlap by an overlap histogram for each combination of terms connected by ⊔ in both situations it can occur:

– Each pair $(T_m, T_n), m \neq n$. In practice, there is usually a small number of such pairs, especially if the number of relations in a peer schema is small. But the larger the peer schema, the more relations can be involved in a query to the peer and the more different groups of query subgoals—reflected in different T_i—are possible.

– Each pair of results of mapping sets $(R(\mathcal{M}_p), R(\mathcal{M}_q)), p \neq q$. The number of different pairs of this kind is dependent on the number of peer mappings a peer has, i.e., the rank of the peer.

Principally, the number of overlap histograms necessary to capture all possible pairs of alternative subresults in a query plan is exponential in the parameters mentioned above [FKL97]. Therefore, it is crucial to minimize the size of each overlap histogram to save memory and computing resources for their maintenance.

5.3.2. Overlap Histograms

Since overlap between results of mapping sets is much more common in practice than overlap between complete rewritings, Equation (5.2), and the potential savings are much higher we focus on the former. Overlap between the top level terms of a query plan can be treated similarly.

We turn our attention to pairs of mappings possibly containing projections and discuss which attributes of the peer schema have to be included in overlap histograms. As in

Sec. 5.2, we are interested to initialize the overlap histograms such that overlap can be estimated for all possible queries a peer receives. Since the local rule-goal tree at a peer consists of both GaV and LaV expansions, we have to find out for which pairs of mapping sets an overlap histogram has to be created.

Comparing Mapping Results. We align the tuple sets returned by the mapping sets for each pair of mapping sets. Please recall that these tuple sets are the results of the respective mapping sets, Def. 9. As mentioned above, we require those tuple sets to be compared for overlap to have a common key attribute. Then, two tuples are regarded as overlapping, if their key attributes have the same value. Otherwise, the tuples correspond to different real-world entities. This task is straightforward for GaV mappings concerning the same relation of a peer schema.

However, comparing GaV and LaV mappings is more difficult. First, we study LaV mappings in more detail. Key attributes of peer schema relations covered by a LaV mapping m are only a key for the result $R(m)$ of the mapping, if they are *not* involved as a target in a foreign-key relationship as highlighted by the following example.

Example 13. *Consider the two relations $P_1.R(\underline{a}, b)$ and $P_1.S(\underline{b}, c)$ connected by a foreign-key relationship over attribute b, a relation $P_2.RS(\underline{a}, b, c)$, and the LaV mapping*

$$P_2.RS(u, v, w) \subseteq P_1.R(u, v), P_1.S(v, w).$$

Since b is the target of a foreign-key relationship, the result of the LaV mapping, namely $P_2.RS(\underline{a}, b, c)$, and $P_1.S(\underline{b}, c)$ do not share a common key attribute although both tuple sets include values for the attribute b of the relation $P_1.S(\underline{b}, c)$. The attribute b is no key in $RS(\underline{a}, b, c)$. □

This observation leads to a general condition for the results of a GaV and a LaV mapping to be compared for overlap.

Theorem 3. *Let m_1 be a GaV mapping of the form $Q_h \subseteq T$ and m_2 be a LaV mapping of the form $H \subseteq T_1, T_2, \ldots, T, \ldots, T_n$, i.e., both mappings can potentially be used to answer a query for T. Let none of the mappings project out any variable that corresponds to a key attribute of a schema relation. Then, it is only possible to test the results of m_1 and m_2 for overlap, if the key of the schema relation the predicate T matches to is also a key of the relation the mapping head H matches to.*

Proof. According to the conditions of the theorem, all variables corresponding to a key attribute of a schema relation are part of the results of the mappings. The results of two mappings can be checked for overlap, if their corresponding schema relations have the same key. Hence, the check is possible, if T and H share the same key attribute.

As demonstrated in the above example, the mappings cannot be checked for overlap, if the key attribute of the schema relation T corresponds to a target of a foreign-key relationship in the tail of the LaV mapping, here $T_1, T_2, \ldots, T, \ldots, T_n$. In that case, the key attribute that T corresponds to is no key in the result of the LaV mapping. □

Notice that it is possible that result sets of the queries in the head and the tail of a mapping can have different keys, i.e., variables corresponding to key attributes can be projected out. For such cases a more sophisticated method to compare tuples is necessary, for instance duplicate detection. However, with such techniques, instance data has to be included into the examination for overlap. In such situations, the overlap cannot be determined using solely schema and mapping information. We do not cover these situations in this thesis.

Histogram Dimensions. As stated above, our goal is to find the minimal set of attributes to be considered in an overlap histogram $H_{\mathcal{M}_1,\mathcal{M}_2}$ for a certain pair of mappings sets $(\mathcal{M}_1, \mathcal{M}_2)$ such that it can be used for *all* possible queries involving $(\mathcal{M}_1, \mathcal{M}_2)$. Similarly to Sec. 5.2, we examine which queries have to be treated by the histogram and which do not. The result is that the set of attributes to be considered in $H_{\mathcal{M}_1,\mathcal{M}_2}$ corresponds to the intensional overlap between results $R(\mathcal{M}_1)$ and $R(\mathcal{M}_2)$ of the mappings sets:

Theorem 4. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two mapping sets with their results $R(\mathcal{M}_1)$ and $R(\mathcal{M}_2)$ respectively. Let $Attr(R(\mathcal{M}))$ be the set of attributes of $R(\mathcal{M})$, i.e., the schema of a result of a mapping set. Then the minimal set of attributes to be considered in the overlap histogram $H_{\mathcal{M}_1,\mathcal{M}_2}$ for $(\mathcal{M}_1, \mathcal{M}_2)$ such that it can be used for all possible queries combining $\mathcal{M}_1$ and $\mathcal{M}_2$ by a full outerjoin-merge $\sqcup$ is $Attr(R(\mathcal{M}_1)) \cap Attr(R(\mathcal{M}_2))$.*

Proof. The attribute set $Attr(R(\mathcal{M}))$ of a result of a mapping set is the union of the attributes of the results of the individual mappings m_i in $\mathcal{M}$, because the join-merge operator $\sqcap$ in Def. 9 keeps all the attributes of the results of the m_i:

$$Attr(R(\mathcal{M})) = \bigcup_i Attr(R(m_i)) \tag{5.4}$$

Combined with Theorem 2, it follows that $\bar{A}_1 = Attr(R(\mathcal{M}_1))$ and $\bar{A}_2 = Attr(R(\mathcal{M}_2))$ contain the candidate attributes for $H_{\mathcal{M}_1,\mathcal{M}_2}$. Let ψ be a variable mapping from the mapping variables of $R(\mathcal{M}_1)$ and $R(\mathcal{M}_2)$ to the query variables assigning a particular query variable to each mapping variable.

Analogously to Theorem 2, the attributes in $\bar{A}_1 \setminus (\bar{A}_1 \cap \bar{A}_2)$ need not be considered in $H_{\mathcal{M}_1,\mathcal{M}_2}$, because queries involving selections over $\psi(\bar{A}_1 \setminus (\bar{A}_1 \cap \bar{A}_2))$ cannot be answered by $\mathcal{M}_2$. Due to symmetry between $\mathcal{M}_1$ and $\mathcal{M}_2$ the same holds for $\bar{A}_2 \setminus (\bar{A}_1 \cap \bar{A}_2)$ and $\mathcal{M}_1$. In contrast, queries with selection predicates over $\psi(\bar{A}_1 \cap \bar{A}_2)$ can be answered by both sets of mappings. $\square$

Intuitively, the above theorem means that the overlap histogram only needs to comprise the attributes for which selection predicates can be translated by *both* of the given mapping sets.

Example 14. *We continue Example 10 from Page 74 and add a second mapping m_2 establishing a relationship to beds and patients at another hospital L:*

$$\begin{array}{lcl} L.Bed(BID, PatientSSN, Building) & & \\ L.Patient(SSN, Name, Age) & & \\ m_2 : L.Bed(BID, p, b), L.Patient(p, n, a) & \subseteq & H.Bed(BID, DID, p, st), \\ & & HID = 65432 \end{array}$$

We have a simple example with $\mathcal{M}_1 = \{m_1\}$ and $\mathcal{M}_1 = \{m_2\}$. The attributes of the result of the mapping m_1 from Example 10 were $Attr(R(m_1)) = \{BID, PatientSSN, Status\}$, whereas we have $Attr(R(m_2)) = \{BID, PatientSSN\}$ for m_2 above. Consequently, the overlap histogram $H_{\mathcal{M}_1,\mathcal{M}_2}$ has the dimensions

$$Attr(R(\mathcal{M}_1)) \cap Attr(R(\mathcal{M}_2)) = \{BID, PatientSSN\}.$$

The attribute Status $\in Attr(R(m_1))$ is no dimension of the overlap histogram, because m_2 cannot answer queries with selections over Status. □

An example of an overlap histogram with a comparison of the corresponding data distributions is depicted in Fig. 5.4.

5.3.3. Using Overlap Histograms

This section explains how to use the overlap histograms to estimate overlap between data resulting from the corresponding mapping sets. We denote the rewritings resulting from applying the two mapping sets $\mathcal{M}_1$ respectively $\mathcal{M}_2$ on a query Q with $Q'_{\mathcal{M}_1}$ respectively $Q'_{\mathcal{M}_2}$. These rewritings occur in the C_{ij} in Equation 5.3. Hence, they have the form

$$Q'_{\mathcal{M}_i} = R(\mathcal{M}_i), Sel(\mathcal{M}_i) \tag{5.5}$$

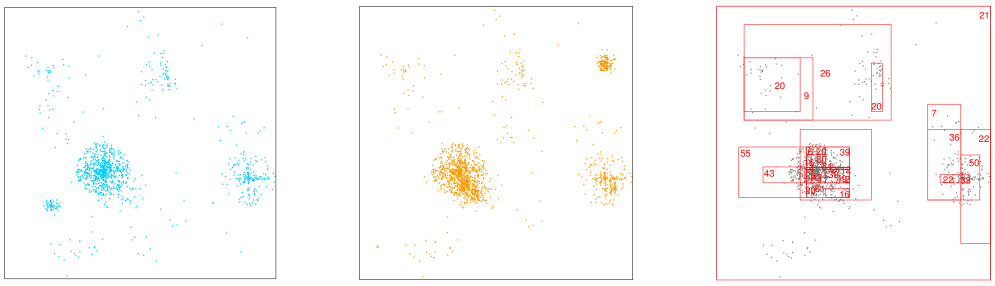
consisting of the result and the selection predicates of the mapping set. Given the rewritings $Q'_{\mathcal{M}_1}$ and $Q'_{\mathcal{M}_2}$, we are interested in the set $\mathcal{S}$ of selection predicates to be used for overlap estimation with the given histogram $H_{\mathcal{M}_1,\mathcal{M}_2}$.

Since we assume overlapping tuples returned by $Q'_{\mathcal{M}_1}$ and $Q'_{\mathcal{M}_2}$ to be equal in the values of all common attributes, the overlapping tuples fulfill the selection predicates of *both* rewritings. Consequently, we find that

$$\mathcal{S} = Sel(\mathcal{M}_1) \cap Sel(\mathcal{M}_2). \tag{5.6}$$

5.4. Related Work

Related work for this section comes from two fields: Multivariate statistics for selectivity estimation in centralized databases and efforts to capture statistics on overlap between information sources. To the best of our knowledge, there is no research that investigated the the difficulties in applying multi-dimensional statistics on the comparably complex and volatile setting of Pdms for query-dependent cardinality and overlap estimation.



(a) Data distribution of mapping set $\mathcal{M}_1$.

(b) Data distribution of mapping set $\mathcal{M}_2$.

(c) Overlap histogram for the mapping sets $\mathcal{M}_1$ and $\mathcal{M}_2$.

Figure 5.4.: Example of an overlap histogram with the data distributions of alternative mapping sets. The numbers show the tuple frequency of the buckets.

5.4.1. Multivariate Statistics

In this section we discuss related work covering multivariate statistics. Multi-dimensional histograms were introduced in [PI97] to overcome potential high inaccuracies of one-dimensional histograms in conjunction with the attribute value independence. Especially in the PDMS context, where peers deal with completely unknown data sources it is very probable that the independence assumption may lead to highly inaccurate estimations. As creating optimal multi-dimensional histograms is NP-hard [MPS98], a variety to approaches has been proposed to achieve accurate approximations. In the following, we focus on self-tuning approaches.

The first proposal on using query feedback for maintenance of statistics was [CR94]. This paper deals with one-dimensional statistics. Aboulnaga and Chaudhuri [AC99] exploit query feedback to maintain multi-dimensional histograms. The extensive experimental study in that work reveals that self-tuning histograms are feasible for low-dimensional data sets and medium skew in the underlying data. They also proved to be sufficiently robust against a biased tuning workload as well as against significant database updates. These characteristics make them well suited for application in highly dynamic PDMS. An important drawback of the rectangular decomposition of the multi-dimensional data space used in [AC99] is the very high memory and CPU consumption for high-dimensional datasets, e.g., with more than 4 or 5 dimensions.

Chaudhuri et al. overcome this obstacle with their STHoles histograms [BCG01] that we use in an improved version in this thesis. However, due to their uniformity assumption for an individual histogram bucket, query feedback is not exploited to the full extent. In our version of STHoles, we derive the exact tuple density for a new hole by querying the query feedback to be used for the histogram update. By keeping the most current query feedback for each bucket in the histogram, it is also possible to perform merges of buckets with higher accuracy than in the original STHoles histogram, Sec. 5.1.2. The estimation accuracy of STHoles histograms for high-dimensional data of with 4 or more dimensions is not sufficient in some cases.

To improve accuracy and consistency of selectivity estimation especially for high-dimensional data sets, Markl et al. apply the principle of maximum entropy (ME) to this problem [MMK+05, MHK+07]. This approach exploits all (possibly partial) information on the data distribution of the data set at hand. Principally, it is possible to apply the ME approach to improve accuracy and consistency of self-tuning histograms for high-dimensional data. The maximum entropy approach is also proved to adapt readily to changes in the underlying data [SHM+06].

The approach to assess join selectivity in [GTK01] goes much beyond the completeness model used in this work. However, it requires intensive analysis of dependencies in the underlying data to establish probabilistic models and is therefore less appropriate in the heterogeneous and dynamic environment of a PDMS.

The *Semantic Gossiping* approach of Aberer et al. uses cycles in mapping networks to examine loss of information [ACMH03]. That is, instead of explicitly modeling completeness as in our approach, the authors use instance sampling to assess information quality criteria. The authors use a simple data model and define mappings only between attributes. Moreover, their work does not cover selection queries.

The basic idea of using histograms for query planning in a P2P system in [PKP04] is quite similar to ours. However, that work only uses 1-dimensional histograms in conjunction with the attribute value independence assumption. Additionally, in contrast to PDMS, every peer in a P2P system has the same peer schema. Hence, the approach in [PKP04] does not deal with creating histograms in the context of a complex query plan comprising GaV and LaV style rewritings. The authors also do not discuss how to adapt their histograms to changes in the system.

The work in [NKN05, NK04, NKH03, NK01] is closely related to our approach. Nie et al. employ techniques from data mining to collect multivariate statistics for coverage and overlap in a data integration scenario. They also provide a best effort approach as our research does. Similar to this thesis, their systems BibFinder and StatFinder [NKH03] assume that the autonomous data sources do not export statistics on their coverage and overlap with other sources. The approach of the StatMiner component to learn the statistics is based on identifying query classes with a necessary level of granularity. So in fact, they also follow a query feedback approach. A difference to our work is that they only consider queries frequently occuring for their statistics. In contrast, our goal is to provide sufficient estimation accuracy for *all* random queries.

Nie et al. claim that the loss of accuracy by choosing the query classes too coarse grained is not critical for their application, because they are primarily interested in relative coverage statistics to rank data sources accordingly. Since we want to use cardinality and overlap estimates together with a cost model to assess the efficiency of querying certain neighboring peers on a finer grained level, Sec. 9, we are interested in absolute, highly accurate estimates. Furthermore, as reported in [NK04], Nie et al. measure estimation accuracy by looking on the *average* error of a whole query workload. In this thesis we examine how the estimation accuracy changes over time and thus over the query workload. This is necessary to avoid that worse query planning decisions are made based on the cardinality and overlap estimations. This is particularly probable for queries referring to areas in the data space that are either seldom covered and thus receive low attention in the approach of Nie et al. or that differ drastically in their tuple density from their surroundings.

The experiments that Nie et al. present deal with low dimensional data sets. Usually, they use two dimensions. We believe that their approach to represent multivariate statistics does not promise to scale well for a higher number of dimensions. This is also an issue for the STHoles histograms that we use, but they can easily be improved by employing the maximum entropy approach proposed in [MMK^{+}05].

5.4.2. Overlap Statistics

The approach in [FKL97] basically has the same goals as our work. It builds a probabilistic model on coverage and overlap between information sources in a mediator. The model represents the probability distribution over a set of collections of a mediated schema. Using this model their system can assess the coverage and overlap for a given user query. The authors point out that their probabilistic model cannot consider arbitrary constant values of user queries. Rather, these values are mapped to the collections of the mediated schema. In fact, this means that their mediator is not able to make estimations for queries containing selection predicates on continuous ranges of attributes like our histogram-based approach can. Moreover, unlike to our approach, the solution in [FKL97] assumes uniform data distribution and independence of attributes and is therefore not capable to deal with highly skewed data distributions.

The work of Nie et al., e.g., [NK04], discussed in detail in the previous section also addresses the problem of gathering overlap statistics to rank sources in data integration. Similar to our approach, they treat statistics for coverage and data overlap by the same technique.

The work in [BMT^{+}05] comes from the area of information retrieval and aims to measure overlap between document collections with respect to the keywords they comprise. Minerva is a P2P system for distributed Web search. In their approach, they have also to reconcile the goals of high accuracy and scalability. A limiting factor for scalability is the size and number of messages on their local indexes exchanged between the peers. Although the maintenance of index information in Minerva is actually global, they face to similar conflicts as we discuss in Sec. 7.

5.5. Summary

The focus of this chapter was on defining multi-dimensional histograms in the context of complex query plans consisting of both GaV and LaV style query rewritings. We proposed several improvements for STHoles histograms to obtain higher estimation accuracy and better consistency of the statistics. Then, we compared different approaches for collecting and maintaining statistics in PDMS with respect to preserving peer autonomy, adaptability to massive changes in PDMS, and their efficiency. We found that using query feedback for maintenance of statistics is clearly superior to sampling techniques or update propagation.

We examined how to build histograms of minimal size both for cardinality and overlap estimation for query rewritings involving peer mappings containing projections. For the purpose of overlap estimation is was necessary to identify occurrences of the union-like full outerjoin-merge operator in the general form of a query plan local at a peer.

Self-tuning histograms as described in this chapter are a solid basis for completeness-driven as well as overlap-aware query optimization as discussed in the following chapters. STHoles histograms are extensively used throughout our experiments.

Part III.

Efficient Query Answering in PDMS

6. Completeness-Driven Query Planning

To implement large-scale data sharing using PDMS in a flexible and efficient way, pruning the search space of query planning is crucial. The main contribution of this thesis is to optimize query planning based on completeness-related metadata gathered locally at the peers. The first two parts of this work have introduced solutions for query answering and maintaining corresponding metadata on benefit and cost in the flexible and volatile context of a PDMS.

In this part we employ these metadata and propose new techniques for completeness-driven query optimization. First, we show the potential of pruning the search space based on estimations of the completeness of results returned from neighboring peers. Then, Chapter 7 proposes solutions to keep the statistics necessary for completeness estimates up to date in presence of changes as well as pruning. The budget-driven algorithms of Chapter 8 employ a simple cost model together with cardinality estimations to both optimize and limit resource consumption of query answering. Finally, in Chapter 9 we propose a novel approach to exploit query-dependent information on data overlap between alternative peers to introduce another dimension of query optimization.

This chapter first explains the sources of information loss in PDMS query answering and motivates our first pruning approach. Next, we describe how to use the cardinality histograms introduced in Sec. 5 to identify mapping paths that contribute only few data. Then, we present our threshold-based algorithm for pruning query plans and report on its experimental evaluation.

As explained in Sec. 2.5, information loss along a mapping path means that data can be filtered out by *projections* in peer mappings after they were transported between several peers before. Since these data do not contribute to the query result, transporting them even to the location where they are dropped is useless. However, the effort for query reformulation and data transport already has been spent. So efficiency of query answering is unnecessarily decreased. Observe that this effect only occurs if the query language does *not* support projections. We believe, that for heterogeneous environments this is a realistic assumption. For instance, in many query interfaces in the Web, the user usually cannot choose the set of attributes in the result. If the query language would support projections in queries a peer could project out attributes from a foreign peer schema that are not mapped to its own schema.

Selections usually accumulate along mapping paths during query planning. In effect, selection predicates in the query or in any mapping along a mapping path are applied to the data stores. Hence, there is no unnecessary transport of data due to selections in peer mappings as long as every peer in the path supports selections in the query language. If some peers do not accept selections for some attributes of a query, this leads to similar problems as for projections.

6.1. Assessing Subplan Utility

Results of mapping sets are the smallest elements in query plans and they are combined by a full outerjoin-merge into a subgoal of a query rewriting (Fig. 2.6 on page 31). As described in Sec. 4.4, all of these results of mapping sets maximally contribute the same set of attributes to the rewriting subgoal, i.e., they overlap intensionally. Additionally, the results of the mapping sets can overlap extensionally. Therefore, we can expect that pruning away a particular result will not reduce the overall completeness of the query plan too much, because it is probable that a considerable part of the results pruned away will be part of alternative results.

As stated in [Nau02], for the maximization of the completeness of a query plan of an integrated information system, e.g., a peer, the principle of optimization does not hold. This means that adding the result from a mapping set with higher completeness than another one does not necessarily increase the completeness of the resulting plan more than the mapping set with the lower completeness. This shows that the utility of a mapping set for a query plan cannot be valuated in isolation. Rather, the result of each mapping set must be considered in the context of the virtual overall query plan. However, this is difficult in a PDMS query plan that consists of a mixture of GaV and LaV mappings. The number of possibly incomplete query plans at a peer is exponential in the number of query subgoals s and the number of mappings m contributing to each query subgoal. Assuming that these numbers are known, the size of the search space amounts to $2^{s \cdot m}$.

To the best of our knowledge, there is no approach that considers any information quality criteria *during* the creation of a query plan involving both GaV and LaV mappings. The work in [NLF99] uses a query planning approach dealing with GLaV-like mappings and exploits information quality on the data sources *sequentially.* The authors state that—for simplicity—they separated the query planning step from the quality-based ranking of data sources and plans. Additionally to the results in [NLF99], Naumann presented an integrated method for quality-aware query planning in a mediator setting with LaV mappings [Nau02, NFL04]. However, in that work each LaV mapping is used in isolation.

In contrast to the approach in [Nau02], that aims to find the best query plan *without* exhaustively enumerating all query plans, we assume that at a peer it is usually possible to compute the query plan to its full extent. Unlike the setting of a single mediator being directly connected to a very high number of sources, in a PDMS we can assume that the number of neighbors of a peer is usually quite small, i.e., less that 10. This is due to the distributed nature of a PDMS. A particular peer only needs to know a few neighbors. Yet, through the recursively branching network of peers most of them have indirect access to a large part of the system.

6.1.1. Estimating Subplan Completeness

To valuate the utility of the result of a mapping set, we can refer to the fully expanded query plan denoted by $\hat{P}$. Using the completeness model presented in Sec. 4, we assess the completeness of all pruned plans derived from $\hat{P}$ from the local perspective of the peer under consideration. The completeness model requires the following input data for each mapping set $\mathcal{M}_l \in \hat{P}$:

– Density scores for all attributes of the corresponding rewriting subgoal C_{ij},

– Coverage of the result $R(\mathcal{M}_l)$.

In the following, we describe how to gather these data.

Density. For simplicity, we assume the attributes of the data returned by the neighboring peers have full density:

$$\forall a \in \mathit{Attr}(R(\mathcal{M}_l)) : d(a) = 1. \tag{6.1}$$

Of course, this is usually not the case because of projections in the peer mappings along mapping paths. Projections in the mapping to the neighboring peer are considered in our approach since the peer at hand knows about them. In experiments that have a significant fraction of peer mappings with projections, we track how this assumption increases the estimation error for the completeness of pruned query plans. It would be possible to overcome this assumption by introducing accurate statistics on the attribute densities of each mapping set in the query plan, e.g., by multi-dimensional histograms analogously to the cardinality histograms. But this is not in the focus of this work.

Projections in mappings of $\mathcal{M}_l$ on the one hand means that the result of the mapping set comprises *fewer* attributes than the corresponding rewriting subgoal:

$$\mathit{Attr}(R(\mathcal{M}_l)) \subset \mathit{Attr}(C_{ij}).$$

This is reflected in the input for the completeness calculation by setting the density of these attributes to 0:

$$\forall a \in (\mathit{Attr}(C_{ij}) \setminus \mathit{Attr}(R(\mathcal{M}_l))) : d(a) = 0. \tag{6.2}$$

On the other hand, projections lead to unnecessary data transport. This is the case, if the result of the mapping has *more* attributes than the query subgoal it is covering. Of course, this does not decrease the completeness of the mapping result. Rather, the ratio of completeness to cost, i.e., the efficiency, is worse than it would have been if these extraneous attribute values had not been transported.

Coverage. A second type of input information for the completeness model are the coverages of the results of the mapping sets in $\hat{P}$. Recall that the result of a mapping set is computed by a join-merge between the results of the included mappings. During query planning, our histograms (Sec. 5.1) provide the peer with query-dependent cardinality estimates $|(R(m_k)|$ for the results of the mappings referred to in $\mathcal{M}_l$. By normalizing them using a constant factor, e.g., the estimated size of the world $|w_{m_k}|$, an estimate for the coverage of the result set can be obtained:

$$c(R(m_k)) = \frac{|(R(m_k)|}{|w_{m_k}|}. \tag{6.3}$$

Please note that the size of $|w_{m_k}|$ can be arbitrary for a particular mapping at a particular peer. So it is not necessary to exactly know the size of the world. For instance, $|w_{m_k}|$ can be chosen upon the initialization of the cardinality histogram of the mapping m_k. The coverage $c(R(\mathcal{M}_l))$ can be calculated as usual using Eqn. (4.5).

Based on the attribute densities of the rewriting subgoals and the coverages of the results of the mapping sets the completeness $C(\hat{P})$ of the fully expanded query plan $\hat{P}$ or any of its incomplete subplans can be computed using the formulas in Sec. 4.2.

6.1.2. Identify Pruning Candidates

Our goal is to assess the utility of each mapping set $\mathcal{M}_l \in \hat{P}$ to decide whether the mappings in $\mathcal{M}_l$ are worth following during further query planning. As we argued above, the potential data contribution of a mapping set has to be determined in the context of the local query plan. In [RNHS06], we have proposed to quantify the potential data contribution of a mapping set $\mathcal{M}_l$ by comparing the completeness

- of the local query plan exploiting *all* sets of outgoing mappings and
- of the plan *without* that mapping set.

Intuitively, this approach assesses the impact of the data contribution of $\mathcal{M}_l$ on the query result of the peer. This approach is formalized as follows:

Definition 20 (Potential Data Contribution ΔC)**.** *Let $P(Q)$ be a query plan to answer the query Q. Let $R(\mathcal{M})$ be the result of a mapping set $\mathcal{M}$ contributing to any of the rewriting subgoals in P. The query plan resulting by removing $R(\mathcal{M})$ from P is $P' = P \setminus \langle R(\mathcal{M}_l), Sel(\mathcal{M}_l)\rangle$. Then, the potential data contribution of $R(\mathcal{M})$ in the context of P amounts to*

$$\Delta C(P, R(\mathcal{M})) = C(P) - C(P') \tag{6.4}$$

This definition can be applied analogously for the the data contribution of a rewriting subgoal in P.

The potential data contributions of mapping sets can be used to compare different mapping sets and—based on that—decide on pruning as discussed in the following section and the Chapters 8 and 9.

6.2. Pruning Subplans

Having the means to assess the utility of mapping sets, we now exploit this information to prune the search space of query planning. First, different general approaches for pruning are compared. Then we present a simple pruning algorithm.

6.2.1. General Pruning Approaches

Corresponding to the open-world semantics and the close-world semantics mentioned in Sec. 2.4, pruning in PDMS comes in two principally different flavors:

Information preserving pruning. In this approach the final query result of the pruned rule-goal tree is equivalent to that of the original tree, i.e., it returns the *same* query answer. This has been implemented by the Piazza project [TIM+03]. They remove branches of the rule-goal tree whose results are known to be contained in other parts of the tree [TH04]. Regard the rule-goal tree depicted in Fig. 6.1. The subgoal $P_2.h$ can be pruned if $P_2.h \subseteq P_2.g$, i.e., the query $P_2.h$ is contained in $P_2.g$, and if there are no joins between $P_2.g$ and $P_1.f$. The subgoals contained in each other must be at the same peer. Tatarinov and Halevy show that this technique may cut down the effort for query reformulation up to an order of magnitude. However, the approach presumes to have knowledge of the global query plan. This contradicts our goal to perform query planning only locally at each peer to preserve their autonomy.

Concessive pruning. In flexible large-scale information sharing users often may be satisfied with a considerable part of all query results. They are neither able to examine every result in detail nor do they want to perform accurate aggregation operations on these results. As systems for information integration scale up, resources may become scarce. Additionally, the data sources are not available permanently. So we believe that query answering in such settings can only be offered as best effort service. The system returns as much benefit as possible for a limited budget of resource consumption, i.e., the query result can be extensionally and intensionally incomplete. Strategies for concessive pruning should minimize this loss of information without exhaustively enumerating the entire search space.

In this thesis we follow the concessive pruning approach since we highly emphasize peer autonomy, Sec. 1.2.

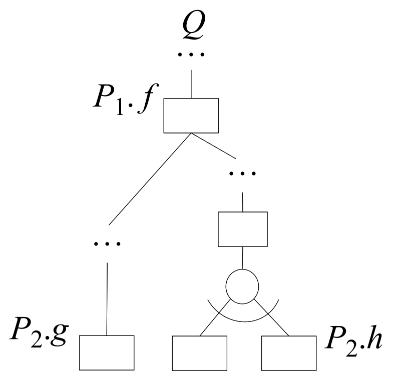


Figure 6.1.: A *global* rule-goal tree for which the Piazza system [TH04] may prune the goal node $P_2.h$ (image from [TH04]).

6.2.2. Threshold-based Pruning

Now we present a first pruning approach that compromises the query result on the one hand, but considerably increases efficiency of query answering on the other hand. More elaborated techniques are presented in Chapters 8 and 9. Here, mapping sets with small expected subresults are pruned if applicable. This is realized in PRUNE depicted in Algorithm 6.2.1. The main idea of PRUNE is to use the potential data contribution of a mapping set to decide about pruning. As mentioned above, the algorithm takes the fully expanded local query plan $\hat{P}(Q)$ of a query Q as an input.

```
Input  : Query $Q$ with selection predicates $Sel(Q)$, fully expanded local query
         plan $\hat{P}(Q)$, pruning threshold statistics $t_{\Delta C}(\mathcal{M})$
Output: Pruned local query plan $P_p(Q)$
1  $P_p \leftarrow \hat{P}$
2  foreach rewriting $Q'_i \in \hat{P}$ do                                   {see Fig. 2.6}
3      foreach conjunct $C_{ij} \in Q'_i$ do
4          foreach mapping set $\mathcal{M}_l \in C_{ij}$ do
5              if $\Delta C(P, R(\mathcal{M}_l)) < t_{\Delta C}(\mathcal{M}_l)$ then
6                  $\mathcal{V} \leftarrow Vars(Sel(Q)) \cup Vars(Sel(\mathcal{M}_l))$      {variables with selections}
7                  if $|C_{ij}| > 1$ or $\forall v \in \mathcal{V} : \exists s \in Q'_i : v \in Vars(s)$ then
8                      $P_p \leftarrow P_p \setminus \langle R(\mathcal{M}_l), Sel(\mathcal{M}_l)\rangle$
9                  end
10             end
11         end
12     end
13 end
14 return $P_p$
```

Algorithm 6.2.1: Threshold-based pruning of a query plan (PRUNE).

Please note that although $\hat{P}$ is fully expanded at each peer, the *global* search space is by far not enumerated completely. Doing this would require to combine all fully expanded local query plans. Despite this, the search space referring to the *whole* PDMS is considerably pruned by excluding a mapping set from further query answering. So by *local* pruning the global search space is *indirectly* reduced. The algorithm PRUNE iterates over each mapping set in $\hat{P}$ (Lines 2 - 4).

An additional input for PRUNE is given by a threshold function $t_{\Delta C}(\mathcal{M})$ whose value depends on the mapping set at hand. This threshold is used to filter out the corresponding mapping set from further query processing if the value for ΔC is smaller than $t_{\Delta C}(\mathcal{M})$ (Line 5). This condition is only necessary but not sufficient. To ultimately be pruned, for a mapping set *one* of the following two conditions (Line 7) must also be fulfilled:

– There must be at least one result left for the parent rewriting subgoal ($|C_{ij}| > 1$)

after pruning the mapping set. Otherwise the join-merge between all the subgoals of a rewriting may collapse and return no result although the other subgoals may yield many data. In Sec. 8.2.1 we present a technique that tries to balance the result sizes of the rewriting subgoals to avoid such cases.

– If the mapping set under consideration is the *last* one below its rewriting subgoal, it may only be pruned if the corresponding subgoal is *not* the only one containing a variable over which a selection predicate is defined in the query. Intuitively, this means the rewriting subgoal does not contribute to the join-merge, but all variables with selection conditions still occur in the resulting tuples of the join-merge. As a result, the query result loses all attributes of that query subgoal. The user must decide whether such highly incomplete result tuples are of any value.

If the pruning conditions are fulfilled, the mapping set $\mathcal{M}_l$ is removed from the plan resulting in an incomplete intermediate plan P_p (Line 8). The last state of P_p is finally returned by PRUNE.

The *runtime* of PRUNE is polynomial in the number of rewritings Q'_i, number of subgoals C_{ij} per rewriting, and the number of mapping sets $\mathcal{M}_l$ per subgoal. As stated above, we assume that the size of $\hat{P}$ is small enough to be handled by a single peer.

Clearly, the completeness of the resulting query plan heavily depends on the choice of the threshold $t_{\Delta C}$:

– If it is too high, almost no mapping sets are pruned. Consequently, the cost of query answering tends to be the same as for the complete query plan $\hat{P}$.

– The other extreme is that by a too small value for $t_{\Delta C}$ almost every mapping set is pruned.

So a suitable choice of the pruning threshold is crucial for this approach. A good value for $t_{\Delta C}(\mathcal{M})$ can be determined starting from the cardinality histogram for a mapping set $\mathcal{M}$. Of course, the amount of data a mapping set returns depends on the selection predicates in the query. However, the *upper bound* for the potential data contribution is the one for a query without selections using a particular mapping. So a start value for the maximum value $\Delta C_{\max}$ can be determined for every mapping set by estimating the mapping set cardinalities for a query including all predicates of the peer schema but *without* any selections. Then, a peer can collect statistics on the values for ΔC during normal operation and possibly adjust $\Delta C_{\max}$. Based on these statistics it can choose $t_{\Delta C}$ by some fraction f lower than the maximal potential data contribution $\Delta C_{\max}$ of the mapping set:

$$t_{\Delta C}(\mathcal{M}) = f \cdot \Delta C_{\max}(\mathcal{M}). \tag{6.5}$$

If a change happens in the PDMS the values for ΔC can change for some mapping sets. Then the new threshold $t_{\Delta C}$ can be immediately set by using the same value for f and a new value for $\Delta C_{\max}$ obtained by a query without selections. Note that such a query also helps all subsequent peers to update their thresholds as long as there are no selections in the peer mappings.

Prune does not employ a cost model or cost related statistics. This constitutes an interesting advantage, because accurately estimating cost in dynamic Pdms is a difficult problem, Sec. 4.6. However, the results under this circumstance cannot be as good as with a cost model of sufficient accuracy. So in this chapter, query processing cost is neither an optimization goal nor a boundary condition. Despite this, we observed considerable cost savings induced by pruning and therefore measure query execution cost in our experiments. Going beyond this fairly simple pruning approach, we employ a cost model for query optimization in in Chapters 8 and 9.

A mapping path is the sequence of peer mappings from a peer receiving a user query and a local data storage at another peer. Clearly, the longer mapping paths are in a Pdms, the higher is the probability for information loss and the higher are the cost for data transport. So threshold-based pruning is more effective the longer the mapping paths in the peer graph are. Observe that the length of mapping paths usually increases with the rank of the Pdms, i.e., the average number of peer mappings at a peer.

6.3. Experimental Evaluation

We are interested in how far the algorithm Prune can increase efficiency of query answering. Of course, we also desire query results with high completeness, since increasing efficiency while simultaneously decreasing the size of the query answer too drastically would not satisfy the user. In our experimental evaluation, we vary some important properties of the Pdms and study the effectiveness of threshold-based pruning:

- the rank of the Pdms,
- overall amount of data and its distribution over the peer graph,
- mixture between mapping paths returning many vs. only a few data.
- Bandwidth of network connections between peers
  (variation: 50 kBit/s, 0.5 MBit/s, 5 MBit/s, 50 MBit/s).

6.3.1. Experimental Setup

All of our experiments base on randomly created Pdms instances $\mathcal{P}$, Tab. 6.1. To vary the above properties, we manually changed the instances without significantly influencing their random character.

	#Peers	Rank	Mappings with projections	Average mapping path length
$\mathcal{P}_1$	10	2.8	63%	2.9
$\mathcal{P}_2$	20	4.2	48%	3.4
$\mathcal{P}_3$	10	3	0%	2.3
$\mathcal{P}_4$	4	2.5	0%	2.5

Table 6.1.: Datasets and their main characteristics.

Observe that $\mathcal{P}_2$ from Table 6.1 is much more complex than $\mathcal{P}_1$, because its number of peers is twice that of $\mathcal{P}_1$, in average it has longer mapping paths, and most importantly it has many more possible mapping paths because of its significantly higher rank.

For each experiment, we perform at least 3 runs with the same configuration on the same computer. We check whether the variation of the results is satisfyingly small and then average the values over these 3 runs. All experiments are conducted on an Apple MacBook Pro with an Intel Core 2 Duo processor running at a speed of 2.3 GHz and with 2 GB of main memory.

6.3.2. Evaluation

Experiment 6.1 (Influence of pruning threshold.) This experiment proves the effectiveness of threshold-based pruning and studies how it depends on the choice of the pruning threshold and the network bandwidth between the mappings. We also gather experience in finding an appropriate pruning threshold, since this would be an important task for the administrator of a peer that uses our algorithm.
Methodology. To find a suitable pruning threshold, we take a close look at the potential data contributions ΔC of mapping sets. We are interested how that value changes for a particular mapping set over the query workload. On this basis, an administrator can choose the fraction f that sets the pruning threshold for filtering out the corresponding mapping set if applicable.

To this end, the pruning threshold is varied and the resulting completeness, cost, and efficiency values are recorded and averaged over all queries. For each mapping, we first choose the pruning threshold such that this mapping is *always* pruned for the query workload at hand. In a second step, the pruning threshold is set to a value that leads to pruning in at least two thirds of the usages of the mapping under consideration during the same workload. If a mapping never contributes any data, its pruning threshold is set to 0. This means that this mapping is always pruned if it is not expected to return any result data. Finally, the pruning thresholds of the mappings are adjusted such that a particular mapping is omitted in at least a third of all cases within the same workload.

Orthogonally, we examine how the efficiency of query answering varies with the network bandwidth between the peers. We expect that threshold-based pruning can increase the efficiency the more, the slower the network transports the data.

We analyze how the completeness depends on the pruning threshold. To this end, we assume that each peer passes the query to its neighbors sequentially, i.e., the query is passed to the next neighbor after receiving the answer from the current neighbor. Then, we investigate each overall query plan and measure time consumed for computation and transport of data.
Discussion. The result of this experiment for PDMS instance $\mathcal{P}_2$ (Fig. 6.2(a)) and a workload of 500 queries is depicted in the diagrams in Fig. 6.2. The effectivity of threshold-based pruning is clearly reflected in Fig. 6.2(b). The more we prune, the higher is the completeness reduction compared to query processing without pruning. However, we shall keep in mind that pruning at least a third of all mappings at each peer still yields a completeness of about 0.72. Even pruning in two thirds of all cases achieves much more

than half of the maximal size of the query answer.

The cost reduction, and more importantly, the efficiency gain of the pruned queries compared to querying the complete answer are displayed over different network bandwidths in Fig. 6.2(c) and Fig. 6.2(d), respectively. Notice that the cost reduction for all pruning degrees and all bandwidth values depicted is at least 68%. By trend, the cost savings are the higher, the slower the network transports the result data between the peers. Interestingly, the efficiency gain drops from quite high values for a bandwidth of 50 kBit/s to about one order of magnitude for 0.5 MBit/s and remains on a level of about 7 for higher bandwidths. This also holds for bandwidths higher than 50 MBit/s. The phenomenon shows that for certain levels of the network bandwidth the computing resources necessary for query planning and to process the intermediate query results dominates the cost for query answering.

We performed similar experiments with the PDMS instance $\mathcal{P}_1$. For $\mathcal{P}_1$, the cost reduction for all pruning strengths was at least 50% and the efficiency gain was only about 50-80% for a bandwidth of 5 or 50 MBit/s. This proves that the approach of threshold-based pruning needs a certain level of size of the PDMS to be truely effective. In our example, it is much more effective for $\mathcal{P}_2$ with 20 peers than for $\mathcal{P}_1$ with 10 peers.

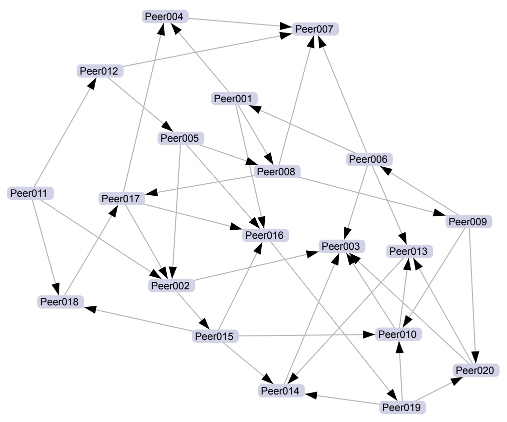
Experiment 6.2 (Impact of distribution of data.) One major characteristic of a PDMS is how the data are distributed over the peers. Every peer can have a similar amount of data or some peers can have more data stored than others, i.e., the overall distribution is skewed. This experiment shows that this distribution has no considerable influence on the effectivity of threshold-based pruning.

Methodology. We vary the skew in the amount of data at the peers according to a Zipfian distribution [Zip49]. The size $s(p)$ of the datasets at the peers are obtained by the equation

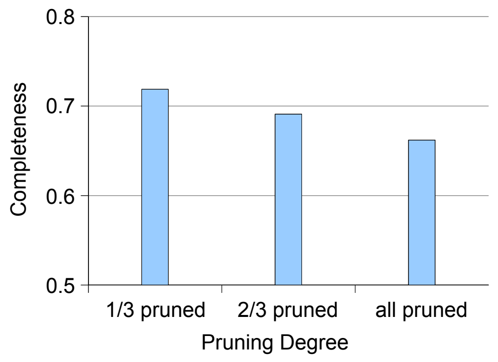
$$s(p) = \frac{c}{p^z}, \tag{6.6}$$

where $p = 1, 2, \ldots$ is the peer number and the parameter z controls the degree of skew. The constant parameter c determines the overall amount of data in the PDMS instance. A value of $z = 0$ means a uniform distribution, hence, each peer has the same amout of data in this case. In constrast, for $z = 1$ and 10 peers the peer with the fewest data holds ten times less data than the peer with the most data. The data distribution in $\mathcal{P}_2$ used in Experiment 6.1 was randomly created and can be approximated with a Zipf parameter between 0 and 1.

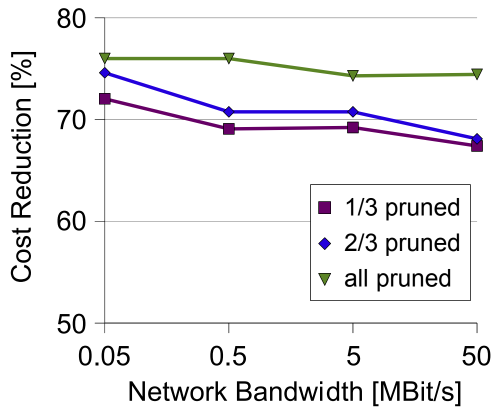
Discussion. As the diagram in Fig. 6.3(b) reveals, the skew of data distribution interestingly has no major influence on the effectivity gain by threshold-based pruning. With exception of very strong skew with $z = 2$ for a bandwidth of 0.05 MBit/s, the skew does not considerably impact the effectivity gain in this experiment with the 10-peer instance $\mathcal{P}_3$. One reason can be that we used a test workload that queried every peer for all of its data and then averaged over this workload. If a peer is queried that is quite "far" from the peers that carry the most data, pruning usually should have a major impact on the efficiency of query answering. However, this is compensated by the queries to the peers that carry many data. There, efficiency gain by pruning is small since the main part of



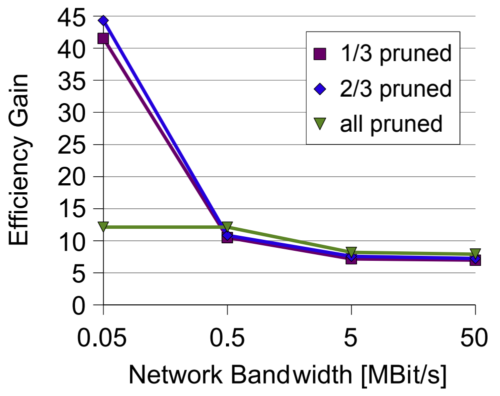
(a) Peer graph $\mathcal{P}_2$.



(b) Loss of completeness by pruning.



(c) Cost reduction by pruning.

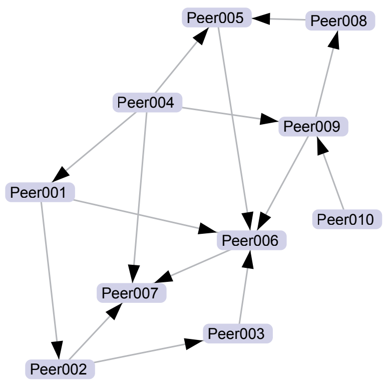


(d) Efficiency gains by pruning.

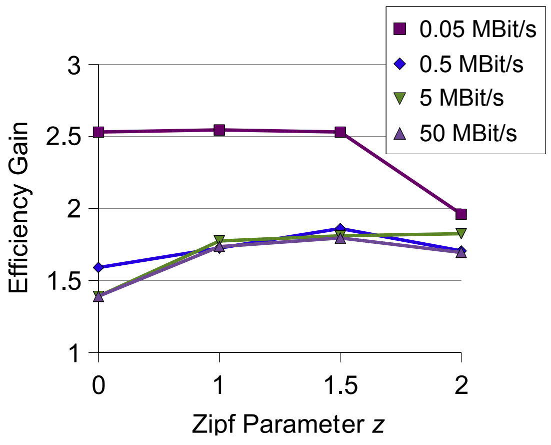
Figure 6.2.: Efficiency increase for different network bandwidths for PDMS $\mathcal{P}_2$.

their query result stems from local data rather than from peers being far apart.

Experiment 6.3 (Efficiency in dependency of the amount of data.) This experiment varies the overall amount of data in the PDMS and examines the impact of this dimension on the efficiency of threshold-based pruning. Recall that the efficiency of query answering has been defined as the ratio between the *completeness* and cost. Pruning decisions in PRUNE are based on the potential completeness difference between alternative mapping paths. The completeness of (intermediate) query answers is *not* expected to change, if the amount of data is uniformly increased over all peers.
Methodology. Increasing the amount of data *without* changing the skew of their distribution over the peers and at the same time retaining pruning decisions should solely affect the cost. We use the simple PDMS instance $\mathcal{P}_4$ and vary the amount of local data at every peer from 1 tuple to 128 tuples. The data sets at the peers are disjoint. Additionally, the world size to calculate the completeness of query answers is adjusted accordingly.



(a) Peer graph $\mathcal{P}_3$.



(b) Efficiency gains by pruning.

Figure 6.3.: Influence of the data distribution on the effectivity of threshold-based pruning for PDMS $\mathcal{P}_3$.

If we additionally choose the pruning threshold such that for the same query the same pruning decisions are made for each PDMS instance, we yield the same completeness results *independently* from the overall amount of data. This approach makes it possible to study the impact of the amount of overall data on the efficiency of query answering.

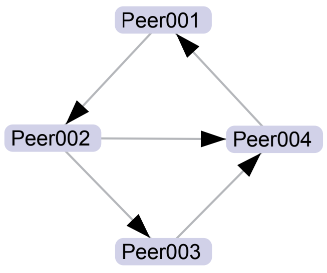
Discussion. The results of this experiment are depicted in Fig. 6.4. Each result is the average of 10 runs of the same configuration. As above, the effectivity of threshold-based pruning is measured by the efficiency ratio between pruned and the same query without pruning.

For the interpretation of this experiment it is important that the PDMS instance $\mathcal{P}_4$ is quite small. For much larger PDMS instances with much more data, the fraction of query processing time spent for data transport between the peers is much higher. Therefore, their behavior is expected to be similar to the result with $\mathcal{P}_4$ for small bandwidth, where the data transportation time also plays a major role.

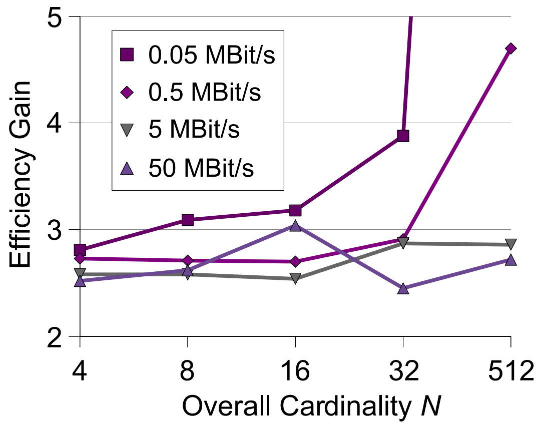
The diagram in Fig. 6.4(b) shows that the threshold-based pruning heavily depends on the bandwidth b between the peers. For a small $b = 0.05$ MBit/s, the efficiency gain clearly increases with the overall amount of data in the system. This effect is weaker for $b = 0.5$ MBit/s. If the bandwidth is even higher, the efficiency gain is nearly constant or varies without any visible trend (for $b = 50$ MBit/s).

6.4. Related Work

First, we discuss work dealing with pruning in query planning in the area of data integration. Then we turn our attention to similar approaches in peer-based systems for data management.



(a) Peer graph.



(b) Efficiency gains by pruning.

Figure 6.4.: Influence of the overall amount of data on the effectivity of threshold-based pruning for PDMS $\mathcal{P}_4$.

6.4.1. Pruning in Data Integration Systems

Nie et al. also use statistics on coverage and overlap to control query planning in a mediator setting [NK04]. As PDMS are a generalization of mediators, our work goes much further in that we examine the behavior of a *network* of mediators. The same holds for the work in [FKL97] and [DH02]. Of course, all algorithms proposed for mediator query planning that build on information locally available can be applied to PDMS as well. However, the focus of our work is how locally controlled query planning influences the global behavior of a PDMS.

Naumann, Leser, and Freytag follow a three-step process to find high-quality query plans in a mediator setting [NLF99]. Their query planning approach uses GLaV-like mappings described in detail in [Les00]. In [NLF99] data sources to be considered in the planning step are first selected based on their information quality scores. Then *all* possible plan are computed. Finally, the plans are ranked according to their information quality characteristics. Our approach is similar to that work, because we also sequentially perform query planning and completeness-based pruning of the fully enumerated search space of a mediator, given by the fully expanded local query plan at a peer.

In [Nau02] Naumann presents a branch & bound algorithm for quality-driven query planning in a mediator. It can find the top N plans without exhaustively enumerating the whole search space of query plans. As stated above, we assume that in a PDMS setting a peer is capable of computing the fully expanded local query plan, because a peer typically has only a few neighbors.

6.4.2. Pruning in PDMS

The mediation between schemas of a Pdms is the main concern of the *Piazza* system [HIMT03, HIST03]. Concessions to the completeness of query results are mentioned, but not discussed further. Instead, Tatarinov and Halevy propose an approach for information preserving pruning during query reformulation [TH04]. It is based on query containment rather than on statistics about neighboring peers. In that sense, the approach in this thesis can be seen as complementary to containment-based pruning and it additionally comes with the important advantage that it can be performed fully locally at a peer. In contrast, checking query containment needs to know some parts of the non-local, overall query plan; peer autonomy is compromised.

In contrast, concessive pruning routes queries in the Pdms based on statistics about neighboring peers and their subsequent neighbors.

The approach of [HLS06] assumes that peers exchange information about updates of their data. This can be used to maintain statistics about neighboring peers. In [PKP04], histograms are used to route queries in a peer-to-peer system with a common peer schema. The work in this thesis is in the spirit of that approach and extends it to deal with the difficulties present in heterogeneous Pdms.

6.5. Summary

Based on the completeness-related metadata presented before, this chapter introduced an approach to prune the local query plan. Pruning is controlled by a threshold for the potential data contribution of a mapping set. Since no cost information is needed, this is a lightweight, yet effective solution, as we show in our experimental evaluation.

First, we showed how to employ our completeness model and statistics to valuate incomplete subplans and to identify mapping sets as candidates for pruning. Then, we discussed two principally different approaches to pruning in Pdms query planning and argued that in large-scale information sharing best effort solutions often suffice, or even are the only way to scale up such infrastructures. The choice of the pruning threshold was identified as a critical decision within this approach. Therefore, we provided a method to find an appropriate threshold based on the completeness statistics and experience from answering previous queries. Finally, our experimental evaluation has shown that Pdms characteristics like the fraction of mappings with information loss due to projections and distribution of the data across the peer graph mainly influence effectivity of our approach.

Pruning mapping paths from query answering impacts our technique for maintaining completeness-related statistics that relies on query feedback. Pruning cuts off query feedback from peers and that can lead to insufficient accuracy of statistics. In the next chapter we develop solutions for this conflict.

7. Maintaining Metadata Statistics

As we argue in Sec. 1.1, flexibility is the major advantage of PDMS. Peers are allowed to enter the system or to go offline at will. Depending on the structure of the PDMS, it can happen that even small changes can have significant impact on the data distribution perceived at certain peers. For instance, peers that constitute a bottleneck-like path in the PDMS topology can cut off access to parts of the system by leaving the network. The changes to the data distribution we face to in PDMS can be much larger than those resulting from usual inserts, updates, and deletes in centralized databases. As a consequence, the statistical approximations of the data distribution at a particular mapping set can become drastically inaccurate, thus leading to wrong query planning decisions. So the challenge arises that PDMS peers must adapt their cardinality and overlap statistics quickly to a possibly completely different situation.

However, query results are required both for detecting the above changes and updating the statistics accordingly. Pruning subplans as introduced in the last chapter naturally cuts off query feedback. Hence, we must compromise between pruning and query feedback.

This chapter first examines the challenges for statistics maintenance that arise from hidden changes, i.e., change events that are located beyond the direct neighborhood of a particular peer. Next, we present a technique to detect hidden changes solely by exploiting query feedback. To this end, we provide algorithms to track the estimation error history at a fine grained level. We also present a solution for the problem of trading off pruning and query feedback, which is based on finding the necessary amount of query feedback to accurately update our statistics. The feasibility of all these techniques is extensively evaluated in experiments conducted using our testbed *Humboldt Peers*.

The techniques and experiments provided in this chapter complement the brief discussion of approaches for histogram maintenance in Sec. 5.1.3. Here we show that using query feedback suffices to accurately and efficiently maintain our statistics while preserving the autonomy of neighboring peers at the same time.

7.1. Implications of PDMS Volatility

As a consequence of their high flexibility, PDMS face a large degree of volatility. This volatility influences the data distribution of a PDMS and has two facets:

Amount of data. The overall amount of data available in the PDMS changes over time due to updates to the data locally stored at the peers and as a result of peers entering or leaving the system. Changes in the amount of data are an important challenge for statistics about result cardinalities. For instance, a peer going offline can be compared with a large instantaneous delete on a significant part of a local database. So the influence of changes in the PDMS structure on histogram accuracy may be much higher than the impact of typical updates in centralized databases. Similarly, in data integration systems a few sources going offline do not influence the amount of data that can be received from the remaining sources. This is completely different in PDMS.

In [HLS06], changes in the PDMS structure are regarded as a special case of updates on local peer data. There, all updates are assumed to be communicated to neighboring peers. In contrast, this work pays special attention on the impact of changes in the structure of the PDMS on the overall amount of available data and conducts experiments covering that issue.

Topology of the peer graph. The topology of a PDMS changes by peers leaving or entering the network or by creation or deletion of peer mappings. This may have a drastic influence on the data distribution throughout the topology and therefore also on the amount of data available for a particular peer. In general, a topology can be highly sensitive to local changes. In an extreme situation, creating a single peer mapping can provide new access to many further peers.

In practice, many distributed systems have bottlenecks in their topology, most prominently the Internet [ASS03]. Another example are mergers of companies. Originally, each company has an own network of interconnected information systems. Usually, only a few mediators connect these separate spheres to quickly establish data exchange immediately after the fusion. Thus, in the resulting distributed information system, the above mediators form bottlenecks.

The graph structure of the PDMS depicted in Fig. 7.1 contains several bottlenecks. There is a number of peer nodes that are so-called articulation nodes. By definition, the graph falls apart into unconnected subgraphs when an articulation node disappears. For instance, $Peer005$ is an articulation node in the peer graph in Fig. 7.1. If $Peer005$ vanishes, $Peer014$ and $Peer018$ in the right part have no longer access to $Peer007\_REFPEER$, $Peer011$, $Peer013\_REFPEER$, $Peer010$, $Peer004$, $Peer009$, and $Peer012$ in the left part of the peer graph.

To understand the importance of particular peers for query answering in PDMS, we have to examine the graph topology in more detail. Recall that a PDMS is a *directed* graph of peer mappings. So with respect to a particular peer where a user query Q_u originates, some other peer node p_a can be an articulation node although its disappearance does not let the PDMS graph fall apart. However, a certain subgraph can be unreachable by user query Q_u after p_a disappeared, because there is no path of directed peer mappings to that subgraph anymore.

Taken together, even small changes in PDMS can significantly affect the data distribution as it is perceived at individual peers. For the cardinality and overlap estimations to still be accurate, it is necessary to quickly adapt to a possibly fundamentally different data distribution.

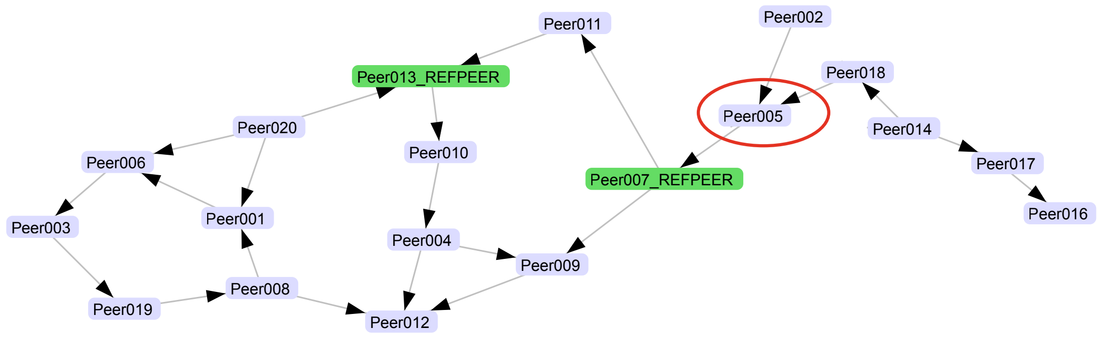


Figure 7.1.: Graph with several articulation nodes generated by *Humboldt Peers*.

7.1.1. Estimation Accuracy and Completeness

To underline the requirement of high adaptivity of the completeness-related statistics, we study the impact of inaccurate cardinality estimations on the size of the query result as well as on the efficiency of query answering in a PDMS.

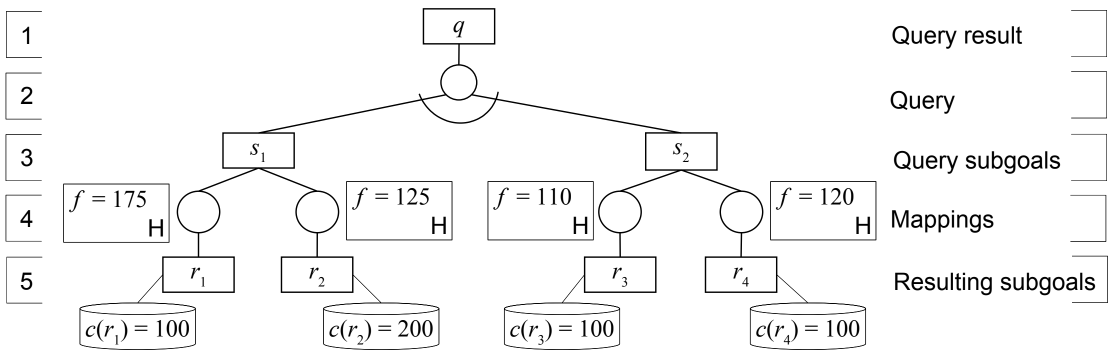


Figure 7.2.: Example of a rule-goal tree with resulting data sets and simple histograms.

Example 15. *Regard the local rule-goal tree in Fig. 7.2. It comprises the two query subgoals s_1 and s_2. Each of the four goal nodes on Level 5 has a local data source, whose cardinality is inserted in the data store symbol. We assigned a symbolic histogram with an estimated frequency f to each of the rule nodes that represent peer mappings in Level 4. We focus on query subgoal s_1 and compare the cardinality estimation with the actual values $c(r_1)$ and $c(r_2)$. The relative estimation error at r_1 is $+75\%$, whereas the cardinality at r_2 is underestimated by -37.5%. The resulting estimated completeness differences ΔC_e for the rule nodes above the r_i are listed in Table 7.1. The peer on Level 3 assumes the world to comprise 1000 tuples in total. Since this is only a normalizing factor it does not matter that the actual world size is $100 + 200 + 100 + 100 = 500$ tuples.*

Assume that the threshold $t_{\Delta C}$ for pruning is 0.025, which is the lowest value that results in pruning any *mapping. Then, the mapping corresponding to r_2 would be pruned,* *whereas all other Level-5 goal nodes would* not *be pruned. So due to the fact the estimation error at the histogram above r_1 is much larger than the inaccuracy of the estimation on r_2, threshold-based pruning makes a wrong decision. The reason is that due to the different estimation errors at r_1 and r_2 the actual cardinality at r_2 is twice as high as the actual cardinality at r_1.*

How such a poor decision impacts the result of a local rule-goal tree depends on the context given by the other rule and goal nodes. The more alternative mappings a query subgoal on Level 3 has, the less the influence of an individual mapping path on the result can be expected to be, if we assume independence with respect to data overlap. Similarly, the more query subgoals exist on Level 3, the less the impact of a mapping on Level 4 is supposed to be. In our example, based on the histograms it is estimated that pruning r_2 would change the completeness of the complete rule-goal tree by 0.022, which amounts to 36%. Actually performing this pruning would lead to an expected completeness of 0.019 for the query result. If the histograms were more accurate, pruning r_1 (because of its actual completeness difference of 0.015) instead of r_2 would have led to a doubled overall completeness of 0.038. Observe that the reduced completeness of the pruned rule-goal tree is combined with a cost saving that is not taken into account in this example. □

Pruned Level-5 goal node	r_1	r_2	r_3	r_4
Estimated completeness difference $\Delta C_e(r_i)$	0.033	0.022	0.027	0.030
Actual completeness difference $\Delta C_a(r_i)$	0.015	0.034	0.025	0.025
Actual coverage $c(r_i)$	100	200	100	100
Estimated completeness C_e *complete* rule-goal tree	0.060			
Actual completeness C_a *complete* rule-goal tree	0.053			
Actual completeness C_a rule-goal tree without r_2	0.019			
Actual completeness C_a rule-goal tree without r_1	0.038			

Table 7.1.: Estimated and actual completeness results for the complete and pruned rule-goal tree in Fig. 7.2. $\Delta C(r_i)$ is the completeness difference between the fully expanded local rule-goal tree T and the the one created from T by pruning the rule node above r_i. Estimated values are based on the simple cardinality histograms in Fig. 7.2.

7.1.2. Hidden Changes

Changes in the Pdms graph can happen *beyond* the direct neighborhood of a particular peer. For instance, if the articulation node $Peer005$ in Fig. 7.1 disappears, $Peer014$ would not notice, because it does not know about the existence of $Peer005$ at all. To describe such situations, we introduce the notion of a hidden change.

Definition 21 (Hidden Change)**.** *Let the* Pdms $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ *be given by a set of peers. Let the direct neighborhood $\mathcal{N}(P)$ of a peer P be the set of peers that occur in the head of any of the outgoing mappings of P (Def. 3). A hidden change with respect* *to P is given by any modification in $\mathcal{P} \setminus \mathcal{N}(P)$ or in the outgoing mappings of the peers in $\mathcal{N}$.*

Observe that for autonomy reasons discussed in Sec. 1.2 we assume that a peer does not know about the outgoing mappings of its direct neighbors. This means that a peer has no information from where its neighboring peers receive their data. Modifications beyond the direct neighborship, above denoted by $\mathcal{P} \setminus \mathcal{N}(P)$, can occur if peers leave or enter the system or mappings become invalid due to schema changes.

The following example serves to illustrate the impact of high volatility in the PDMS on our local statistics by studying a hidden change. Moreover, the example highlights the requirement of high adaptability to the maintenance of the statistics.

Example 16. *Regard the* PDMS *in Fig. 7.3, which is a subset of the* PDMS *in Fig. 1.1, showing an infrastructure for medication logistics during a nationwide disaster. It contains hospitals, medication inventories, relief organizations, a logistics company, a government control center, and others. Let the peers* National medication inventory *and* National pharmacies association *have comparably large amounts and great variety of medications in their stocks (depicted as larger database icons).*

Let peer Regional hospital north *query other peers for medicine. Among others the two following alternative mapping paths can be used to reach the two large medication inventories (highlighted in Fig. 7.3):*

p_1*:* Regional hospital north → Capital hospital → Capital medication inventory → National medication inventory → National pharmacies association

p_2*:* Regional hospital north → Red cross capital headquarters → Government control center → National medication inventory*,* National pharmacies association*.*

The path p_1 is better than p_2, i.e., it usually returns more data, because it contains mappings between similar organizations, namely hospitals and medication inventories. This naturally results in a comparably low loss of information on the complete mapping path. In contrast, we assume that the mapping path p_2 loses a considerable part of the data that can be found at the large data sets at the end of both paths due to projections is the mapping from Regional hospital north *to* Red cross capital headquarters*.*

Now we assume that the peer Capital medication inventory *goes offline. The situation existing immediately after this event is highlighted in Fig. 7.3 by the dashed peer mappings. These mappings do not exist any more after the change. The disappearance of the peer* Capital medication inventory *radically changes the size of results accessible via p_1 respectively p_2. The two large data sets at the* National medication inventory *and* National pharmacies association *are not reachable by (the remaining rest of) p_1. In contrast, the path p_2 remains unaffected. Taken together, now p_2 promises much more data than p_1. Observe, that the peer initiating the queries, namely* Regional hospital north*, is not informed about the change "behind" the peers* Capital hospital *and* Government control center*. Rather, it can detect this massive change only by exploiting query feedback. Once it has detected the change, the peer* Regional hospital north *must quickly adapt its statistics to this new situation.* □

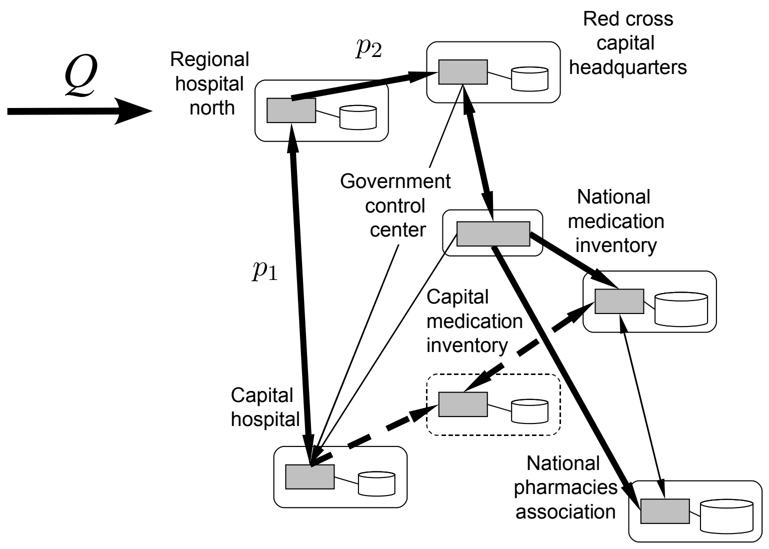

Q
Regional hospital north
p_2
Red cross capital headquarters
Government control center
National medication inventory
p_1
Capital medication inventory
Capital hospital
National pharmacies association

Figure 7.3.: PDMS topology immediately after the peer Capital medication inventory went offline (extract).

7.2. Capturing Hidden Changes

One of our main assumptions is that in practice the peers are very interested in high autonomy. This means that the peers offer only a query service. We assume, that the peers are not willing to contribute actively to the maintenance of other peer's statistics. Consequently, hidden changes must be detected solely from query feedback. As discussed in Sec. 5.1.3, the horizon of query feedback comprises all data that potentially will be returned for a query. This is why it is in principle possible to detect hidden changes from query feedback.

Our main idea in this section is to track the estimation error to find out about hidden changes. To this end, we observe the estimation accuracy at a fine grained level. If a hidden change is detected, the peers must decide how to quickly adapt their statistics to this event. We present several approaches for this task.

7.2.1. Tracking the Estimation Error

The estimation accuracy of a histogram can be measured by the relative estimation error, which is defined as follows:

Definition 22 (Relative Estimation Error)**.** *Let Q be a query and $R_a(Q)$ the actual result. With the estimated result $R_e(Q)$ the relative estimation error amounts to*

$$e(Q) = \frac{||R_a(Q)| - |R_e(Q)||}{|R_a(Q)|}.$$

The STHoles histogram approach assumes the tuple density within a particular bucket to be uniformly distributed. In reality, however, the tuple distribution within a bucket is skewed. Consequently, the estimation error can also differ over the volume of a bucket. Hence, the error is volume-dependent. So the estimation accuracy is the higher the smaller the volume under consideration.

Therefore, we propose to observe the history of the estimation error at the level of histogram buckets.

Definition 23 (Estimation Error History)**.** *Let $\mathcal{W} = \langle Q_1, Q_2, \ldots, Q_n \rangle$ be a sequence of queries, i.e., a query workload. With $time(Q_i)$ denoting the point in time the query was processed, it holds that $time(Q_i) < time(Q_{i+1})$. Let $E_i = \langle e_i, B_i, Q_i, R(Q_i) \rangle$ be an individual entry of the estimation error history of a particular bucket B with the estimation error e_i at $time(Q_i)$, the possibly modified bucket B_i (see below), as well as the result $R(Q_i)$. Then the estimation error history of bucket B for $\mathcal{W}$ is given by the sequence $\mathcal{H}(B) = \langle E_1, E_2, \ldots, E_n \rangle$.*

Note that new entries are added at the *end* of the estimation error history. So the estimation error history reflects the order the respective queries were used for adapting the corresponding histogram. In the following, we describe the maintenance of estimation error histories in case their corresponding bucket is modified during histogram maintenance.

Adapting Error Histories to New Holes. Our goal is to compare estimation errors over time to detect changes in the distribution of data accessible over a certain mapping. Since the estimation error is volume-dependent, we necessarily have to compare similar volumes for this task. Recall that drilling a new hole into a bucket changes its volume [BCG01]. So, if a new hole is to be drilled into a bucket B at a certain time, it also has to be drilled into *every* earlier entry of B's estimation error history. Every entry of the error history then has to be recalculated with the new bucket volume created by drilling the hole. These steps are performed by Algorithm 7.2.1.

When inserting the new hole into each bucket of the estimation error history we have to take into account their respective states. Buckets of history entries at earlier points in time than that of the latest entry (denoted by E_c) can contain children that are not present in E_c. Such children could have been merged with their parent in the meantime. Recall that in the STHoles approach, buckets may not overlap. So before actually drilling a new hole, the candidate bucket has to be shrunken to exclude overlap with any other children of the parent bucket [BCG01] (see line 3 in Algorithm 7.2.1). So in fact, the same technique as for the new hole in the latest bucket has to be applied to drill that (candidate) hole into all the buckets of earlier members of the error history. After doing so, the estimation error of the corresponding entry can be recalculated.

Merging Error Histories. If two buckets are merged, their error histories have to be merged as well. Next, we propose Algorithm 7.2.2 to compute the combined estimation error histories of two buckets B_a and B_b being merged into a resulting bucket B_r. Intuitively, we iterate in parallel over the error histories and look for entries corresponding

```
Input   : Estimation error history $\mathcal{H}(B)$, new candidate hole $B_c$
Output: Modified estimation error history $\mathcal{H}'(B)$
1 foreach $E \in \mathcal{H}(B)$ do
2     Shrink $B_c$ such that it does not overlap with any child of $E.B$ [BCG01]
3     Drill $B_c$ into $E.B$ [BCG01]
4     Recalculate $E.e$ using the adapted volume of $E.B$, $E.Q$, and $R(Q)$
5 end
6 return $\mathcal{H}'(B)$
```

Algorithm 7.2.1: Adapting an estimation error history to a new hole.

to the same query that are to be merged (lines 22 - 26). If an entry has no suitable partner, it must be checked whether its bucket covers the bucket B_r to a certain extent (lines 13 and 18). If this is the case, the entry can represent the whole volume of B_r while keeping the error of this approximation low (lines 14 and 19).

If the two entries under consideration refer to the same query Q, the weighted mean $\bar{e}$ of the estimation errors is calculated (line 24). The errors coming from the two histories are weighted with respect to the fraction of their buckets overlapping with Q.

7.2.2. Detecting Changes

To check whether a possibly hidden change has happened in the data accessible over a certain mapping path, we examine the estimation error history of each bucket in the corresponding histogram. To detect a change event, it suffices that any of these buckets reports a sudden variation in the data distribution of its query feedback.

In experiments we observed that even within an estimation error history of a single bucket, estimation errors can differ significantly although the underlying data distribution did *not* change. As discussed in the previous section, the estimation error is volume-dependent due to skew in the data distribution.

Therefore, our method to detect changes from query feedback presented in Algorithm 7.2.3 only compares entries of an estimation error history if they cover a similar sub-volume of a histogram bucket (lines 2 and 3). For instance, in Fig. 7.4 on Page 114 the queries $E_1.Q \cap B$ and $E_2.Q \cap B$ have a large overlap with respect to the sum of their volumes (line 2 in Algorithm 7.2.3). So we can assume that their estimation errors are very similar. In contrast, the overlap volume between $E_3.Q \cap B$ and $E_1.Q \cap B$ respectively $E_2.Q \cap B$ is much smaller. Therefore, their estimation errors probably differ more due to skew in the data distribution in bucket B. Note that calculating the volume v according to [BCG01] excludes children of a bucket, for instance the child B_c of bucket B in Fig. 7.4 on on Page 114.

If no change happens in the data accessible over a peer mapping, the estimation errors of two entries covering overlapping volume are similar, i.e., the difference between their errors is below a certain threshold that relates to the mean of the two errors (line 3). Observe that the estimation errors of a pair of history entries in Algorithm 7.2.3 do

```
Input  : Estimation error histories $\mathcal{H}(B_a)$ and $\mathcal{H}(B_b)$, resulting bucket $B_r$
Output: Estimation error history $\mathcal{H}(B_r)$ of $B_r$
$i \leftarrow 1$, $j \leftarrow 1$
while $i \leq |\mathcal{H}(B_a)|$ or $j \leq |\mathcal{H}(B_b)|$ do
    $E_a \leftarrow \{\}$, $E_b \leftarrow \{\}$                              $\{E_x$: current entry of $\mathcal{H}(B_x)\}$
    if $i \leq |\mathcal{H}(B_a)|$ then                                 $\{$No correspond. entry in $\mathcal{H}(B_a)$?$\}$
        $E_a \leftarrow \mathcal{H}_i(B_a)$
        $i \leftarrow i + 1$
    end
    if $j \leq |\mathcal{H}(B_b)|$ then                                 $\{$No correspond. entry in $\mathcal{H}(B_b)$?$\}$
        $E_b \leftarrow \mathcal{H}_j(B_b)$
        $j \leftarrow j + 1$
    end
    if $time(E_a.Q) < time(E_b.Q)$ then                      $\{E_a$ has no corresponding entry$\}$
        if $v(E_a.B) > \frac{1}{2}v(B_r)$ then                           $\{v(E_x.B)$: volume of $E_x.B\}$
            $\mathcal{H}(B_r).add(\langle E_a.e, B_r, E_a.Q, E_a.R(Q)\rangle)$          $\{$add $E_a$ to result$\}$
        end
    end
    if $time(E_a.Q) > time(E_b.Q)$ then                      $\{E_b$ has no corresponding entry$\}$
        if $v(E_b.B) > \frac{1}{2}v(B_r)$ then
            $\mathcal{H}(B_r).add(\langle E_b.e, B_r, E_b.Q, E_b.R(Q)\rangle)$
        end
    end
    if $time(E_a.Q) = time(E_b.Q)$ then
        $Q \leftarrow E_a.Q$                                                     $\{$here: $E_a.Q = E_b.Q\}$
        $\bar{e} \leftarrow \sum_{x=a,b} \frac{v(E_x.B_i \cap Q)}{\sum_{x=a,b} v(E_x.B \cap Q))} E_x.e$               $\{\bar{e}$: averaged estim. error$\}$
        $\mathcal{H}(B_r).add(\langle \bar{e}, B_r, Q, E_a.R(Q)\rangle)$                 $\{$here: $E_a.R(Q) = E_b.R(Q)\}$
    end
end
return $\mathcal{H}(B_r)$
```

Algorithm 7.2.2: Merging estimation error histories.

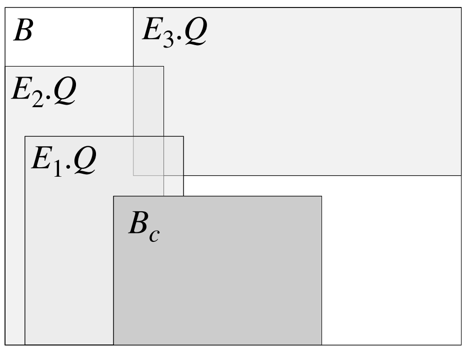


Figure 7.4.: Bucket with entries from its estimation error history.

not only differ because they do not exactly cover the same sub-volume of a bucket. Rather, the above approach to merge estimation error histories (Algorithm 7.2.3) is an approximation.

An extension of Algorithm 7.2.3 is to give to more belief to pairs of history entries the higher their overlap is. This could be implemented by varying the threshold t_e dependent on the size of the overlap $v(E_i.Q \cap E_j.Q \cap B)$ between the entries and the bucket.

```
Input  : Histogram $H$ with estimation error histories $\mathcal{H}(B_i)$, threshold for
         volume overlap $t_v$, threshold for error difference $t_e$
Output: *true*, if a change has been detected, *false* otherwise
1 foreach $B \in H$ do                                            {$v$ : volume}
2     $\mathcal{O} \leftarrow \{(E_i, E_j) \in \mathcal{H}(B) | v(E_i.Q \cap E_j.Q \cap B) > t_v \cdot (v(E_i.Q \cap B) + v(E_j.Q \cap B))\}$
3     if $\exists (E_i, E_j) \in \mathcal{O} : |E_i.e - E_j.e| > t_e \cdot \frac{1}{2}(E_i.e + E_j.e)$ then
4         return *true*
5     end
6 end
7 return *false*                                    {no change has been found}
```

Algorithm 7.2.3: Checking for a change in a histogram.

7.2.3. Controlling Histogram Adaption

Our ultimate goal in using statistics is to achieve a high cardinality estimation accuracy, i.e., a low estimation error. Once a peer has detected a change in the distribution of the data accessible over a certain mapping, it can use current query feedback as well as the corresponding estimation error history to find out

– how large the estimation error is and

– where large errors are located in the data space.

Based on the heuristic that the possibly hidden change is the bigger the higher the estimation error is, the peer can conclude from the estimation error size to the size of the change. Moreover, the peer can gain information which areas of the data space are concerned by considerable change. Before discussing actions the peer can take to adapt to the change, we continue our example from the disaster data management domain (Example 16).

Example 17. *Regard Fig. 7.3 again. Since the peer* Capital medication inventory *has just left the* PDMS*, the peer* Regional hospital north *observes a high estimation error for the mapping path p_1. This is because p_1 no longer returns the large amount of data accessible at* National medication inventory. *Obviously, the current statistics about p_1 at the peer* Regional hospital north *have to be adapted to a hidden change in this mapping path.*

To avoid poor query planning decisions based on wrong cardinality estimates, a peer needs to *quickly* adapt the corresponding histogram to changes responsible for these bad estimates. To this end, a peer can take the following steps:

– *Adapting query planning.* In a situation with high estimation errors, a histogram needs as much query feedback as possible. Hence, the peer should not prune the corresponding mappings during a certain number pf queries, even if a pruning criterion is fulfilled. In budget-driven query planning as presented in Chapter 8, the peer can give more budget to mappings showing a high estimation error. In Sec. 7.3 below, we elaborate on this trade-off.

– *Additional sampling.* A peer can exploit information on where in the data space the estimation error indicates a PDMS change. Either it can extend the selection predicates of future queries to cover these regions. Or it can pose additional sampling queries to gather query feedback for these critical sub-volumes of the data space. However, this would come with all the drawbacks of sampling mentioned in Sec. 5.1.3.

  As a result, the peer obtains more query feedback than it would have received if it only had processed the queries posed from other peers. Of course, before passing the result of extended queries back to where the original query came from, the peer must compute the correct answer by applying the selection predicates of the original query. Note that this technique is an *additional* sampling. In contrast to completely independent sampling, it primarily *extends* queries induced by the user and is thus more efficient than only issuing additional queries possibly overlapping with queries processed anyway.

7.2.4. Experimental Setup

This section presents experimental evaluation of the main aspects our solution based on random synthetic data and various workloads. We show that STHoles histograms can detect changes in the PDMS to a certain extent by only exploiting query feedback. The histograms prove to rapidly adapt to massive changes in the data distribution of a PDMS caused by these changes. In particular, we examine whether estimation accuracy is sufficient to effectively and efficiently route queries in the PDMS based on the threshold-based pruning approach discussed in Chapter 6. Moreover, we are interested in how the pruning of mappings influences the accuracy and adaptability of their histograms; i.e., we investigate whether our techniques ensure enough query feedback despite repeated pruning of mappings. The above changes are due to peers entering or leaving the system during an experiment.

Data Sets As described in Sec. 3.3, a PDMS instance is created in *Humboldt Peers* by varying a reference schema and a corresponding instance and then assigning the resulting extensions to the peers. The extension of the reference schema is created at random conforming to a Zipfian data distribution [AC99]. This distribution reflects data skew due to functional dependencies between different attributes of a relation. Such skewed data serve to examine the differences between cardinality estimation results based on multi-dimensional histograms and the assumption of uniformly distributed and independent attribute values.

As described in [AC99], we first generate N distinct random data values for each attribute dimension of a relation. These are combined into N distinct tuples. Next, a frequency value out of a Zipfian distribution is assigned to each of the N tuples. Each of these tuples corresponds to a cell in the joint frequency distribution matrix [PI97]. The frequencies are obtained by the equation

$$f(r) = c \cdot r^{-z} \text{ with } r = 1, 2, \ldots, N \tag{7.1}$$

where the parameter z controls the skew of the data distribution. A uniform distribution has $z = 0$, whereas $z = 1$, $z = 2$, and $z = 3$ can be seen as medium, high and very high skew respectively. The constant c is used to normalize the frequencies to create a certain number of overall tuples. In our experiments, the size of the reference extension was 1,000 tuples for PDMS $\mathcal{P}_3$ and 10,000 for $\mathcal{P}_5$. We experimented with $N = 100$ distinct data values per attribute dimension. Both for an initial calibration of the histograms and for user queries we employ randomly generated query workloads. The queries are posed to randomly selected peers in the PDMS. An additional instance used in the experiments in this section is described with some important characteristics in Table 7.2. Its topology is depicted in Fig. 7.5.

	#Peers	Rank	Mappings with projections	Average mapping path length
$\mathcal{P}_5$	5	2	0%	3

Table 7.2.: Additional PDMS instance and its main characteristics (continued from Table 6.1 from Page 98).

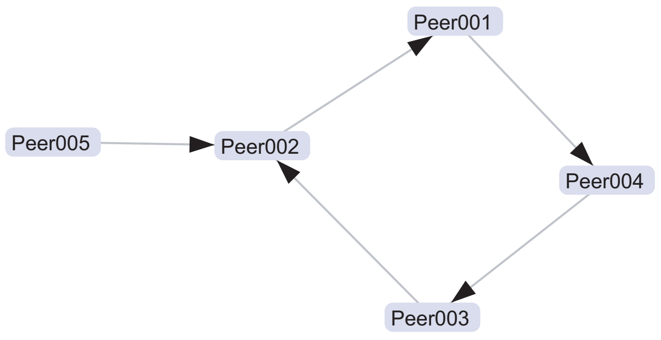


Figure 7.5.: Structure of the PDMS $\mathcal{P}_5$.

Measuring histogram adaptability. To measure adaptability of the histograms to the dynamic behavior of our system, we observe the query-dependent relative estimation error $e(Q)$. What can be regarded as a good cardinality estimation depends on our pruning strategy. To decide about pruning peer mappings, we compare their potential data contribution as discussed in Sec. 6.1. In our experiments, we found that this data contribution differs by several orders of magnitude between alternative peer mappings. This means that a relative cardinality estimation error below an order of magnitude, i.e., below 1,000%, is already sufficient for distinguishing the size of data contributions of alternative peer mappings.

Storybook of Experiments. To examine the influence of massive changes in the PDMS data distribution, we follow a storybook describing the steps to be performed for a particular data set:

1. Vary the schema and the extension of the given reference data set and assign the resulting heterogeneous schemas and data sets to the individual PDMS peers.
2. Initialize an STHoles histogram for each peer and local mapping in the PDMS.
3. *Train* histograms by a random *training workload* to the peers in the PDMS.
4. Issue a first part of a *testing workload* and measure histogram accuracy.
5. *Change the data distribution*, for instance by turning a certain fraction of peers offline or by adding new peers.
6. Issue a second part of a user query workload and *observe adaption* of histograms to the new PDMS configuration.

Note that in Step 3 each query contributes to refine all histograms along the mapping paths used for its reformulation. We characterize the size of changes of the PDMS by counting all non-NULL attribute values at the peers that are removed or added to the system in relation to the overall number of attribute values. For instance, a change size of −50% means that a data set half of the size of *all* data existing in the PDMS before become unavailable.

7.2.5. Evaluation

In the first set of experiments presented in this section we seek to evaluate the ability of our self-tuning histograms to quickly adapt in a highly dynamic setting such as a PDMS.

Adaptability in dynamic PDMS Adapting to large changes in the system is a major requirement to ST-histograms in the context of PDMS. This first set of experiments deals with detecting changes and the estimation accuracy after changes.

Experiment 7.1 (Detecting changes). The challenge in detecting hidden changes in the PDMS from the point of view of a particular peer is to distinguish corresponding changes in the query feedback from usual variation of the estimation error. Additionally, we are interested to measure whether changes that happen in a certain distance (in terms of peer mappings) from a histogram can be detected.

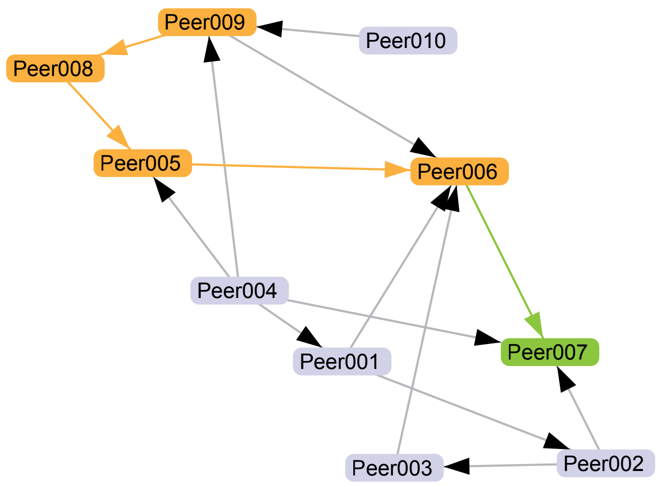
Methodology. After a training phase, a change happens in a PDMS. This change is given by a peer leaving the system. The size of the change is measured by the amount of data this peer stores. To discover a hidden change, each peer records the estimation error for the cardinality histograms assigned to its peer mappings. To assess the influence of the distance to the change, we compare the estimation error graphs for the same query workload along a mapping path. Observe that in practice a peer has to detect changes without being able to compare with a workload without changes.

To better distinguish between normal variation and a potential change, outliers of the estimation error are eliminated as follows: As we discuss in conjunction with the next experiment, large estimation errors are usually due to a high skew of the actual data distribution within an individual histogram bucket. Since we use the average relative estimation error to detect changes in the PDMS, we omit all estimation errors higher than 500%. Additionally, all measurements that are more than a factor of 5 above the average of the last 5 measurements and exceed a threshold of 200% are also eliminated from the statistics.

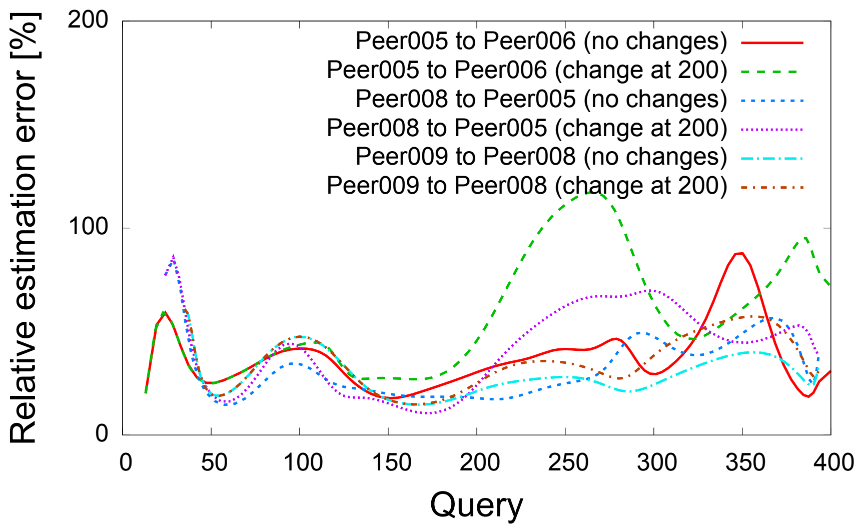
Discussion. The measurements for PDMS $\mathcal{P}_3$ are displayed in Fig. 7.6 on Page 119. The diagram in Fig. 7.6(b) shows the estimation errors along the mapping path highlighted in Fig. 7.6(a) both for the experiment with a change at query 200 and the experiment without any change in the peer graph.

$Peer005$ can clearly notice the change in the data accessible over the mapping from $Peer005$ to $Peer006$ by a large increase of the averaged estimation error after query 200. As can be seen in the experiment without changes at query 350, for the same mapping a peak with similar level occurs due to highly selective queries. So such sudden increases of the estimation error can lead to false positives in change detection. We believe that this effect can be reduced by a more sensitive method for outlier elimination based on the discussion of the reasons for large estimation errors in the following experiment.

If the distance to the change is increased by one peer mapping by considering the estimation error for the mapping from $Peer008$ to $Peer005$, one can still see considerable



(a) Initial peer graph. A change happens at query 200: *Peer*007 with about 10% of all data goes offline.



(b) Estimation error (average of last 5 measurements and smoothed with with Bezier splines) along the mapping path highlighted in the peer graph above.

Figure 7.6.: Relative estimation error in dependency of the distance from the change for PDMS $\mathcal{P}_3$.

increase, which lasts for a comparable long period. The length of such an increase in terms of the number of peers is also a means to distinguish changes in the PDMS from oscillation due to outliers. However, compared to the mapping discussed before, this peak is more difficult to separate from normal variation.

When considering the next peer mapping in the path highlighted in Fig. 7.6(a), it can be seen that the response to the change at query 200 cannot be distinguished from the local maxima during the training phase.

To learn about the influence of the size of the change, we increased the amount of data stored at *Peer*007. With a doubled amount of data, the change is visible with similar strength at all three mappings examined in detail. In an experiment with *Peer*007 having four times more data than the other peers, the change in the relative cardinality estimation error is very strong in the mapping from *Peer*008 to *Peer*005. In the mapping with the biggest distance out of the three considered, the change is also clearly reflected in the error graph. Interestingly, in the mapping that lies nearest to the *Peer*007, the increase is too small to detect the change. A closer examination revealed that this mapping yields a comparably high estimation error of 166% right before the change. After the change, is has no cardinality greater than 0 for a period of about 30 user queries. Then, the estimation error is only moderate.

Taken together, we are able to detect a hidden change in the data distribution of the neighbor of a neighboring peer in this example. If the change is of sufficient size, it can even be detected at peers with a higher distance from the change's location.

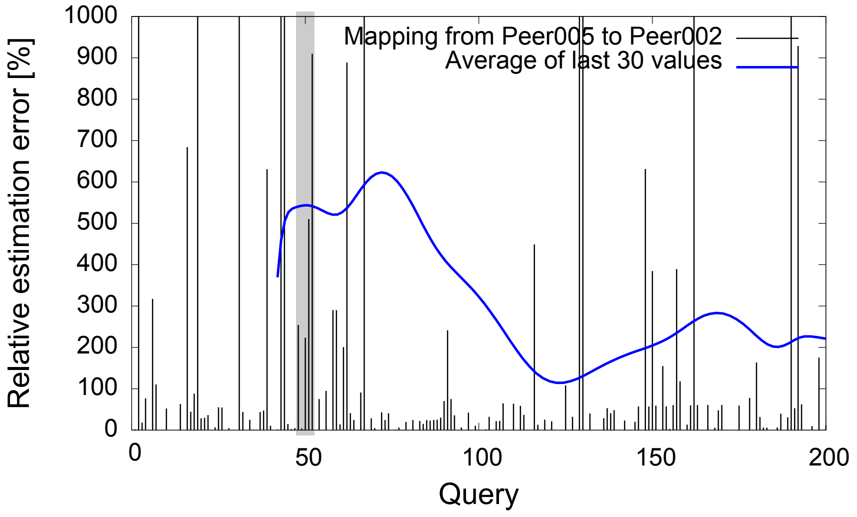
Experiment 7.2 (Accuracy after changes). This experiment serves to study the adaption of STHoles histograms to large changes in the PDMS without taking any further actions: in this experiment we perform no pruning.

Methodology. We changed the size of the PDMS $\mathcal{P}_5$ by disabling the peers *Peer*001, *Peer*004, and *Peer*003 such that they no longer accept any queries (Fig. 7.5). In this example, this means that about 55% of all data values go offline. Observe that this is a considerably large change in the overall data distribution. The histograms are tuned at every query to achieve the maximum adaptability.

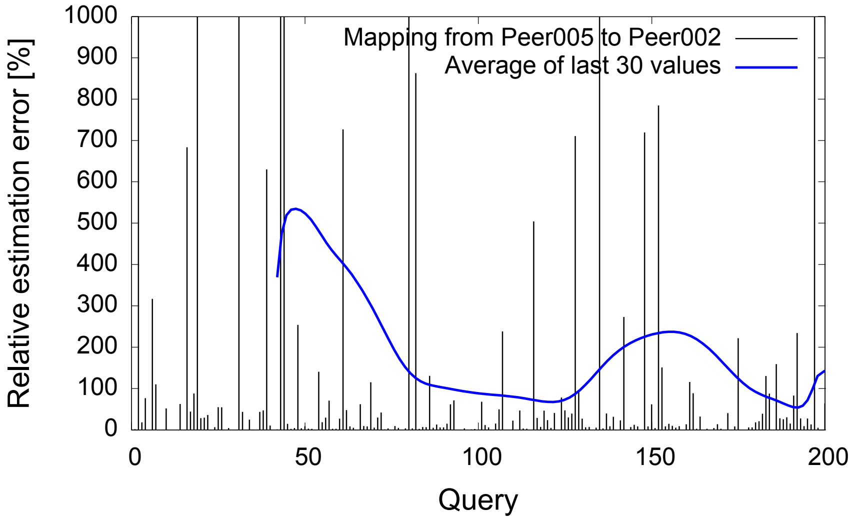
Discussion. The massive change in the PDMS is reflected by the estimation error in the diagram in Fig. 7.7(a). Compared to the same experiment without any changes depicted in Fig. 7.7(b), the estimation error increases significantly after the change at user query 50. This is clearly reflected in the very different graphs of the average estimation error in Fig. 7.7. The upper diagram shows that the estimation error rapidly decreases to a level of less than 100%. Observe that no elimination of outliers is performed in this experiment.

Large estimation errors. Now we examine the reason for the very high estimation errors of more than an order of magnitude occuring in Fig. 7.7. For instance, the error at user query 129 overestimates the actual size of query result drastically. The estimation is 71 tuples, whereas the actual cardinality is only 5. This is a consequence of our highly skewed data distribution of this example. There is a very large area around these 5 tuples that does not contain any other tuple. Due to our quite coarse decomposition of the data space by only 10 buckets in this experiment, this user query selects a fraction of several comparably large buckets, that averages the tuple density within its area. The data distribution in these buckets is also skewed. In this case, the query only covers a small part of the tuples located in these buckets. If the selection area is large but the actual number of tuples in this selected area of the bucket is very low or the skew of the data distribution in the buckets is large, the estimation error will become very large.

However, note that including this mapping for answering user query 129 does no harm, because of the *over*estimation the actual effort for transporting the result is much less



(a) Hidden change: *Peer*001, *Peer*004, *Peer*003 go offline at query 50. The change is marked by a grey bar in the background.



(b) Without any changes.

Figure 7.7.: Relative estimation error (measurements and averaged) for PDMS $\mathcal{P}_5$.

than expected. In general, it depends on the reason that a certain mapping path actually returns much less data than estimated. Only if data is lost along a mapping path the cost for query processing can be too high for a comparably small result. In such cases, a cost-based query optimization such as described in Chapter 8 achieves better efficiency results than a simple threshold-based pruning (Chapter 6).

*Under*estimating the cardinality of a mapping path induces that the mapping is possibly pruned or ranked lower, which means that execution time is saved. However, if an underestimated mapping path is used, query execution cost will be higher than expected.

The extremely large estimation error at query 67 in Fig. 7.7(a) has a different reason. Directly before the change of the PDMS instance at query 50 the ST-Holes histogram has a small bucket comprising 1040 tuples. This is a considerable fraction of all tuples (6550) that are available over the mapping path under consideration. All tuples in that bucket become unavailable after the change. Then, query 67 covers this bucket that still assumes that these tuples are there. In other words, the histogram overestimates the actual data distribution due to the *change* and the fact that a large amount of data is located within a small area of the data space. Using the query feedback of query 67, the wrong frequency of the above bucket is set to 0, which drastically improves the estimations over the corresponding area in the data space in following queries. At this point, our improvement from Sec. 5.1.2 to use the actual frequency to create new holes or adapt the frequency of existing holes comes into play.

To summarize, this experiment shows the ability of the STHoles histograms to quickly adapt to massive changes in PDMS if we have enough query feedback to tune the histograms. The next experiments discuss situations where the amount of query feedback is reduced by pruning mappings from query answering.

Taken together, the main observation in these experiments is that ST-histograms prove to be robust against changes in the data distribution of the PDMS. With some tolerable exceptions, the estimation error obviously remains on the level of about 100 %, which is absolutely acceptable for our purposes.

7.3. Trading off Pruning and Query Feedback

In contrast to centralized DBMS, in the much more dynamic environment of a PDMS changes

– are much more frequent than in centralized DBMS,

– can concern a comparably high fraction of the overall data, and

– can dramatically modify the data distribution as perceived from the point where queries are initiated.

Therefore, in PDMS, self-tuning histograms continuously need query feedback to be updated. This is especially important in situations, where *large* changes in the PDMS have been detected as described in the previous section.

However, the need for query feedback conflicts with strategies of *pruning* the search space in PDMS query planning as discussed in the Chapters 6, 8, and 9 of this thesis or in the literature, e.g., [HLS06, PKP04, RNHS06]. When a certain mapping is pruned, i.e., not considered for query reformulation, the corresponding histogram receives no query result to for histogram adaption. Moreover, if this mapping is pruned *repeatedly* since a small result size is expected, changes in the PDMS structure happening “behind” that mapping are never reflected in the histogram. This is especially problematic, if the change would *increase* the size of data accessible by that mapping path and thus cause that the mapping no longer has to be pruned. Such a situation is illustrated in the following example.

Example 18. *We extend the Examples 16 and 17 (from pages 109 and 115, respectively). Recall the* PDMS *change depicted in Fig. 7.3 on Page 115. The change cuts off the mapping path p_1 from a large data store at* National medication inventory*. This is a hidden change with respect to the peer* Regional hospital north *on which a query workload is posed to. We assume that peer* Regional hospital north *has detected this change and has adapted its histogram for p_1.*

In the following, this might cause repeated pruning of p_1, because the alternative path p_2 promises more data. Then, suppose Capital medication inventory *switches back online and the original mapping path p_1 to the large data set at peer* National medication inventory *is completely available again. However, due to the peer-to-peer paradigm* Regional hospital north *is not informed about that event. Hence, it continues to prune p_1, because the corresponding histogram remains unchanged. This is because the hidden change is masked by repeated pruning. Of course, this change can not be detected since no query feedback is available to the histogram.*

However, in this new situation, it would be better for the peer Regional hospital north *to switch query reformulation from p_2 back to the path p_1, for instance because p_1 better preserves the data on their way from* National medication inventory *to* Regional hospital north*.* □

A solution to this problem must somehow compromise between both objectives. In the following, we present an approach that addresses the two subproblems of this conflict:

– Avoid pruning after a peer has just detected a change in the data accessible by a peer mapping and in the following needs to adapt the corresponding histogram as already mentioned in Sec. 7.2.3.

– Avoid repeated pruning to check for changes in the data “behind” a peer mapping.

The main idea of our solution is to sacrifice *some* efficiency gains of pruning to obtain sufficient query feedback. This means that, in some situations, we explicitly do not prune although pruning a mapping path is supposed to increase efficiency of query answering.

Observe that the better we can exploit query feedback, the less we have to compromise pruning. At this point, our efforts to optimally make use of query feedback by tracking the estimation error on the finest granularity pay off again. In particular, this helps detect PDMS changes from query feedback with only a small amount of query feedback.

7.3.1. Guaranteeing Sufficient Query Feedback

After a peer has detected a change, it needs further query feedback to adapt the corresponding histogram. The histogram enters an *adaption phase* lasting until its estimation error in average decreases below a threshold. During this phase pruning must be discontinued with the sole purpose of obtaining query feedback.

To control the trade-off between pruning and query feedback, we introduce the following parameter.

Definition 24. *(Minimum Non-Pruning Frequency) Let $\mathcal{M}$ be a set of outgoing mappings at a peer P being covered by a histogram $H_{\mathcal{M}}$. The minimum non-pruning frequency f of the histogram $H_{\mathcal{M}}$ guarantees that at least after a sequence of $f-1$ prunings of the mappings in $H_{\mathcal{M}}$ they are used for creation of the local query plan.*

A peer considers each mapping's minimum non-pruning frequency in its pruning decisions. In this way, we ensure that at least each f-th query actually returns query feedback for the corresponding mapping. With the minimum non-pruning frequency we have an independent mechanism to control the compromise between pruning and query feedback for each histogram. By decreasing f for a certain histogram, we can enforce that the peer checks the query feedback of a mapping more frequently.

In this work we experimentally gather experience on the choice of f for complete histograms only, Sec. 7.3.2. As a further enhancement, our technique can be refined by assigning an own minimum non-pruning frequency for different subareas of the data space, i.e., individual histogram buckets.

To implement the adaption phase of a histogram mentioned above, the minimum non-pruning frequency is set to $f = 0$ to completely prevent pruning. If the estimation error of the corresponding histogram reaches an acceptable level, f can be increased again.

7.3.2. Experimental Evaluation

This paragraph addresses the conflict between pruning mappings and using query feedback for histogram tuning.

Experiment 7.3 (Avoiding masking changes by pruning). We extend the experiment on detecting changes from Sec. 7.2.5 by including our technique based on the minimum non-pruning frequency f. We study how f influences the estimation accuracy of mappings frequently being pruned.

Methodology. We return to PDMS $\mathcal{P}_3$ and consider the same mapping from $Peer005$ to $Peer006$ that lies one peer mapping away from the change. Again, we examine how $Peer005$ can detect the hidden change using the query feedback for this mapping. The pruning threshold for this mapping is chosen such that the mapping is *always* pruned, if pruning is not disabled. First, we check the behavior for a minimum non-pruning frequency $f = \infty$. To this end, f is actually set to 1000 for the corresponding cardinality histogram. This means that repeated pruning of the same mapping in different user queries is *never* disabled. Next, we decrease f to 10, 5, and 3 and apply the same setting. We compare the estimation errors of the above mapping for the different values of the pruning frequency.

Discussion. The results of this experiment is depicted in Fig. 7.8. When disabling of pruning is applied with $f = 10$, the histogram of our mapping has only 5 query feedbacks between the user queries 100 and 200. Right after the change at query 200, the completeness difference is above the pruning threshold several times. So no pruning applies and the histogram can use query feedback for tuning and determining a relative estimation error. However, since there are so few estimation errors available before the change, the *smoothed* average of the estimation error increases uniformly between query 100 and 200. Despite this ramp, it can be detected from the graph that a change around query 200 must have happened. However, after the increase the estimation error does not decrease significantly. This is due to the fact that the histogram only can exploit 8 query feedbacks between query 250 and 400, which obviously remain quite high in that run of the experiments. Repeating the experiment several times did not change this behavior.

For $f = 5$ the adaption to the change is better but the change is more difficult to detect, because the increase of the estimation error after the change is moderate. This is again due to the very small number of query feedback events before the change. Observe that the detection could be improved by considering the estimation error without smoothing.

When the minimum non-pruning frequency is set to $f = 3$, there are about 15 query feedbacks between query 100 and 200 and therefore (1) the change can clearly be detected and (2) there is also enough query feedback for the estimation error to quickly adapt to the change. The local maximum around query 350 is due to an individual estimation error of about 200%. All other estimation errors beyond query 250 are clearly below 100%. Additionally, the curve for the experiment *without pruning* in Fig. 7.8 shows that the estimation error in this experiment decreases slowly after the change at query 200. With $f = 3$ we still observe average cost savings of about 50% while the average completeness of the query answer is about 85%. In summary, this experiment shows that the technique of ensuring a minimum frequency of query feedback works well only with small values for the minimum non-pruning frequency.

7.4. Related Work

Nie et al. gather multivariate statistics using a data mining approach over the queries posed to a mediator. Their work is discussed in detail in Sec. 5.4.1. Here, we only examine the issue of statistics maintenance. In contrast to our work, the authors do not prove adaptability of their statistics on coverage and overlap in case of large changes in the underlying data sets [NKN05, NK04]. They solely mention that computing even one-dimensional statistics for a fine grained decomposition of the data space is very time-consuming, e.g., it can take hours. It remains unclear how long multi-dimensional statistics take to be built and how much query feedback it takes to achieve sufficient accuracy. The authors point out that the statistics can be computed offline and that they could consider incremental updates as new query feedback comes in.

We believe that PDMS face the challenge to adapt statistics to massive changes in the data distribution in an online manner. So such systems are forced to incrementally update their statistics, which seems to be an open challenge in the results presented in [NKN05, NK04]. In our work, we use self-maintaining cardinality histograms that immediately adapt to incoming query result. For a comparison of different approaches of self-maintaining histograms see [Lem07, IP95].

The work in [TIGK02] deals with creating dynamic multi-dimensional histograms over

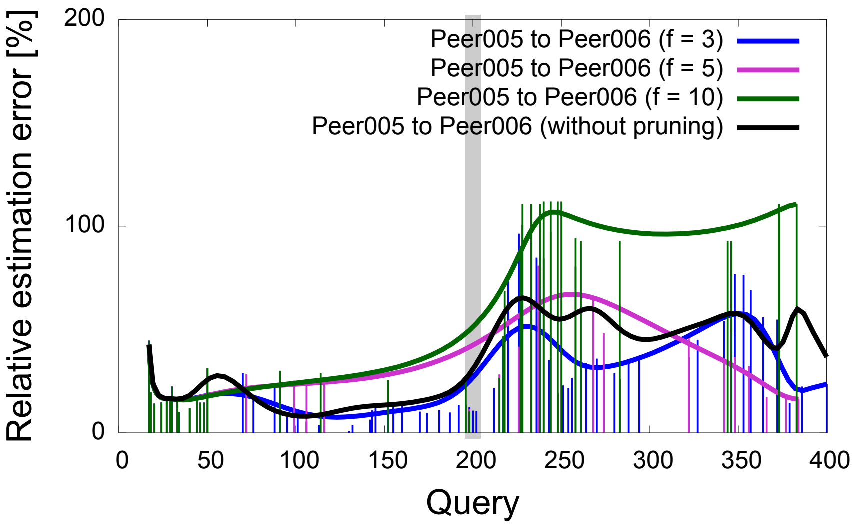


Figure 7.8.: Relative estimation error in dependency of the minimum non-pruning frequency for PDMS $\mathcal{P}_3$. The graphs represent the average over the last 5 values. Outliers have been removed before averaging. Each smoothed graph is accompanied with the raw impulses in the same color. The change at query 200 is marked by a grey bar in the background.

data streams, i.e., their focus is on statistics maintenance. The authors employ a so-called dynamic summary data structure to capture statistics of the data stream. If desired, algorithms can be applied to derive a histogram from that structure for which approximate guarantees on the estimation quality can be given. The paper compares the STHoles histogram [BCG01] with their own approach, which for some setups outperforms STHoles in terms of accuracy. The main difference to our setting is that we face the problem that changes to the underlying data structure *cannot* be detected, because query feedback is missing due to repeated pruning. In contrast, in the setting in [TIGK02] every change is available to the agent maintaining the statistics.

Pitoura et al. [KPPT05] use histograms both as routing indexes [PKP04] and to construct small world overlays in a P2P system. An overlay consists of a network of peers and is as such comparable to the network of peer mappings in a PDMS. In contrast to our work, they create their histograms based on the content of the peers within the horizon of the histogram. In this way, that approach contradicts the peer-to-peer paradigm, that each peer only knows about its direct neighbors. Their solution that peers offer their local index for the construction of routing indexes on a global level also does not fit to the main assumption of autonomous peers. Additionally, there is no information in the paper on the maintenance of the histograms in case of changes in the system, which is the major focus in this chapter.

In [HLS06] statistics about data potentially being returned by neighboring peers is maintained by exploiting update information that the peers propagate through the network. So these statistics require other peers to actively publish information about their data distribution rather than exploiting query feedback. This assumption is very optimistic as experience from practice shows. For instance, a peer turning offline would have to publish all of its information as “deleted”. That work also does not consider complex mappings between the peers.

7.5. Summary

The goal of this chapter was to prove that query feedback—exploited to its full potential—can serve to accurately maintain multi-dimensional statistics on cardinality in the context of PDMS. To this end, we presented the challenges set by the high degree of volatility in PDMS as well as their distributed nature. We solved the problem of detecting hidden changes arising from the peer-to-peer paradigm by keeping track of the estimation error at a fine grained level. Additionally, we proposed several techniques to support the adaption of our histograms to a changing data distribution in the PDMS. An extensive experimental study has shown that based on our techniques, the relative estimation error clearly remains below an order of magnitude, even after massive changes in the PDMS. This is considered to be sufficient for our techniques of query optimization presented in this thesis.

Further, we discussed an approach to the problem of trading off pruning and query feedback by interweaving pruning mappings and obtaining enough query feedback to detect sudden changes and to adapt histograms. To this end, pruning can be disabled in a frequency that can controlled in dependency of the current estimation accuracy.

Having the means of accurate statistics in dynamic PDMS, we now present our approaches to query optimization under cost constraints (Chapter 8) and with information overlap (Chapter 9).

8. Query Optimization under Limited Resources

To preserve peer autonomy, we focus on purely local reasoning in query planning. Therefore, we assume the number and length of mapping paths "behind" neighboring peers to be unknown from the local perspective of a peer. For this reason, our approach for threshold-based pruning introduced in Chapter 6 has two major shortcomings:

– Benefit and cost of query answering are difficult to control. They are heavily dependent on the completeness threshold that controls pruning of peer mappings. Furthermore, the suitable value for this threshold is specific for the query at hand.

– The result size and the cost of query answering can be arbitrarily poor. In extreme cases our approach may prune every result available in the PDMS. On the other hand, pruning too few mappings may lead to a unacceptable response time.

Bounding the cost of query answering is an important requirement in practice. Users usually expect information systems to return an answer in sub-second time. If a system shows longer response times, its acceptance is likely to drop. So particularly for PDMS potentially high execution cost conflict with user expectations of sub-second query answering.

To address these problems, this chapter introduces an approach to limit resource consumption of query answering while hardly compromising our requirement for local reasoning. We assume that the peers accept a time budget for processing queries received from other peers or directly from the user. This budget is passed along with each query. Together with the queries, the corresponding budget is recursively distributed to downstream peers.

In this chapter we first present in an illustrative example how this approach can be used to explore the PDMS intelligently. Then we propose two major alternatives to distribute a query's budget to a set of alternative query reformulation options. To establish non-monotonic reasoning for query optimization, budget can also be given back to the peer it came from. We present two alternatives for doing so. Finally, we report on the extensive experimental evaluation we conducted on these algorithms using our PDMS testbed *Humboldt Peers*.

8.1. Bounding Resource Consumption

In this section we first define the completeness maximization problem for an individual peer answering a query. Then we turn our attention to the whole PDMS and illustrate our technique of budget-driven query answering by an example.

8.1.1. Problem Definition

In principal, estimating the cost of distributed query answering is difficult if only local information and query feedback can be exploited, Sec. 4.6. Even if statistics on cost are accurate, a change in the PDMS structure can lead to dramatically higher resource consumption for query answering due to the exponential nature of this problem. For query execution cost, the same implications of high PDMS volatility apply as discussed in Sec. 7.1. Even small changes with only an individual or a few peers can have massive impact on the data accessible over a certain mapping path.

Definition 25 (Budgeted Completeness Maximization Problem)**.** *Given a query Q over the schema of a peer, a budget B to be spent for accessing a subset of the given outgoing mapping sets $\mathcal{M}_1, \mathcal{M}_2, \ldots$ of the peer. And given estimations on their coverage and density scores as well as their query execution cost $Cost(\mathcal{M}_i)$ are available, find a local query plan P for Q with maximal estimated completeness $C(P)$ together with a corresponding assignment of the budget B to the mapping sets $\mathcal{M}_i \in P$.*

To overcome these difficulties and the shortcomings of our approach for threshold-based pruning mentioned above, the following sections show how to spend a given budget to maximize the benefit of query answers while guaranteeing a limited amount of resource consumption. To this end and in the spirit of the Mariposa system [SAP$^+$96], we propose a budget-driven approach, where peers are assigned a budget to use for query answering.

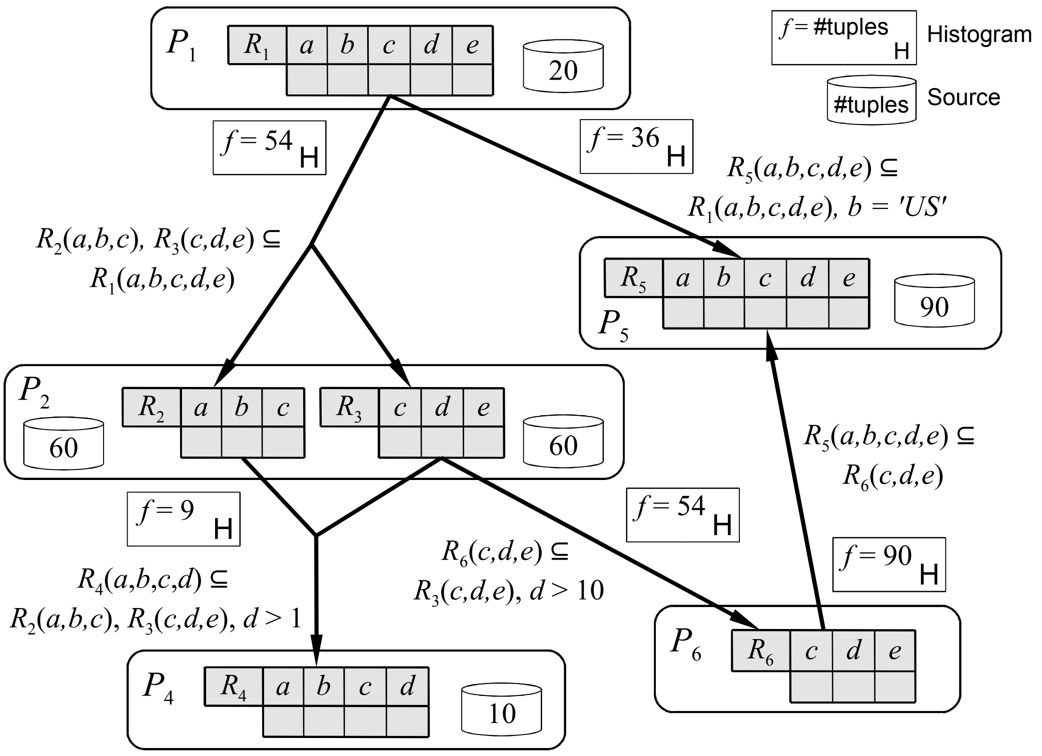
8.1.2. Budget-driven Query Answering

Every peer is free to decide about how to spend the budget it receives with the query. Additionally, if a peer has no outgoing mapping, it is reasonable to give budget back to the next upstream peer from which the budget was passed. This enables a non-monotonic optimization strategy. If a peer has spent budget to a mapping path that cannot exploit it to the full extent, the peer will get back the unused budget. Then it can allot this refunded budget to other mapping paths that were ignored before. If the cost measure is just response time, refunding of budget is implicitly done by finishing query processing before the given time has passed.

Intuitively, our approach to exchange budget along with queries and their results is a means to establish a weak coordination between peers. In this chapter we examine different fully local strategies to spend and possibly refund budget. The strategies differ in the way peers allot budget to their outgoing mappings.

The main idea behind our budget spending strategies is to distribute the budget to alternative mappings based on the expected amount of data they contribute to the local query result at a peer. To estimate this contribution, we use the completeness model from Sec. 4.1 and our cardinality histograms introduced in Chapter 5. Before we present our new techniques in detail, the following example gives an overview on how budget-driven query optimization works.

Example 19. *To illustrate our approach, we introduce an example of a simple* PDMS *with some schemas and mappings between them, and guide the reader through the process* *of query answering while considering the completeness of the intermediate query result (Fig. 8.1).*



P_1
R_1 a b c d e
20
f = #tuples H
Histogram
#tuples
Source
f = 54 H
f = 36 H
$R_5(a,b,c,d,e) \subseteq R_1(a,b,c,d,e)$, b = 'US'
$R_2(a,b,c)$, $R_3(c,d,e) \subseteq R_1(a,b,c,d,e)$
R_5 a b c d e
90
P_5
P_2
R_2 a b c
R_3 c d e
60
60
$R_5(a,b,c,d,e) \subseteq R_6(c,d,e)$
f = 9 H
f = 54 H
f = 90 H
$R_6(c,d,e) \subseteq R_3(c,d,e)$, d > 10
$R_4(a,b,c,d) \subseteq R_2(a,b,c)$, $R_3(c,d,e)$, d > 1
P_6
R_6 c d e
P_4
R_4 a b c d
10

Figure 8.1.: A small PDMS with mappings, source cardinalities, and symbolic histograms.

System Description. *Each peer mapping is shown in Datalog notation. The mappings can contain projections, for instance the mapping $P_2 \to P_4$ does not map attribute e. Also, mappings may have selection predicates. For instance, the mapping $P_2 \to P_6$ has a selection predicate $d > 10$. The cardinality histogram at each peer mapping is symbolically depicted in Fig. 8.1. We only annotate the frequency in a single root bucket, which represents the whole STHoles histogram, Sec. 5.1.2. Further, we assume that our* PDMS *overall holds data about 100 real-world entities. For each data source, the number of tuple it offers is annotated. For example, based on its histogram, peer P_2 estimates to receive 54 tuples from P_6 for queries* without *any selections. However, since P_6 itself expects 90 tuples that are actually stored at P_5, we observe that the selection predicate in the mapping $P_2 \to P_6$ removes all but 60% of the tuples available at P_6.*

Query Planning. *We now regard query planning with a simple cost model and our completeness model, discussed in detail in Sec. 4.1. We assume a simple cost model: Using a set of peer mappings has a cost of 1, independent of the amount of result data transferred. A fixed budget assigned along with the initial user query serves to bound the resources spent for query planning and evaluation. To ensure that we spend the budget wisely, i.e., that we retrieve as many results as possible, our completeness model counts the number of retrieved attribute values in relation to the overall number of attribute values. Please* *note, that it is not necessary to know the latter, because it only acts as a normalizing factor.*

Since peer P_5 stores 90 tuples, each with data across all five attributes, it has a completeness of 90%. However, if this data is passed through the mapping to P_1, the number of tuples is reduced to 36 because of the selection predicate in the peer mapping $P_1 \rightarrow P_5$. Thus, P_5 has a completeness of approx. 0.29 (180/500 attribute values) from the perspective of P_1. Please note that in general this effect of decreased completeness is accumulated along mapping paths.

Now, consider a query

$$q(a, b, c, d, e) :- P_1.R_1(a, b, c, d, e)$$

posed to peer P_1 asking for all objects of relation R_1. Let the budget of this query be 4, i.e., we can recursively access four remote peers to answer the query. The database at P_1 can answer the query itself with a completeness of 0.2 (20 · 5/500 attribute values). P_1 has mappings to two other peers, P_2 and P_5, and it must now decide, which paths to follow and how much budget to allot to each path.

Using one of the strategies outlined below, P_1 decides to pass along the query only *to P_2, because it promises to return 54 tuples compared to only 36 tuples estimated by the histogram of the alternative mapping to P_5. Since we assume full autonomy of peers, we employ a fully local optimization strategy, Sec. 1.2. However, the histograms give us a means to somehow globally assess the data accessible over a peer mapping.*

By passing the query to P_2, the aggregated coverage is incremented to include the data stored at P_2:

$$c(P_1.R_1 \sqcup (P_2.R_2 \sqcap P_2.R_3)) = 0.2 + (\underbrace{0.6}_{c(P_2.R_2)} \cdot \underbrace{0.6}_{c(P_2.R_3)}) - 0.2 \cdot (0.6 \cdot 0.6). \tag{8.1}$$

Since both P_1 and P_2 comprise all 5 attributes, we yield an intermediate completeness $C(P_1.R_1 \sqcup (P_2.R_2 \sqcap P_2.R_3)) = 0.49$. The subtraction in Eqn. (8.1) accounts for duplicates among P_1 and P_2, assuming for now independence of tuples stored in P_1 and P_2.

Next, P_2 has a remaining budget of 3 and decides to spend it all for the path to P_6, because the histogram of the mapping $P_2 \rightarrow P_6$ promises 54 tuples instead of only 9 tuples for the mapping $P_2 \rightarrow P_4$. As P_6 lacks a data source of its own, it cannot contribute to the overall completeness and passes the query to P_5 along with a budget of 2. By these steps, the intermediate completeness is increased to $C(P_1.R_1 \sqcup (P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5))) = 0.59$. Thus, the large data set of P_5 is reached after all, however, along a different path. This alternative path conserves more data on the way back to P_1 than the direct path $P_1 \rightarrow P_5$.

Since P_5 has no outgoing mapping, it can refund *the budget of 2 to P_6, which in turn also gives the budget back to P_2 along with the query answer. Peer P_2 can exploit this budget by passing the query to the previously ignored path to peer P_4. Finally, P_4 increments completeness to $C(P_1.R_1 \sqcup ((P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5)) \sqcup P_4.R_4)) = 0.62$, accounting for the fact that it supplies only four of the five attributes and only 10% of the overall data items in the system. Now the entire budget is used up and the query* *results flow back to P_1 from where the query originated.* □

While this simple example is meant to convey the main idea of the approach for query optimization under limited resources, real PDMS must deal with additional difficulties that we discuss in the following sections.

8.2. Spending Budget

We first discuss major requirements and rules for strategies for spending a budget to sets of mappings. Then we present two budget spending approaches in detail.

As pointed out in Sec. 6.1, we can assume that peers are able to enumerate the search space completely, because they are only connected to a small number of neighbors and their peer schema contains only contain foreign key chains of small length. For these reasons, we employ a two step approach:

1. The peer computes the fully expanded query plan for a received query as presented in Sec. 2.3.
2. A budget spending algorithm selects a fraction of the fully-expanded local query plan along with an assignment of the available budget to the sets of mappings used in the rewritings of that subplan.

The user expects results for all variables of the query posed to a peer. Moreover, we have to assume that every variable can be concerned by selection predicates. Therefore, it is important that the budget is spent in such a way that similar amounts of data are retrieved for the different subgoals of the query at hand. Then the probability that the multi-way join in a conjunctive query yields many results is maximized. The importance of this aspect is highlighted by following example.

Example 20. *We are given a query plan $P(Q) = R_1 \sqcap R_2$ as well as the two mapping results $R_1(\underline{a}_1, a_2, a_3)$ and $R_2(\underline{a}_3, a_4)$. Suppose the budget had been spent such that the mapping results have following coverage and density scores:*

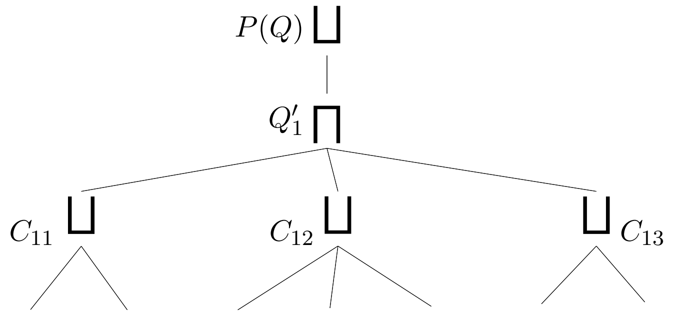
	$c(S)$	$d(a_1)$	$d(a_2)$	$d(a_3)$	$d(a_4)$	$d(S)$	$C(S)$
R_1	*0.1*	*1*	*1*	*1*	*0*	*0.75*	*0.08*
R_2	*0.8*	*0*	*0*	*0.3*	*0.1*	*0.1*	*0.08*

R_1 has low coverage but returns many attribute values per tuple. In contrast, although yielding the same completeness as R_1, the mapping result R_2 shows a completely different configuration. It has high coverage, i.e., returns many tuples, but the density with respect to the attributes a_1, a_2, a_3, and a_4 is comparably low. The completeness of $P(Q)$ is assessed as $C(R_1 \sqcap R_2) = 0.05$. Interestingly, combining the contributions from R_1 and R_2 by a join merge yields less completeness than both mapping results considered in isolation. The reason for this is the large difference in the coverage scores of R_1 and R_2 that leads to a low coverage of the result of the join-merge. □

Recall the structure of a fully expanded local query plan at a peer as depicted in Fig. 2.6 (on Page 31). A query plan is a full outerjoin-merge of different rewritings, each of which is in turn a join-merge over contributions of several subgoals C_{ij}: $P(Q) = \bigsqcup_i (C_{i1} \sqcap C_{i2} \sqcap \ldots \sqcap C_{ik})$. Each subgoal retrieves data from alternative mapping sets that are combined by a full outerjoin-merge: $C_{ij} = \bigsqcup_l (R(\mathcal{M}_l), Sel(\mathcal{M}_l))$. A budget spending approach must decide how to distribute the budget to the mapping sets at the lowest level of the query plan. In particular, it can prune mapping sets by *not* allotting any budget to them.

Example 21. *Regard the fully expanded query plan in Fig. 8.2. It shows a query plan with an individual rewriting Q'_1 that has three subgoals below. Each of them combines results from the mapping results listed in the table along with their potential data contributions ΔC, cost, and $\Delta C/Cost$ ratio. The latter acts as a measure for efficiency of a mapping set in the query processing: The higher the value for $\Delta C/Cost$, the more data is retrieved based on the cost.*

For instance, the rewriting C_{12} retrieves data from the mapping sets $\mathcal{M}_3$, $\mathcal{M}_4$, and $\mathcal{M}_5$. We shall use this query plan to explain our budget spending algorithms below. □



Result	$R(\mathcal{M}_1)$	$R(\mathcal{M}_2)$	$R(\mathcal{M}_3)$	$R(\mathcal{M}_4)$	$R(\mathcal{M}_5)$	$R(\mathcal{M}_6)$	$R(\mathcal{M}_7)$
c	0.3	0.8	0.1	0.65	0.7	0.3	0.9
ΔC	0.05	0.47	0.008	0.14	0.176	0.023	0.49
$Cost_{\max}$	3	6	2	5	16	4	7
$\Delta C/Cost$	0.016	0.078	0.004	0.028	0.011	0.006	0.07

Figure 8.2.: Example of a fully expanded local query plan $P(Q)$. The mapping results $R(\mathcal{M}_i)$ are listed along with their respective coverage c, data contribution ΔC, their cost for retrieving their maximal completeness, and their $\Delta C/Cost$ ratio.

An important observation to keep in mind when deciding on how to distribute budget is that—usually—yielding high completeness is accordingly expensive. However, the main idea behind our pruning approaches is to find exceptions from this heuristic and prune mapping paths from query processing if they promise small results with disproportionately high cost. For instance, in the query plan in Fig. 8.2 the mapping set $\mathcal{M}_4$ is three times more efficient than $\mathcal{M}_5$ and yields 86% of the data of $\mathcal{M}_5$.

Our two approaches for spending the budget differ in how they distribute the budget to the available mappings. The first strategy follows a breadth-first approach and distributes the budget in proportion to the potential data contribution of the mapping paths. The second algorithm takes a depth-first approach and exclusively chooses the mapping paths with the highest completeness estimation whose cost is below the limit given by the budget.

Both algorithms return an assignment of budget to every rewriting constituent of a query plan as follows:

Definition 26 (Budgeted Query Plan)**.** *Let P be a local query plan. Let b be a mapping $\{R(\mathcal{M}_i)\} \rightarrow float$ from the mapping sets $\mathcal{M}_i \in P$ to a corresponding budget for further query processing. Then the pair $\langle P, b\rangle$ is called a budgeted query plan. It fulfills the budget constraint*

$$\sum_{\mathcal{M}_i \in P} b(\mathcal{M}_i) \leq B, \tag{8.2}$$

where B denotes the given budget.

The budgets are passed to the neighboring peers along with the queries. These peers are expected to follow this budget constraint.

8.2.1. Weighted Distribution of Budget

In this strategy, a peer distributes its budget in a breadth-first manner. It starts from a fully expanded local query plan denoted by $\hat{P}$. Every result of a mapping set in this plan is assigned a fraction of the given budget. The algorithm is called Weighted because that fraction of budget is proportional to its expected data contribution. Weighted is documented in detail in Algorithm 8.2.1. It returns a mapping $b : \{R(\mathcal{M}_i)\} \rightarrow float$ from the mapping sets $\mathcal{M}_i$ in the given query plan to a budget for downstream query processing that satisfies the budget constraint.

On the top level of the query plan, the budget is simply distributed in proportion of the potential data contribution of the rewritings of the plan (Lines 1-2). Since the alternative data contributions of a union-like full outerjoin-merge are independent, we can maximize its result size by maximizing the result sizes of the operands in that way.

Things are different with the join-merge operator. As for the usual join, the result size depends on how well the data of the conjuncts “fit together”. In a centralized database that guarantees referential integrity for foreign key relationships, the cardinality of a relation connected by a 1-to-1 foreign-key relationship must be at least as large as the relation where the foreign-key relationship starts from. This could be considered when distributing the budget among the conjuncts. However, for 1-to-many relationships it is in general not possible to assess the lower bound for the cardinality of the target relations. And in data integration, we even cannot assume that referential integrity holds. Rather, when retrieving data for relations connected by a foreign key from *different* peers, the cardinality of the join result can be much smaller than the size of the joined relations as Example 20 demonstrates above.

```
**Input** : Query $Q$, fully expanded query plan $\hat{P}(Q)$, budget $B$
**Output**: Budget mapping $b$ for $\langle \hat{P}, b \rangle$
1  foreach rewriting $Q'_i \in \hat{P}$ do
2      $B_{Q'_i} \leftarrow \Delta C(\hat{P}, Q'_i) / \sum_{Q'_k \in \hat{P}} \Delta C(\hat{P}, Q'_k) \cdot B$        {weighted distribution}
3      $n_s \leftarrow 5$        {#steps to distribute $B_{Q'_i}$}
4      $b_C \leftarrow \perp$        {budget mapping $\{C_{ij}\} \rightarrow float$}
5      for $i \leftarrow 1$ to $n_s$ do
6          foreach $C_{ij} \in Q'_i$ do
7              $b_C(C_{ij}) \leftarrow b_C(C_{ij}) + \Delta C(\hat{P}, C_{ij}) / \sum_{C_{ik} \in \hat{P}} \Delta C(\hat{P}, C_{ik}) \cdot \frac{1}{2} B_{Q'_i} / n_s$
8              $b_{ij} \leftarrow b_C$
9              $b_{ij}(C_{ij}) \leftarrow b_C(C_{ij}) + \frac{1}{2} B_{Q'_i} / n_s$        {what-if analysis wrt. each $C_{ij}$}
10         end
11         find $j$ with $\sum_{k \neq j} \left| \frac{s_{ij}}{s_{ik}} - \frac{|\langle C_{ij}, b_{ij} \rangle|}{|\langle C_{ik}, b_C \rangle|} \right| = min_{C_{ij} \in Q'_i}$        {$\langle C_{ij}, b_{ij} \rangle$: bud. subplan}
12         $b_C(C_{ij}) \leftarrow b_C(C_{ij}) + 1/n_s \cdot B_{Q'_i}$        {spend fraction of budget}
13     end
14     foreach $C_{ij} \in Q'_i$ do
15         foreach $\mathcal{M}_l \in C_{ij}$ do
16             $b(\mathcal{M}_l) \leftarrow \Delta C(\hat{P}, R(\mathcal{M}_i)) / \sum_{\mathcal{M}_k \in C_{ij}} \Delta C(\hat{P}, R(\mathcal{M}_k)) \cdot b_C(C_{ij})$
17         end
18     end
19 end
20 return $b$
```

Algorithm 8.2.1: Allotting budget in proportion to the expected contribution to the query result (WEIGHTED).

Therefore, we do distribute the budget among the subgoals of a rewriting *not only* in proportion to their data contribution. Rather,

– we follow our main goal of maximizing the completeness of the query plan while spending the budget.

– But we also aim to fulfill a boundary condition on the result sizes of the subgoals as presented below.

This constraint specifies the ratio between the result sizes of the subgoals. The result cardinality of the individual subgoals changes with different budgets assigned to them. Therefore, we propose to iteratively assign small fractions of the rewriting's budget to a particular subgoal (Lines 7 and 12).

During this process we both follow our optimization goal and try to minimally deviate from the boundary condition. In each step, we first assign a fraction of the budget in proportion to the data contribution of that subgoal in the given plan. This reflects WEIGHTED's approach to achieve maximal completeness of the query plan *without* exhaustively enumerating all plans. Second, we allot another fraction that is oriented to fulfill the budget condition. This is performed by an exhaustive what-if analysis (Lines 6-11). To this end, a budget mapping b_{ij} for each subgoal C_{ij} of the rewriting is created by copying the main budget mapping b_C (Line 8). Then, the fraction of the rewriting budget is added to *each* b_{ij}.

Please note that at this point, we assume to have statistics to estimate the completeness of a *budgeted* query plan. Finally, WEIGHTED distributes the budget assigned to every subgoal of the rewriting (as stored by b_C) in proportion to the data contributions of the results of the mapping sets (Lines 14-19).

The boundary condition serves to prune the search space of our optimization problem. It is a means to control the size of the joins between the rewriting subgoals. The join size between two relations is dependent on the ratio of their cardinalities. For instance, consider a join between two relations R and S connected by a 1-to-many relationship. We double the size of S while R remains unchanged. Statistically, the size of the join is expected to be doubled as well, since the probability for each new tuple to find a join partner in R is supposed to be the same as for the old tuples. Consequently, we assume a boundary condition that specifies the ratios between the cardinalities $|C_i|$ of the data contributions of the subgoals $C_{ij} \in Q'_i$:

$$|C_{i1}| : |C_{i2}| : \ldots : |C_{ik}| = s_{i1} : s_{i2} : \ldots : s_{ik}. \tag{8.3}$$

In practice, s_{ij} in Eqn. (8.3) are the output of a special approach for query optimization based on join selectivity estimation, which is beyond the focus of this thesis. Here, we assume that we do not know whether the type of the relationship between rewriting subgoals is 1-to-1 or 1-to-many. So to avoid highly selective joins as in Example 20 above, we aim at similar cardinalities of the data contributions of the rewriting subgoals, i.e., referring to Eqn. (8.3) we choose $\forall C_{ij} \in Q'_i : s_{ij} = 1$.

The deviation from this condition is measured for a certain rewriting subgoal by summarizing the differences of the ratios of the subgoal cardinalities from their target value for that subgoal with each other subgoal of the rewriting (Line 13 in Algorithm 8.2.1). To maximally reduce the deviation from the boundary condition from Eqn. (8.3), the algorithm WEIGHTED greedily assigns the next fraction of the rewriting budget $B_{Q'_i}$ to the subgoal that maximally reduces this difference. Please note that this technique can also be applied in the other algorithms for query planning (PRUNE) and optimization (GREEDY following in the next section and the algorithms in Chapter 9).

Example 22. *We apply the algorithm* WEIGHTED *to our query plan introduced in Example 21 above. In this example, we assume that the coverage of a result for a given budget can be assessed by the linear function*

$$c(B) = c_{\max} \cdot \frac{B}{Cost_{\max}}, \tag{8.4}$$

where $c_{\max}$ denotes the coverage that can maximally be achieved at the cost of $Cost_{\max}$. Since the fully expanded query plan contains only one rewriting Q'_1, the complete budget of $B = 20$ is assigned to Q'_1.

While in the first part of each step the budget is distributed in the same proportion during all steps, the second step considering the boundary condition chooses different rewriting subgoals to assign the budget to. Finally, Weighted *ends up with the following budget assignment:*

Subgoal	C_{11}	C_{12}	C_{13}
Assigned budget	*4.9*	*9.4*	*5.7*

To understand this result, recall the valuation of the result sets from Fig. 8.2. The subgoal C_{11}, especially $R(\mathcal{M}_2)$ returns data with a high coverage at moderate cost as reflected by the scores for $\Delta C/Cost$ measuring the efficiency of the mapping path. For the subgoal C_{13}, $R(\mathcal{M}_7)$ has similar characteristics. In contrast, the score $\Delta C/Cost$ for the subgoal C_{12} is less than half as good as for the two other subgoals. These are the reasons for the comparably small amounts of budget spent to C_{11} and C_{13}.

The cardinalities of the rewriting subgoals relate to each other as follows:

$$|C_{11}| : |C_{12}| : |C_{13}| = 0.61 : 0.63 : 0.70.$$

So the maximal deviation from the boundary condition from Eqn. (8.3) of about 13% is satisfying. Taken together, half of the budget has been used to follow the boundary condition, whereas the other half has been distributed in proportion to the potential data contributions. Of course, this ratio can easily be varied in the algorithm Weighted *by choosing the s_{ij} in Eqn. (8.3) to accentuate either the completeness maximization of the individual rewriting subgoals or the join merge of their results.* □

8.2.2. Greedy Distribution of Budget

Instead of expanding the query at hand in a breadth-first way as Weighted does, our Greedy strategy follows the *best* of a set of alternative mapping sets. It assigns the whole budget available for a rewriting subgoal to the mapping set promising the most data. Regard Algorithm 8.2.2, which we now explain in detail. It uses the functions $followBudgetConstraint$ and $increaseCostOfPlan$ that are documented in Algorithm 8.2.3 and Algorithm 8.2.4 respectively. As discussed above, the algorithm Greedy takes the fully expanded query plan as input.

The main idea of Greedy is to find a query plan that has the structure, i.e., the C_{ij}, of *one* of the rewritings in the fully expanded plan but only considers the best mapping set of each rewriting subgoal while satisfying the budget constraint. All rewritings of the complete query plan are included in the search for this best plan. Similarly to Weighted, the algorithm Greedy returns a mapping $b : \{R(\mathcal{M}_i)\} \rightarrow float$ from the mapping sets $\mathcal{M}_i$ in the *returned* query plan $P_{\max}$ to budget values for each result of a mapping set.

As explained in [Nau02], the principle of optimality does not hold for the budgeted completeness maximization problem, i.e., a less complete data source can better fit to an

```
    Input   : Query $Q$, fully expanded query plan $\hat{P}(Q)$ with $l$ query rewritings, an
              average of $k$ subgoals per rewriting, each with $n$ mapping sets, budget
              $B$, look ahead $l_a$
    Output: Pruned local query plan $P_{\max}$, budget mapping $b$ for $\langle P_{\max}, b\rangle$
 1  $C_{\max} \leftarrow 0$, $P_{\max} \leftarrow \bot$
 2  foreach rewriting $Q'_i \in \hat{P}(Q)$ do
 3      foreach subgoal $C_{ij} \in Q_i$ do                         {$\mathcal{R}_{C_{ij}}$: priority queue}
 4          $\mathcal{R}_{C_{ij}} \leftarrow sortByCompleteness(\{R(\mathcal{M}_l) \in \hat{P} | R(\mathcal{M}_l) \in C_{ij}\})$
 5      end
 6      $P \leftarrow \bot$, $i \leftarrow 0$                                  {$i$: look-ahead counter}
 7      repeat
 8          if $P = \bot$ then
 9              foreach subgoal $C_{ij} \in Q_i$ do
10                  $P \leftarrow P \sqcap \mathcal{R}_{C_{ij}}.pop()$
11              end
12          else
13              find $C_*$ with $\Delta C(\hat{P}, \mathcal{R}_{C_*}.first()) = max_{C_{ij} \in Q'_i}(\Delta C(\hat{P}, \mathcal{R}_{C_{ij}}.first()))$
14              $P[C_*] \leftarrow \mathcal{R}_{C_*}.pop()$                        {$P[C_*]$: subgoal $C_*$ in $P$}
15          end
16          $P \leftarrow followBudgetConstraint(P, \hat{P}, B, \{\mathcal{R}_{C_{ij}}\})$        {see Algorithm 8.2.3}
17          $P \leftarrow increaseCostOfPlan(P, \hat{P}, B, Q'_i, \{\mathcal{R}_{C_{ij}}\})$        {see Algorithm 8.2.4}
18          if $C(P) > C(P_{\max})$ then
19              $P_{\max} \leftarrow P$, $C_{\max} \leftarrow C(P)$, $i \leftarrow 0$
20          else
21              $i \leftarrow i + 1$
22          end
23      until $i = l_a$ or $\forall \mathcal{R}_{C_{ij}} : \mathcal{R}_{C_{ij}}.isEmpty()$
24  end
25  foreach $\mathcal{M}_i \in P_{\max}$ do
26      $b(\mathcal{M}_i) \leftarrow \Delta C(P_{\max}, R(\mathcal{M}_i)) / \sum_{\mathcal{M}_j \in P_{\max}} \Delta C(P_{\max}, R(\mathcal{M}_j)) \cdot B$
27  end
28  return $P_{\max}, b$
```

Algorithm 8.2.2: Allotting budget in a greedy manner (Greedy).

```
Input  : Pruned local query plan $P$, fully expanded query plan $\hat{P}$, budget $B$,
         priority queues $\{\mathcal{R}_{C_{ij}}\})$
Output: Pruned local query plan $P$
1  while $Cost(P) > B$ do
2      $d \leftarrow C(P)$                    {change $P$ with min. decrease of $C(P)$}
3      foreach $C \in P$ do
4          $P' \leftarrow P$
5          $P'[C] \leftarrow \mathcal{R}_C.first()$
6          if $|C(P) - C(P')| < d$ then
7              $C_* \leftarrow C$, $d \leftarrow |C(P) - C(P')|$
8          end
9      end
10     $P[C_*] \leftarrow \mathcal{R}_{C_*}.pop()$
11 end
12 return $P$
```

Algorithm 8.2.3: Adjusting a plan to the budget constraint ($followBudgetConstraint(P, \hat{P}, B, \{\mathcal{R}_{C_{ij}}\})$).

```
Input  : Pruned local query plan $P$, fully expanded query plan $\hat{P}$ budget $B$,
         rewriting $Q'_i$, priority queues $\{\mathcal{R}_{C_{ij}}\})$
Output: Pruned local query plan $P$
1  while $Cost(P) < B$ do
2      find $C_*$ with $\Delta C(P, C_*) = min_{C_{pq} \in Q'_i}(\Delta C(\hat{P}, C_{pq}))$
3      $P' \leftarrow P$
4      $P'[C_*] \leftarrow P'[C_*] \sqcup \mathcal{R}_{C_*}.pop()$
5      if $Cost(P') < B$ then
6          $P \leftarrow P'$
7      end
8  end
9  return $P$
```

Algorithm 8.2.4: Increase cost of a plan to maximally exploit the budget constraint ($increaseCostOfPlan(P, \hat{P}, B, Q'_i, \{\mathcal{R}_{C_{ij}}\})$).

existing plan than a more complete one. Therefore, once a plan with a local completeness maximum has been constructed, Greedy proceeds in a look-ahead fashion (Lines 21 and 23). It checks a pre-defined number of additional plans (constructed in Lines 8 to 17) for being better than the current maximum.

The strategy for constructing new plans taken by the algorithm is based on pre-sorting the respective mapping sets of each rewriting subgoal C_{ij} (Lines 3 - 5). The first plan is created by taking the mapping set with the highest completeness from each subgoal (Lines 9 - 11). To fulfill the budget condition, each new plan is modified such that its cost is below the given budget. To this end, the function $followBudgetConstraint$ (Algorithm 8.2.3) substitutes the mapping result that minimally reduces the completeness of the plan.

A plan created by taking a result of a mapping set for each rewriting subgoal can be cheaper than the given budget. In such cases we extend the plan in function $increaseCostOfPlan$ (Algorithm 8.2.4) by including additional results of mapping sets in the rewriting subgoal with the minimal data contribution. At this point we again address the above requirement to uniformly retrieve data over all subgoals of the rewriting. More data can be expected from a conjunctive query if the cardinality of the results for the subgoals is similar than in the case where some subgoals contribute much less data than others as demonstrated in Example 20 above.

After a plan has been chosen, the algorithm finishes by assigning a budget to each mapping set in the plan in proportion to its potential data contribution ΔC (Lines 25 - 27). Recall that the rewriting subgoals can cover parts of the query to be answered of very different size. So between the rewriting subgoals it is appropriate to distribute the budget weighted on the data contribution of the mapping sets.

Example 23. *We apply algorithm* Greedy *on our query plan introduced in Example 21 above. Assume we a given a budget of $B = 20$. An initial plan is created by taking the mapping set with the highest completeness measure ΔC for each rewriting subgoal:*

$$P_1 = \underbrace{R(\mathcal{M}_2)}_{C_{11}} \sqcap \underbrace{R(\mathcal{M}_5)}_{C_{12}} \sqcap \underbrace{R(\mathcal{M}_7)}_{C_{13}}.$$

This plan yields a completeness of $C = 0.5$ Our cost model assesses $Cost(P_1) = 29$, which is higher than the given budget. Therefore, Greedy *is looking for a substitute for one of the conjuncts in the plan that preserves as much completeness of the whole plan as possible. It finds that the plan $P_2 = R(\mathcal{M}_2) \sqcap R(\mathcal{M}_4) \sqcap R(\mathcal{M}_7)$ is nearly as complete as P_1, i.e., $C(P_2) = 0.468$. Yet, P_2 is much cheaper, because $R(\mathcal{M}_4)$ has a much better completeness/cost ratio than $R(\mathcal{M}_5)$. With $Cost(P_2) = 18$ we have still 2 units of cost left. So* Greedy *can include the rewriting $R(\mathcal{M}_3)$ with cost of 2 units and we end up with*

$$P_3 = \underbrace{R(\mathcal{M}_2)}_{C_{11}} \sqcap \underbrace{R(\mathcal{M}_3) \sqcup R(\mathcal{M}_4)}_{C_{12}} \sqcap \underbrace{R(\mathcal{M}_7)}_{C_{13}}.$$

The completeness of this plan is assessed as $C(P_2) = 0.493$, so almost the same as the much more expensive plan P_1.

During a look-ahead phase, this plan is compared with others that are constructed by substituting the mapping sets below a rewriting subgoal by the next best out of the ordered collections of the subgoals. It can turn out that a certain mapping set better fits into the current plan than another one since it has a density pattern in its attributes that better complements the plan. In this example we assume that this does not happen and thus Greedy *returns P_3 as the best pruned plan.* □

Runtime Analysis The runtime of the algorithm Greedy depends on

– the look-ahead parameter l_a,

– the parameters determining the size of the fully expanded query plan: the number l of query rewritings, the average number k of subgoals per rewriting, the average number n of mapping sets per subgoal,

– and, importantly, the configuration of the mapping sets in the given query plan.

In essence, the algorithm creates containers comprising n mapping sets for each of the k subgoals of a rewriting. To create query plans, these mapping sets are selected. The outer *foreach* loop (Line 2) is performed once for each of the l query rewritings of the query plan. The inner *repeat* loop (Line 7) chooses at least a single mapping set from the containers. Due to the nested *while* loops (encapsulated by the functions $followBudgetConstraint$ and $increaseCostOfPlan$), more than one mapping set can be assembled into a plan. However, both the *repeat* and the *while* loop are controlled by the same collection of mapping sets. So performing more *while* loops reduces the number of possible *repeat* iterations. In addition, the look-ahead parameter l_a can terminate the *repeat* loop well before all mapping sets in the containers are used up.

In the worst case, the look-ahead mechanism does not terminate the *repeat* loop at all. Rather, the *repeat* and the *while* loops are executed until every mapping set has been inserted into a plan, because every plan created is better than all previous ones. Whenever a mapping set is selected, a nested loop over all subgoals of a rewriting is performed to find a maximum or a minimum among them (Lines 13, Lines 3 - 9 in $followBudgetConstraint$, and Line 2 in $increaseCostOfPlan$). In summary, the worst-case complexity is $l \cdot k \cdot n \cdot k = O(lnk^2)$. Since Greedy does not perform an exhaustive enumeration of all possible query plans its runtime is polynomial rather than exponential as an exhaustive search would be.

8.2.3. Effectiveness and Comparison

The ratio between the given budget and the size of the search space, i.e., the complexity of the Pdms, is supposed to be a factor influencing the effectivity of query planning approaches. For instance, if the budget is very small compared to the size of the Pdms, a depth-first search as in Greedy promises to better find peers returning a comparably high amount of data, because Greedy acts more selective than Weighted.

Additionally, the *rank* of a Pdms (the average number of peer mappings of a peer) has impact on the effectivity. The more interconnected the peers, the more likely it is that there are mapping paths that reach peers that otherwise would not have been found. Another consequence is that there are both more long mapping paths and more short mapping paths. If the budget is small and there is a high fraction of mapping paths of short length, Weighted promises better effectivity compared to Greedy than if there are less short mapping paths. Short mapping paths support the breadth-first approach of Weighted to find more data. In contrast, the existence of more mapping paths means no major advantage for Greedy, since this approach only has low fan-out.

Most notably, the fraction and distribution of loss of information in the peer mappings strongly influences the effectiveness of our strategies. The more information loss is encountered, the better the strategies work. On the other hand, if there would be no information loss, the Weighted approach would behave just like the expansion of all mappings. A disadvantage of Greedy is that the Pdms might not be explored equally, because the different recursive reformulation paths operate independently at different peers.

Weighted prefers to explore the neighborhood of a peer rather than more remote peers and thus results in shorter mapping paths with less likely loss of information (and higher semantic relevancy) and *lower cost*. As a further advantage, this approach enables parallel usage of alternative mappings. However, in cases where a peer faces to a high fraction of mappings with information loss, it is forced to spend budget on poor mappings. Thus, this approach should be used in conjunction with a threshold-based pruning strategy [RN05]. Additionally, it has to be considered that mappings resulting in local-as-view style reformulations [HIST03] cover several relations at a certain peer. Tuples retrieved over such mappings promise to contribute more completeness to the overall result than mappings that substitute only a single relation in a global-as-view fashion. Some of the above topics are subject of an experimental evalution in Sec. 8.5.

8.3. Refunding Budget

In case a peer has no or not enough outgoing mappings, some budget possibly cannot be used for further query processing. It can be important information for the peer the query has been passed from that it allotted too much budget for that query. Regard that our algorithms are performed strictly locally at the peers. However, under limited resources it is better to allow a weak form of cooperation to intelligently spend a limited budget. In our altruistic variations of Greedy and Weighted a peer can refund budget to where it came from if it has no more (promising) mappings to follow or it has detected a cycle in query planning. In this way, the calling peer is provided with information on how well the budget was exploited. Peers receiving and processing queries are interested in letting the calling peer know about their resource consumption.

8.3.1. Altruistic Greedy and Altruistic Weighted

In this section we describe how to extend our algorithms Greedy and Weighted can be extended by budget refunding. In general, peers getting back budget have the following options:

- *Revise earlier pruning decisions*: The refunded budget is spent to mappings that were ignored initially. These mappings may either explicitly have been pruned by threshold-based pruning (Sec. 6) or they have been omitted in distribution of budget.
- *Refund budget*: A peer can in turn refund the budget that has been given back from its neighbors if it already has used each of its outgoing mappings. In effect, budget may be refunded back over several peers.
- *Re-send queries with higher budget*: Queries already sent to a neighboring peer can be re-issued with higher budget if it can be expected that more data will be returned.

Next, we discuss these options and relate them to our algorithms for budget spending. The altruistic variation of budget refunding can be combined with both Weighted and Greedy. However, the above options are more or less suitable in different situations.

Revise Earlier Pruning Decisions The benefit of revising earlier pruning decisions heavily depends on the amount of data these mapping paths can contribute. If threshold-based pruning has been applied in conjunction with Weighted, the threshold influences the utility of mappings available to be used with the refunded budget. If the pruning threshold $t_{\Delta C}$ is high, the mappings that have been pruned promise to contribute a high amount of data and there is a higher number of pruned mappings than in the case of a low $t_{\Delta C}$. If the mappings that have been pruned promise little data, budget is possibly wasted. If Weighted is used without pruning, there are no mappings that have received no budget, because the query plans are fully expanded. In this case, the only options are to refund budget to the next upstream peer or to send queries again with a higher budget, which is discussed below. However, in the former case, the budget may return by recursive refunding to the peer to which the initial query was posed to.

Giving the refunded budget to mappings pruned earlier works well with Greedy, because this algorithm prunes all but the best mapping from a set of alternative results combined in a rewriting subgoal. Since usually there are more than one single mapping, the refunded budget can be allotted to mappings that promise to be attractive. Example 19 (Sec. 8.1) contains such a case at peer P_2.

Refund budget Using the Greedy approach it can happen that a peer gets back budget and at the same time has no promising opportunities to spend this budget, because the cardinality estimations are all poor. In such a situation, the peer should refund the budget to the peer from where it received the query. When all Pdms peers act in a greedy fashion, this budget may be better used by an upstream peer. Usually, there are many mappings not expanded yet on the way back to the peer the query originates from.

Another reason for giving back budget refunded from neighboring peers is that the size of this budget is too small for Weighted to re-send any query. Clearly, to yield more data from a certain mapping path than on a query already sent, the corresponding budget must be higher. If the refunded budget is smaller than the minimal budget spent for any query before, the peer only can refund the budget too.

Budget can be given back for several reasons:

- *Inaccurate Cost Statistics*: Budget is given back when the cost of a query has been overestimated. Then, the peer has the chance to update its statistic that predicts cost of query answering.
- *Changes in the* PDMS: Anywhere in the mapping path that gives back unused budget a change in the PDMS structure can happen. Of course, this can massively influence both cardinality and cost of query answering, Chapter 7. Refunded budget is an another opportunity to adapt statistics to possibly hidden changes in the PDMS. Since we believe that the techniques to exploit the feedback from refunded budget would similar to our proposals in Chapter 7, we do not cover this issue in this thesis.
- *Cached Query Results*: To improve efficiency of query answering, a peer internally can use a query result cache. Then, cost of query answering can be massively smaller compared to recursively retrieving the result from other peers. As a consequence, most of the budget can be given back by that peer.

Re-send queries with higher budget For inaccurate statistics and PDMS changes, it can be promising to adjust the budget assignment of queries and send them again to yield more data. When the budget is saved due to query caches, re-sending a query can lead to a much more expensive and even smaller result, because the peer employing the caches might be forced to obtain the query result in turn from its neighboring peers.

Sending queries again with higher budget is the more promising the larger the unused budget is compared to the cost of the queries that did not refund any budget. A peer can distribute the refunded budget to these query opportunities similarly to the approaches applied in WEIGHTED or GREEDY.

As mentioned above, especially the algorithm WEIGHTED faces to the problem that once budget is refunded from other peers, every opportunity to send queries already has been taken. If the peer wants to increase its query result with budget given back from some of its neighbors, the only way is to re-send queries with a higher budget. GREEDY is only forced to re-send queries if a peer only has a single mapping path to send a query to.

Of course, re-sending queries with higher budget retrieves many results a second time. This is a kind of redundancy. The question is whether to refund budget instead of re-sending any query or whether actually re-sending queries. We believe that a budget-dependent cardinality estimation could help to decide this. But this is beyond the scope of this work.

8.3.2. Deferred Weighted

Whereas refunding budget usually applies to GREEDY, we now introduce a similar approach for WEIGHTED. Deferring pruning decisions in WEIGHTED provides a way to decide about allotting budget based on slightly more global information. Similar to exploiting query feedback for cardinality and overlap estimation, Sec. 5, refunded budget can be exploited to adapt query planning and pruning decisions to better solve the completeness optimization problem as the algorithm WEIGHTED does.

This can be accomplished by deferring the decision about spending budget to the next mapping until it is known how much budget the current mapping will consume and refund. On the one hand, this would take more time, because answering of a query cannot run in parallel. However, this strategy guarantees that the entire budget is used up even for the weighted distribution of budget. Please note that GREEDY is inherently a deferring strategy.

8.4. Exemplary Comparison of Algorithms

Before we examine our algorithms experimentally, this section gives a brief comparison of the algorithms proposed above. To this end, we return to the illustrative Example 19 starting at Page 130.

Example 24. *In Table 8.1, we list all budgeted global query plans for the example query to P_1 ordered by their budget. Each of them is discussed in the following. If budget is not limited, we obtain the maximum completeness of 0.76. This query plan exploits the data at peer P_5 two times, first over the path $P_1 \rightarrow P_2 \rightarrow P_6$ that branches at P_2 and also includes P_4. Second, the data at P_5 is used over the direct mapping $P_1 \rightarrow P_5$.*

The latter is the reason why this complete plan is superior to the plan resulting from Altruistic GREEDY *with a budget of 4, which misses this direct access to P_5 and therefore yields a smaller completeness of 0.62. The data from P_5 can only be transported via the mapping $P_2 \rightarrow P_6$ that acts as a filter due to its selection predicate. With* GREEDY *and a budget of 3 , the plan's completeness of 0.59 is almost as high as with Altruistic* GREEDY *and a budget of 4. This plan does not involve P_4, which only contributes comparably little data in the plans discussed above. The plan that results from applying altruistic* GREEDY *with a budget of 2 achieves a completeness of 0.52 by only accessing P_1, P_2, and P_4. So, interestingly, this plan obtains almost 84 % of the completeness of the plan with the double amount of budget, and 65 % of the maximum completeness available for our example query to P_1.*

WEIGHTED *distributes the full budget of 4 units equally to P_2 and P_5 since only full units of budget can be assigned in the cost model of this example. P_2 in turn assigns its remaining budget of 1 to P_6, where the first of two mapping paths ends. Since P_5 has no outgoing mapping, it cannot use up its budget. 1 unit of budget which remains might be given back to P_1. Performing query answering with* WEIGHTED *and the full budget ends with the same completeness score as* GREEDY *with a lower budget of 3. Including budget refunding would not improve the result of* WEIGHTED *in this example unless deferred* WEIGHTED *is applied. The reason is that P_1 cannot exploit a budget of 1 refunded by P_5, because the other mapping paths has already be queried with a higher budget of 2. This observation leads to the application of deferred* WEIGHTED*. If P_1 can afford, it sequentially sends budget to its outgoing mappings in increasing order of their potential* *data contribution. Then, the budget of 1 unit refunded by P_5 can be packed with the budget of 2 and sent to P_2. P_2 first sends a budget of 1 to P_4. After receiving the query answer it queries P_6 with a budget of 1, which does not contribute further data to the overall result. In summary, deferred* WEIGHTED *includes all peers in query answering and thus yields the full completeness.* □

Approach	Budget [#Mappings]	Global query plan	C
COMPLETE	∞	$P_1.R_1 \sqcup ((P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5)) \sqcup P_4.R_4) \sqcup P_5.R_5$ contributing peers: P_1, P_2, P_4, P_5	0.76
Deferred WEIGHTED	4	$P_1.R_1 \sqcup ((P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5)) \sqcup P_4.R_4) \sqcup P_5.R_5$ contributing peers: P_1, P_2, P_4, P_5	0.76
Altruistic GREEDY	4	$P_1.R_1 \sqcup ((P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5)) \sqcup P_4.R_4)$ contributing peers: P_1, P_2, P_4, P_5	0.62
WEIGHTED	4	$P_1.R_1 \sqcup (P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5))$ contributing peers: P_1, P_2, P_5	0.59
GREEDY	3	$P_1.R_1 \sqcup (P_2.R_2 \sqcap (P_2.R_3 \sqcup P_5.R_5))$ contributing peers: P_1, P_2, P_5	0.59
Altruistic GREEDY	2	$P_1.R_1 \sqcup ((P_2.R_2 \sqcap P_2.R_3) \sqcup P_4.R_4)$ contributing peers: P_1, P_2, P_4	0.52
GREEDY	2	$P_1.R_1 \sqcup (P_2.R_2 \sqcap P_2.R_3)$ contributing peers: P_1, P_2	0.49

Table 8.1.: List of all possible budgeted *global* query plans and their completeness C by budget for the query $q(a,b,c,d,e) :- P_1.R_1(a,b,c,d,e)$ in the example in Fig. 8.1. Some of the brackets solely document how distributed query processing proceeded to compute the result of the plan.

8.5. Experimental Evaluation

In this section we examine experimentally how well our budget-driven strategies actually can reduce query execution cost in terms of response time considerably and follow a cost constraint at the same time.

We conduct similar experiments as in Sec. 6.3. First, a set of random training queries (usually 25) is issued to the PDMS to calibrate the cardinality histograms. The rest of the random query workload serves to measure the cost reduction and completeness yield compared to retrieving all certain answers. Based on these results, we can determine a measure for the efficiency gain for each query. Throughout our experiments, we use the query response time as cost measure. All the experiments in this section were conducted with our testbed *Humboldt Peers*. The main properties of the PDMS instances are listed in Table 8.2.

Experiment 8.1 (Efficiency of budget-driven approaches.) This first experiment is intended to give an impression of the efficiency of our budget-driven strategies for query optimization.

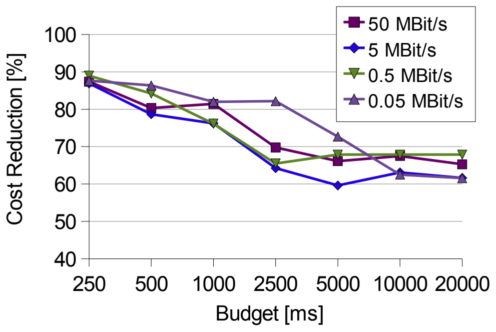
	#Peers	Rank	Mappings with projections	Average mapping path length	Data distribution
$\mathcal{P}_6$	40	4	58%	7	random
$\mathcal{P}_7$	100	4	43%	8	random
$\mathcal{P}_8$	40	4	58%	7	Zipfian ($z = 0.5$)
$\mathcal{P}_9$	100	2.4	42%	2.3	random

Table 8.2.: Datasets and their main characteristics (continued from Table 7.2 from Page 116).

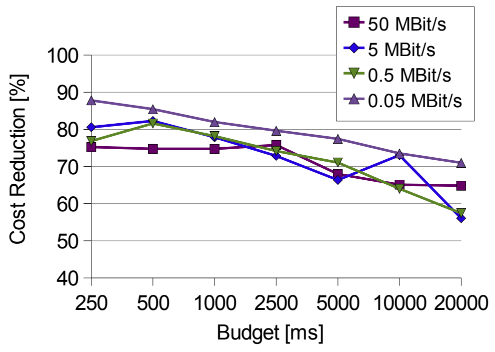
Methodology. The algorithms Greedy and Weighted are applied to some of the randomly created Pdms instances of different size described in Table 8.2. On the one hand, we vary the time budget available for query processing. On the other hand, we study the effect of our approaches for a wide spectrum of network bandwidths. To get an impression of the influence of the size of the Pdms we compare $\mathcal{P}_6$ with 40 peers and $\mathcal{P}_7$ comprising 100 peers. The query workload comprises 100 random queries. 25 of them are training queries.
Discussion. The results of the experiments with $\mathcal{P}_6$ are depicted in Fig. 8.3. With the exception of the smallest network bandwidth of 0,05 MBit/s, all completeness scores are above 0.7. The cost reduction for Weighted is slightly lower than for Greedy, but this is overcompensated by a considerably higher completeness of about 0.9 and more for sub-second budgets in case of Weighted. The completeness results converge to 1 as the budget for the queries increases. But even for a time budget of 20 seconds the completeness clearly remains below 1.

The efficiency gains are even more impressive for the 100 peer Pdms instance $\mathcal{P}_7$ as depicted in Fig. 8.4. For small budgets of 250 or 1000 ms, Weighted yields an efficiency increase between one and two orders of magnitude even for the fastest network bandwidth considered in our experiments (50 MBit/s). Greedy performs even better with efficiency gains clearly beyond an order of magnitude for all budgets and all network bandwidths examined.

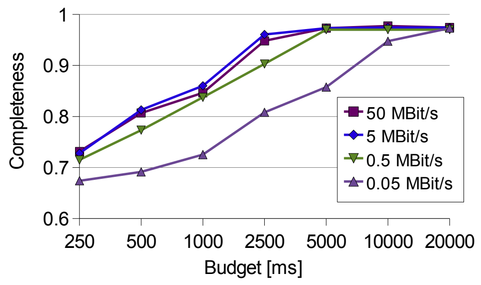
Taken together, the efficiency gains achieved by the budget-based approaches are about an order of magnitude or higher if the budget is limited to be below one second in our system environment. It is underlined that this result is achieved with a level of completeness that can be regarded as satisfying for many scenarios of large-scale information sharing.
Experiment 8.2 (Bounded response time.) This experiment examines whether the budget-driven approaches actually follow the budget constraint. This is difficult to achieve, because a cost model that a peer uses to predict the query response time of its neighboring peers is always inaccurate. To fulfill a cost constraint in the case that local or remote query processing has taken too long, a peer in *Humboldt Peers* can reduce the size of the result that is transferred back to the calling peer and thus save transport time.
Methodology. We display the query answering time for every query for several exper-



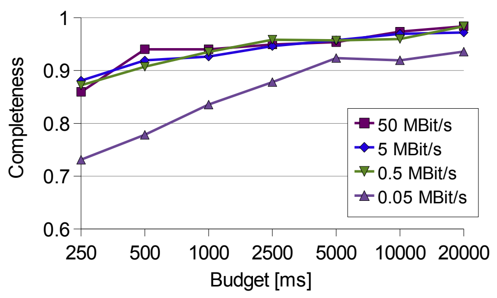
(a) Cost reduction for GREEDY.



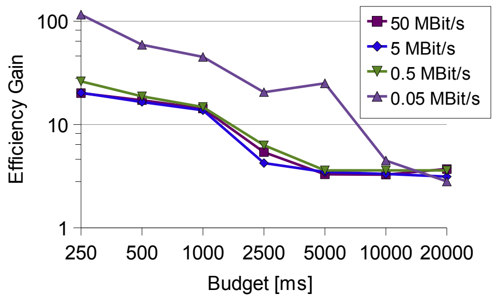
(b) Cost reduction for WEIGHTED.



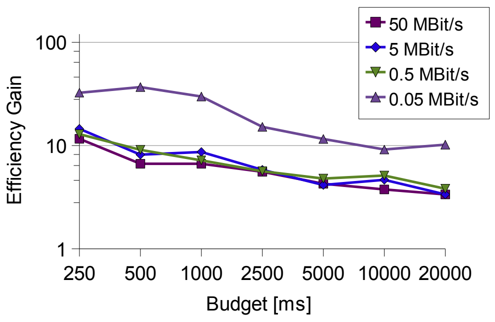
(c) Completeness for GREEDY.



(d) Completeness for WEIGHTED.



(e) Efficiency gain for GREEDY.

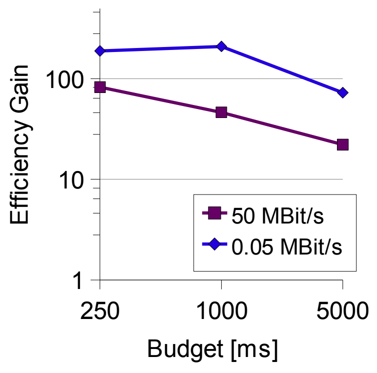


(f) Efficiency gain for WEIGHTED.

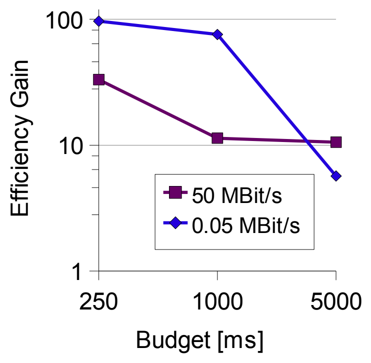
Figure 8.3.: Cost reduction, completeness scores, and efficiency gains for different budgets and network bandwidths for PDMS $\mathcal{P}_6$.

iments and check whether they follow the time budget constraint. These experiments vary in the size of the time budget and the cost constraint. Again the query workload is made up of 100 random queries, of which 25 are considered as training phase for the histograms.

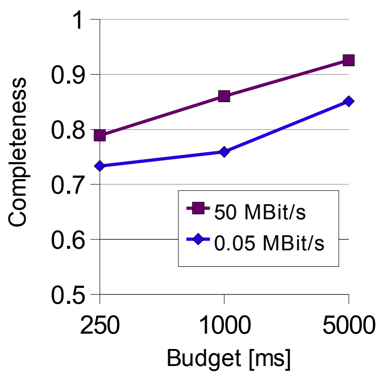
Discussion. Some typical results of query response times are depicted in Fig. 8.5. The results for other budgets and network bandwidths are similar. With exception of some outliers in other experiments, the query response time clearly follows the budget. Most



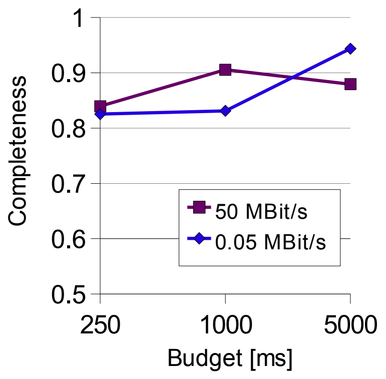
(a) Efficiency gain for GREEDY.



(b) Cost reduction for WEIGHTED.



(c) Completeness for GREEDY.



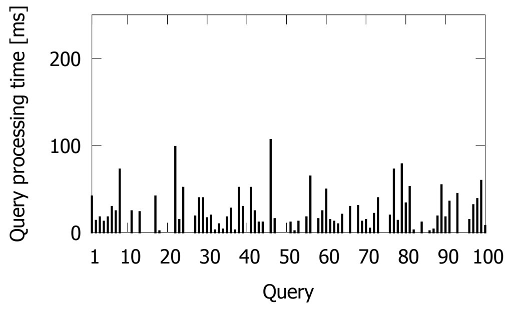
(d) Completeness for WEIGHTED.

Figure 8.4.: Efficiency gains and completeness scores for different budgets and network bandwidths for PDMS $\mathcal{P}_7$ with 100 peers.

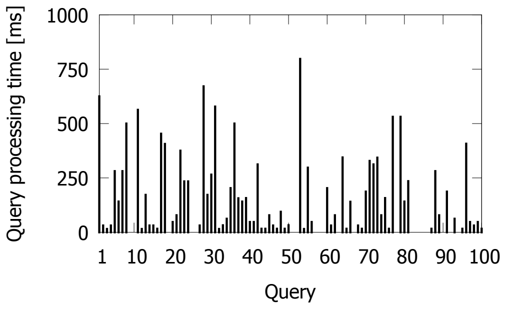
response times are considerably below the allowed response time. This shows that query answering could achieve even better results, if the cardinality and cost estimation is more accurate than for the techniques implemented in *Humboldt Peers*. As a comparison between Fig. 8.5(a) and Fig. 8.5(b) reveals, the average level of the query answering time increases with the budget.

Experiment 8.3 (Data distribution over the peer graph.) Our approaches to explore the PDMS mainly differ in how they explore the peer graph. So one might expect that their effectivity depends on the distribution of data over the network of peers. The more an overall data set it is spread over the system, the higher the effort is supposed to be to find all certain answers since more queries must be generated, more data transport events happen involving latency time each time, and the length of the data transports increases.

Methodology. We use a skewed data distribution of data over the peers. Analogously to the skew of the reference data in the data space in Sec. 7.2.4, we construct a Zipfian



(a) Query response time for budget of 250 ms, network bandwidth of 0.05 MBit/s, and Greedy.



(b) Query response time for budget of 1000 ms, network bandwidth of 50 MBit/s, and Weighted.

Figure 8.5.: Query answering times for Pdms $\mathcal{P}_8$.

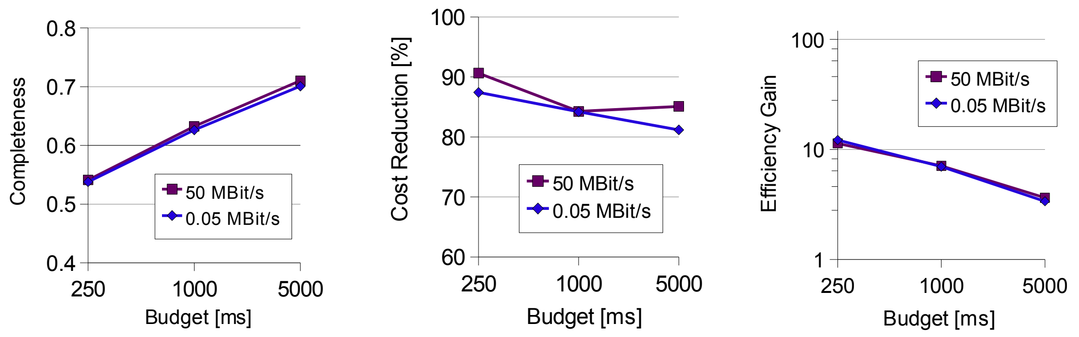
distribution [Zip49] that is controlled by the Zipf parameter z. We experimented with a value of $z = 0.5$ that distributes 1000 tuples in following packages: 88, 62, 51, 44, 39, 37, 33, 29, 28, 26, 25, 24, 23, 22, 22, 21, ... These data packages were assigned to the peers randomly. The workload consists of 100 random queries.

Discussion. With exception of queries with small response time both Greedy are Weighted interestingly prove to be similarly efficient and show cost reductions of more than 70% with completeness scores of more than 50% even for a small budget of 250 ms, Fig. 8.6. Looking into the details, Greedy proves considerably higher efficiency improvements for small budgets of 250 and 1000 ms and for high network bandwidth of 50 MBit/s. This advantage vanishes for slower networks and higher budgets. Further results reveal that only for queries with small answering time Weighted takes longer than Greedy. The main insight that can be drawn from this experiment is that both Greedy are Weighted prove to be robust against skewed data distribution across the Pdms.

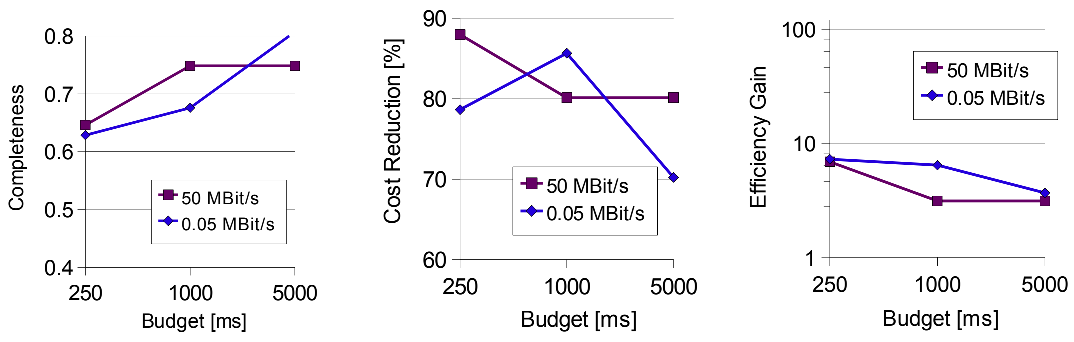
Experiment 8.4 (Interconnectedness of the peer graph). With this experiment, we examine how our approaches behave in Pdms instances with different rank, which is defined as the average number of number of peer mappings of a peer. In less interconnected Pdms instances with a smaller rank, the average length of peer mapping paths is usually also smaller as can be seen in Table 8.2, for instance. This is due to the fact that the more neighbors peer have the higher the more outgoing mappings they have.

Methodology. We derived Pdms instance $\mathcal{P}_9$ from $\mathcal{P}_7$ by removing about every second mapping, Fig 8.7. This was realized by uniformly and randomly removing every second mapping at each peer. The amount of data and the data distribution over the Pdms remain unchanged. This leads to a rank of 2.4 compared to the rank of 4 for $\mathcal{P}_7$. The average mapping path length changes from 8 to 2.3. Then we used the same query workload as in Experiment 8.1 and compared the results with the ones from $\mathcal{P}_7$. The query workload is made up of 100 random queries.

Discussion. The efficiency gains for the experiment with Pdms instance $\mathcal{P}_9$ are depicted in Fig. 8.8. The main conclusion is that Greedy and Weighted change their behavior



(a) Completeness in case of Greedy. (b) Cost reduction in case of Greedy. (c) Cost reduction in case of Greedy.



(d) Completeness in case of Weighted. (e) Cost reduction in case of Weighted. (f) Cost reduction in case of Weighted.

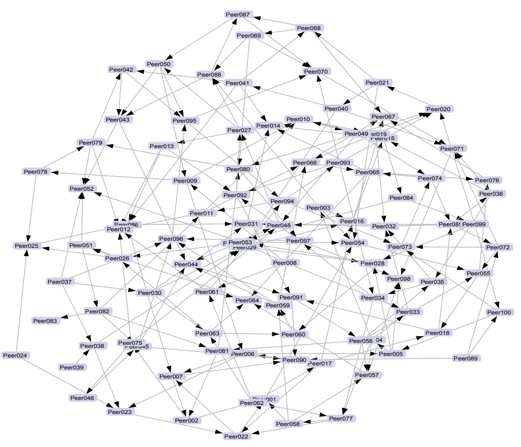
Figure 8.6.: Cost reduction, completeness scores, and efficiency gains for different budgets and network bandwidths for Pdms $\mathcal{P}_8$.

similarly when comparing these Pdms instances with different degree of interconnection. Both show less cost reduction that similarly drops with increasing budget as it is the case for the more interconnected Pdms instances.

An obvious difference is that Greedy shows a considerably higher increase in completeness than Weighted does. The reason for that is that Weighted yields a higher level of completeness than Greedy in the higher interconnected case.

Another interesting insight is that the efficiency gain vanishes for $\mathcal{P}_9$ if the budget increases to a certain range. This highlights the necessity to find out the right level of budget for a given Pdms instance before any productive use. Of course, it would be even better to vary the budget in dependency of the query at hand. This could be based on corresponding statistics, but is out of the scope of this work.

In summary, our experiments clearly show the feasibility of our strategies for query planning under a limited budget. Especially for large Pdms they achieve more than 75% completeness but with drastically reduced cost while guaranteeing to stay below a given cost limit.


(a) PDMS instance $\mathcal{P}_7$ (rank 4).

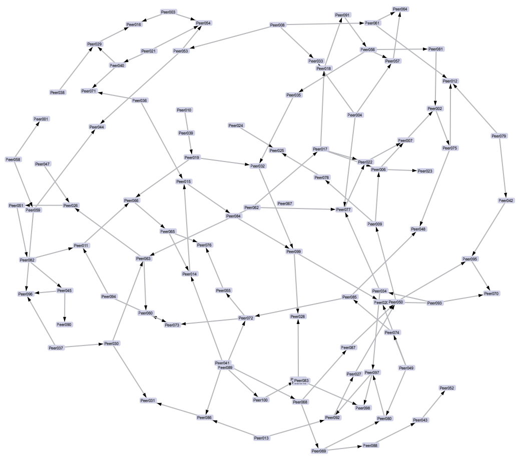
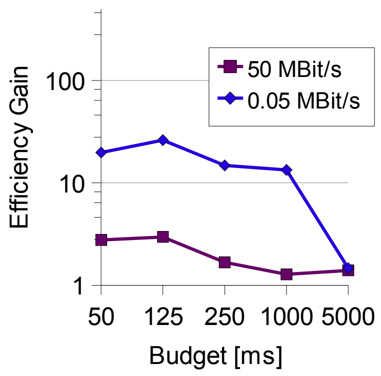
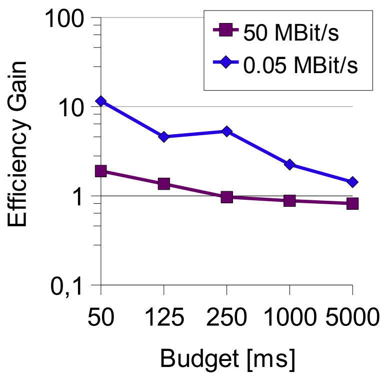

(b) PDMS instance $\mathcal{P}_9$ (rank 2.4).

Figure 8.7.: PDMS instances $\mathcal{P}_7$ and $\mathcal{P}_9$ with the same number of peers but different rank.



(a) Efficiency gain for PDMS instance $\mathcal{P}_9$ and GREEDY.



(b) Efficiency gain for PDMS instance $\mathcal{P}_9$ and WEIGHTED.

Figure 8.8.: Efficiency gains for PDMS $\mathcal{P}_9$.

8.6. Related Work

Both Kossmann [Kos00] and the survey article [OB04] of Ouzzani and Bouguettaya emphasize volatility and autonomy as important characteristics of distributed information sources in today's world. As we agree, we pay special attention on these two requirements for integrated information systems in this work. In the following, we first review related work on optimization for individual data integration systems and second in a network of them, i.e., a PDMS.

8.6.1. Query Optimization in Integrated Information Systems

There are many works dealing with query optimization in data integration in general, e.g., [EKMR06], [FKL97], or [HKWY97]. Here, we focus on some important approaches that are very similar to our best-effort strategy.

Among other topics, the work in [Nau02] deals with completeness-driven query optimization in mediator-based integrated information systems. Since our peers act as mediators as well, we can draw much inspiration from these results. For instance, we apply the completeness model introduced by Naumann in this work, Chapter. 4.

In [Nau02], the so-called translated query contains relations from the global schema of the mediator and the merge operators introduced in that work and thus corresponds to queries against peer schemas in this thesis. The revised query planning paradigm in Sec. 6 of [Nau02] chooses a *set* of sources for each relation in a translated query. Note that in that approach each relation is treated in isolation. Instead, we apply answering queries using views [Hal01] and global-as-view expansions *in parallel*. Both techniques lead to rewritings that can include joins between relations in remote peer schemas. In that context, some results from other peers only can be used *together* in a query plan, because of their special foreign key relationships.

The Greedy Look-Ahead algorithm Naumann proposes maximizes the completeness of the whole plan by selecting the set of sources that maximally increases the completeness of the plan. Again, due to the restrictions for the query planning approach in that work, this algorithm cannot be directly applied to our setting with complex query plans resulting from exploiting both local-as-view and global-as-view mappings at the same time. However, we apply a similar strategy when selecting a subplan from the full query plan in our Greedy algorithm for budget spending, Sec. 8.2. The main difference is that we have to distribute the budget over several subgoals of a query rewriting. Additionally, our algorithm employs pre-sorting of the data sources available for each subgoal and then takes mapping sets out of these collections. So in effect, our algorithm does not perform an exhaustive search on the local level of a peer.

The budgeted maximum coverage problem introduced in [KMN99] is similar to the budgeted maximum completeness problem covered in this section. The difference is that in our context the principle of optimality does not hold for the objective function of plan completeness. The work in [KMN99] deals with approximation to solutions to the budgeted maximum coverage problem. In contrast, we argue that since we usually face to a small number of neighboring peers we are able to enumerate many possible plans and then decide which can be executed given a budget.

Duschka assumes in [Dus97] that it is known for the underlying information sources that subsets of the data at the sources are complete wrt. the whole set of sources a data integration systems accesses. This property is called local completeness. That work shows how to exploit such knowledge in query optimization. Our cardinality histograms can be seen as a much weaker form of local completeness: they specify query-dependent data contributions of information sources. It remains an interesting research question whether some of the query optimization techniques can be carried over from [Dus97] and be applied to our context and vice-versa.

8.6.2. Query Optimization in PDMS

The work of Tatarinov and Halevy [TH04] focuses on information preserving pruning, Sec. 6. It is realized by query containment checks during the creation of the *global* rule-goal tree. In this context, the authors investigate the effectivity and efficiency of breadth-first and depth-first search. They find that a depth-first strategy is much worse than breadth-first in their context. This is due to the non-monotonic nature of their query answering technique. Rewriting nodes far down in the rule-goal tree are likely to be pruned by nodes closer to the root of the tree that are possibly created at a later point in time. In such cases, reformulation results already achieved will be substituted by equivalent rewritings. Hence, reformulation time is wasted.

In contrast, in our work an expansion of a rule node will never be revised. Thus, the negative effects with the depth-first strategy in [TH04] cannot happen in our mechanism. Instead, our approach contains a weaker form of non-monotonicity: Once budget is refunded, a peer can decide to revise pruning decisions and instead give budget to mappings that were pruned earlier on. The difference to [TH04] is that in our technique no earlier work or result structure has to be discarded.

8.7. Summary

The contribution of this chapter is to use a budget to control distributed query optimization in a Pdms. This is a simple, yet effective means to bound resource consumption in a cooperation of autonomous peers. By this approach, the main disadvantage of completeness-driven query planning from Chapter 6, namely unpredictable cost of query answering, is overcome.

First, we proposed two algorithms for spending a limited budget to optimize completeness of the query result. They both distribute the budget by considering potential data contributions of the mapping paths occuring in the local query plan of a peer. Weighted explores the search space in a breadth-first manner, while Greedy follows the best out of a set of alternative opportunities for passing a reformulated query.

Then, the algorithms were extended by a mechanism to refund unused budget to the calling peer. We discussed several possible usages for such budget in the context of our algorithms. Most interestingly, we proposed Deferred Weighted, a variation that deferred spending budget until refunding from the last mapping path has happened.

We conducted an extensive experimental evaluation. It proved that both Weighted and Greedy improve efficiency of query answering by several factors up to two orders of magnitude while guaranteeing a limited resource consumption.

9. Query Optimization using Information Overlap

Overlap between data from alternative mapping paths massively influences the cost of query answering in a PDMS. Due to the nature of such systems, many tuples are unnecessarily transported over redundant mapping paths. This leads to massive data overlap perceived at the peers.

Having focussed on maximizing completeness in Chapters 6 and 8, we now aim to directly *decrease* cost by addressing data overlap. At the same time, completeness of a given query plan must be preserved as far as possible. The techniques developed in this chapter can be orthogonally applied together with threshold-based pruning or budget-driven query optimization. Reducing overlap of the subresults is accomplished by identifying sub-volumes with high overlap in the multi-dimensional data space and rewriting outgoing queries such that data overlap is decreased.

We describe this new technique in the following sections. First, an approach for identifying sub-volumes suffering from high overlap in the multi-dimensional data space is presented. Next, we analyze this optimization problem in more detail by taking a look on how different kinds of cost depend on the procedure of query answering. Then our novel solution for *partially* pruning outgoing queries is introduced. Finally we report on an extensive experimental study in a master's thesis that refined and implemented our ideas [Tie09].

9.1. Reducing Overlap Cost and Preserving Completeness

In general, a local query plan at a peer comprises an exponential number of alternative rewritings and one level below an exponential number of sets of mappings making up the rewritings. This exponential number of possible query plans has often been mentioned in the literature, e.g., [NKN05, VP98, FKL97, Nau02], but has not been handled yet with satisfying efficiency.

Each pair of alternatives can show *different* data overlap and completeness characteristics. The distribution of overlapping data and non-overlapping data over the data space can vary between different pairs of alternative mapping sets, for instance. So an approach to reduce data overlap has to consider each pair of alternatives in isolation. Of course, PDMS do not only mean much more redundancy than a centralized mediator, rather this huge amount work for overlap-driven query optimization is also shared among the possibly large set of peers.

Since each peer has only a small number of neighboring peers and therefore only a few outgoing mappings, we can afford a detailed analysis of data overlap and exploit both overlap and cardinality estimations for optimizing the local query plan at a peer. To the best of our knowledge, this is the first approach that considers the distribution of data overlap over the data space and uses such information to access only sub-portions of data available over a certain set of mappings. We call this technique *partial pruning* of mapping paths. Before diving into the details, we formulate the problem that is solved by this novel approach.

9.1.1. Combining Optimization Goals

Given a local query plan we aim to find a transformed query plan with minimal cost whose completeness exceeds the bound:

Definition 27 (Local Cost Minimization Problem)**.** *Given a local query plan P with estimates for the result cardinality and cost for each mapping set in P, and further given an estimation for the distribution of the extensional overlap between each pair of mapping sets in P as well as a completeness threshold t_C, find a transformed plan P' with minimal cost that has an estimated completeness $C(P') > t_C \cdot C(P)$ and $P' \subseteq P$.*

We combine the objectives of decreasing cost by avoiding overlap and high completeness by using a boundary condition for the plan completeness. It requires the completeness of the transformed plan to be at least a certain fraction of the completeness of the fully-expanded local query plan.

9.1.2. Approximative Solution

Before presenting our novel approach in detail we return to the illustrative example introduced in Sec. 4.4 on Page 59.

Example 25. *There we examined the result sets that a peer P_1 receives from its neighbors P_2, P_3, and P_4 for overlapping tuples. The local query plan of P_1 depicted in Fig. 4.3(b) (on Page 60) reveals that the tuple $\{R(r_1) \sqcap S(s_1)\}$ occurs in the three alternative rewritings $P_3.RS$, $P_2.R \sqcap P_4.S$, and $P_4.R \sqcap P_4.S$. The mapping from P_1 to $P_3.RS$ exclusively returns the tuple $\{R(r_2) \sqcap S(s_2)\}$. So pruning the mapping to $P_3.RS$* completely *would cut off this result.*

If the peer P_1 had known about this by exploiting an overlap histogram for the mapping $P_1 \rightarrow P_3$, it would have been able to manipulate the query sent to P_3 such that the overlapping volume around the tuple $\{R(r_1) \sqcap S(s_1)\}$ is not queried there. However, the complementing tuple $\{R(r_2) \sqcap S(s_2)\}$ would still be returned by P_3. □

In the following we propose an approximation to the local cost minimization problem in the sense that we usually find a good but not the optimal solution. We experimentally compare our approach with the best transformed plan by enumerating all possible transformed plans.

Intuitively, our main idea is to use the information about overlap and data distribution to manipulate the resulting queries of a local plan such that the potential overlap of the corresponding subresults is decreased as much as possible. Our novel procedure consists of the following phases:

1. Identification of sub-volumes of the multi-dimensional data space with high overlap using our overlap histograms introduced in Sec. 5.3.
2. Assignment of these overlapping sub-volumes to the alternative mapping sets.
3. Decomposition of the complement of the overlap sub-volumes assigned to each mapping set into rectangular volumes.
4. Transformation of the given local query plan by replacing the queries to be sent to neighboring peers by selections representing the complement sub-volumes.

The effectiveness of the first step depends on the accuracy of the overlap histograms as well as on the distribution of overlapping and non-overlapping data over the data space. If a sub-volume of the multi-dimensional data space comprises both much overlap and many non-overlapping tuples, eliminating this sub-volume from further query processing would lose also all of these non-overlapping data from being retrieved over the pair of mapping sets currently treated. We decided *not* to involve such cardinality consideration at this point, because when checking a pair of alternative mapping sets, there can be *other* mapping sets that probably can return the non-overlapping tuples mentioned above. Rather, we take into account the completeness of the whole transformed query plan when distributing the overlapping sub-volumes to the mapping sets in Step 2.

The distribution of the overlap volumes to be excluded to the mapping sets in Step 2 is the core of the approach. At this point, our optimization goals control the decision to which one of two overlapping mapping sets a particular sub-volume is assigned. We choose the alternative with the lowest cost that still fulfills our completeness constraint.

Once all overlap sub-volumes are distributed to the mapping sets, they have to be excluded from the query plan. To this end, we compute the complement of overlap volumes for the resulting query for each mapping set. To prepare the next step, this complementing volumes have to be fully subdivided into multi-dimensional rectangular volumes.

In Step 4, these resulting pieces of the multi-dimensional data space actually to be queried can be translated into selection predicates. Taken together they replace the original results of the mapping sets below the rewriting subgoals C_i. In summary, we substitute a single sub-query involving a particular mapping set by many new sub-queries for that mapping set. The results of these sub-queries are combined by the union-like full outer-join operator when forming the result for a rewriting subgoal.

9.1.3. Trading-Off Costs

These new selection sub-queries mean additional cost for query planning and sending them recursively through the network of peers. The additional cost have to be traded-off with the cost saved by excluding overlapping data from being transferred.

Next, we describe the ingredients of a cost model. As stated in [Nau02], we can distinguish sequential and parallel cost models. In the former, the total cost is the sum of the cost for the individual participants. This is the case if only the monetary price is considered as cost dimension. In parallel cost models the total cost is equal to the cost of the most expensive participant. For instance, this applies if a query answer has to be joined from the results returned by several sources, that can be queried in parallel and where response time is the only cost criterion. Since a sequential cost model is easier to study, our testbed *Humboldt Peers* sends queries to neighboring peers sequentially. Furthermore, we process only a single test query at a time in the whole PDMS instance.

This makes no fundamental difference to a parallel mode with a parallel cost model, which *Humboldt Peers* is also capable of. There, the cost for a single test query processed at a time would be much smaller due to parallel processing on the one hand. But on the other hand, if several test queries were processed at the same time, the workload on a particular peer would be higher and thus the execution time for each test query would not be significantly smaller than in the sequential case. The effectiveness of our query optimization approaches is similar in the parallel query processing case.

In [OV11] Özsu and Valduriez present a detailed cost model for distributed databases. It distinguishes between local computing efforts (for CPU and I/O) and network cost. We simplify this model for our purposes and investigate the main influence factors in our setting: Both computing effort and data transport cost is influenced by the size $|R|$ of the query answer. The second important factor is the number and size of queries Q_i sent to neighboring peers. It determines the effort for query planning at all subsequent peers. Moreover, each query requires latency time for setting up a network connection to send the message containing the query and the result on the way back.

As introduced in Sec. 4.6, we model the cost $Cost(P)$ of a *local* query plan P by

$$Cost(P) = v \cdot |R| + c \cdot \sum_i |Q_i|.$$

The specific cost factors v and c heavily depend on the topology of the PDMS. Since they are unknown locally at a peer, these parameters must be determined from query feedback. Gruser et al. [GRZZ00] propose a technique to learn response time for Web data sources from exploiting previous query answers. The work in [NGT98] also mentions the use of query feedback for establishing cost models for mediator-based information systems. Since in PDMS data are usually transported over several peer mappings, we can assume that cost for data transport dominate local computing cost.

Reducing overlap with partial pruning comes at the expense of more queries being sent through the network of peers. Hence, this means a trade-off between the part of the cost influenced by the number of tuples transferred and the number of queries being issued. One of the experiments in [Tie09] aims to find the break-even for this trade-off. To this end, the factors v and c are varied and virtual cost for actual result sizes $|R|$ and the number of foreign peer queries Q_i are calculated based on values measured in the experiments.

At query planning time, a peer can estimate the cost for its local query plan based on the estimates for the cardinalities of the subresults returned by the mapping sets. Additionally, the number and size of queries to be posed to neighboring peers is calculated from the local query plan considering possible join pushdowns. With the global virtual query plan becoming deeper, the selections introduced by partial pruning make the queries being propagated more and more restricted. Hence, some of the mappings that would have been used before will not be expanded because of conflicting comparison predicates between intermediate queries and peer mappings.

9.2. Identifying Data Overlap

To describe overlap between mappings, we search for maximum sub-volumes of the multi-dimensional data space showing high overlap. This is captured by the overlap histograms presented in Sec. 5.3. As a measure for comparison of data overlap between these sub-volumes, we define overlap density as follows. Of course, the volume to be examined for overlap is bounded by the selection predicates of the query received by the peer under consideration.

Definition 28 (Overlap density)**.** *Consider a sub-volume S of the multi-dimensional data space covered by a self-tuning overlap histogram. Let the multi-dimensional volume of S be v_S and the sum of the overlap of all individual buckets in S be o_S. Then the overlap density d_S is defined as $d_S = o_S/v_S$.*

In the experiments in [Tie09], a threshold is used for the overlap density to identify updates for our overlap histograms, i.e., new sub-buckets to be drilled into existing ones.

To exploit the full potential of a set of mappings, overlap reasoning must be integrated with estimates about cardinality. The reason is that a mapping can have a high overlap with others on the one hand. But this disadvantage can be overcompensated by the data contribution, which is part of the complement of the volumes showing high overlap.

Recall that data contributions have two dimensions, Fig. 9.1. Extensional overlap talks about counting duplicate tuples. But the size of a data set is also dependent on the number of attributes, i.e., the intentional dimension. Hence, a mapping can compensate high extensional overlap by its intentional contribution, i.e., attributes not provided by alternative mappings. Observe that the intentional contribution of an additional attribute is dependent on the *overall* number of tuples in the relation if `null`-values are not considered. In contrast, the "disadvantage" of overlap is dependent on the size of the overlap, which usually is only a fraction of the overall number of tuples available from a mapping. These issues are considered in our algorithms for distributing the overlapping volumes to the alternative mapping sets.

The impact of important properties of PDMS on characteristics of the overlap distribution and potential cost savings is qualitatively summarized in Table 9.1. Therein, "distinguishability in data space" is an informal measure how well volumes with an overlap density above a certain threshold can be distinguished from the rest of the data space. "Local source overlap" refers to the overlap of the data stored locally at the peers in the PDMS.

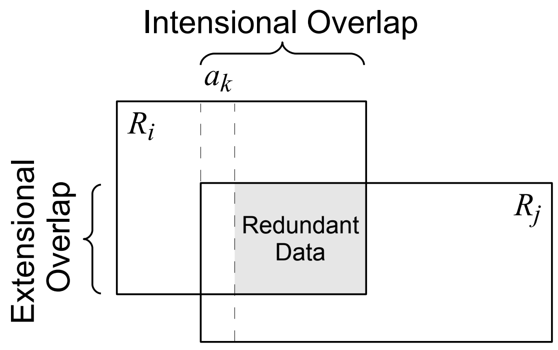


Figure 9.1.: Extensional and intensional dimension of data overlap. a_k denotes the key/foreign key of the relations R_i and R_j. The grey shaded area marks the potential savings.

9.3. Partial Pruning of Mappings

Having identified overlapping sub-volumes in the data space, we proceed by trying to exclude most of the overlapping volumes by adding appropriate selections to alternative mapping sets such that the overlapping volumes are queried only once. This technique has the important advantage that the autonomy of peers is fully preserved, because overlap is solely reduced by changing the queries being sent to neighboring peers. Since only a subset of the data potentially being returned is pruned, we call this approach *partial pruning.* Actually, by this technique, only parts of the data space with high overlap are pruned.

To avoid overlapping results, we transform a selection query into an equivalent conjunctive selection:

Definition 29 (Decomposition of a selection query)**.** *Let Q be the conjunctive selection query. A decomposition of Q is a set $\mathcal{D}$ of conjuncts each of which represents a rectangular sub-volume D_i of the multi-dimensional data space such that*

$$\bigcup_{D_i \in \mathcal{D}} D_i = \mathcal{D}.$$

The main idea of our approach is to manipulate a decomposition of the selections of a mapping set within a query plan in such way that certain sub-volumes of multi-dimensional data space are no longer queries over that mapping set:

Definition 30 (Overlap exclusion)**.** *Let $\mathcal{D}$ be a decomposition. Let $\mathcal{O} = \{O_1, O_2, \dots, O_k\}$ be a set of overlap volumes to be excluded from $\mathcal{D}$. An overlap exclusion $\mathcal{D}' = \mathcal{D} \setminus \mathcal{O}$ is computed by excluding each $O_i \in \mathcal{O}$ from $\mathcal{D}$.*

The result of an overlap exclusion for a mapping set can finally be used to transform the query plan accordingly. During the whole process, we have several degrees of freedom. The most important one is the choice at which mapping sets of a set of alternative

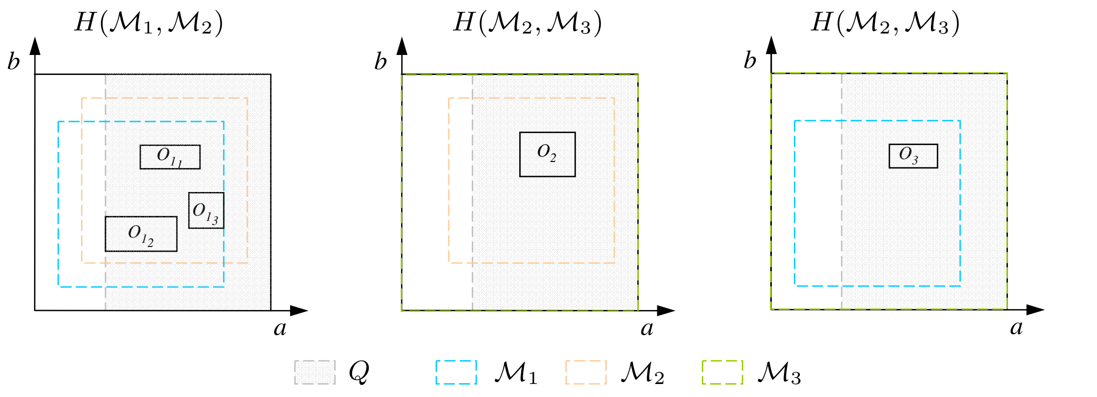
		Overlap size	Extension in data space	Mixture with non-overlapping data	Distinguishability in data space	Potentl. cost savings
Number of peers		+	++	+	−	+
Rank		++	+	++	−	++
Selectivity of mappings		−	−	0	0	−
Projections in mappings		0	0	0	0	0
Length of mapping paths		+	+	+	−	++
Local source overlap		++	+	0	0	++
Direction of mappings	uniform	+	+	+	−	+
	different	−	−	−	+	−

Table 9.1.: Characteristics of Pdms and their influence on locally observable overlap and potential cost savings (++ major impact, + impact, 0 neutral, − negative impact) [Tie09].

mapping sets a certain overlap volume shall be excluded. Another decision concerns the order in which pairs of a set of mapping sets are processed to decrease overlap. The work in [Tie09] found that the order of considering these pairs affects the overall result of a query plan, especially if the overlap histograms are mutually inconsistent. This is usually the case in practice, because overlap histograms are inaccurate in general. We choose the mapping sets to exclude a particular overlap volume such that the completeness of the local query plan maximized.

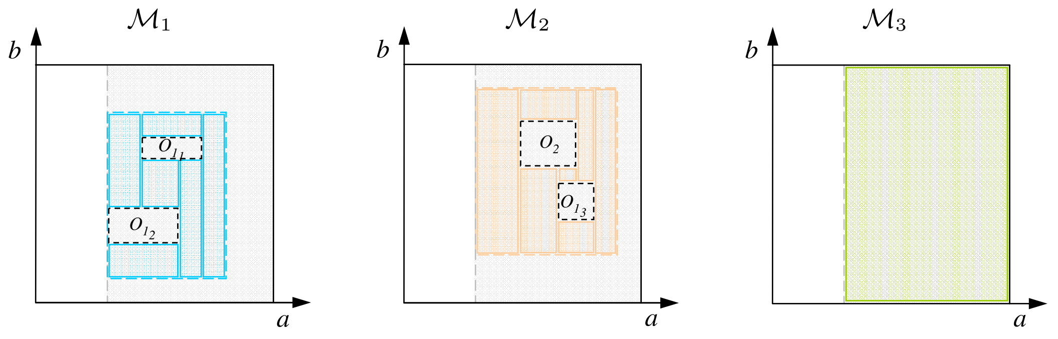
Example 26. *For a simple setting with 3 mapping sets, we show corresponding decompositions in which all of the given 5 overlapping volumes have been excluded. Regard the overlap histograms in Fig. 9.2(a) as a starting point for excluding all the overlap buckets contained in them from the mapping sets $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$. This figure also shows the selection query associated with the mapping sets and the query to be answered. The final state of the decompositions for each of the mapping sets is depicted in Fig. 9.2(b). The overlap volume O_3 need not be excluded at any mapping set since it is contained in O_{1_1}, which is excluded from mapping set $\mathcal{M}_1$.* □

The algorithm OverlapGreedy developed by Tietz in [Tie09] performs a transformation of a given local query plan according to given overlap histograms. It is documented in an adapted form in Algorithm 9.3.1. For every rewriting subgoal of a given



$H(\mathcal{M}_1, \mathcal{M}_2)$
$H(\mathcal{M}_2, \mathcal{M}_3)$
$H(\mathcal{M}_2, \mathcal{M}_3)$
b
a
o_{1_1}
o_{1_2}
o_{1_3}
o_2
o_3
Q
$\mathcal{M}_1$
$\mathcal{M}_2$
$\mathcal{M}_3$

(a) Overlap histograms with selections for the query and mapping sets for mapping sets $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$.



(b) Final state of decompositions for mapping sets $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$.

Figure 9.2.: Overlap histograms and final state of decompositions after excluding all overlapping volumes for mapping sets $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$.

fully-expanded local query plan we consider every pair of mapping sets. First, a decomposition for each mapping set is initialized with the intersection of the selections in this mapping set and the query (Lines 4 - 7). Then, the set of overlapping volumes is determined using the overlap histogram $H_{\mathcal{M}_1,\mathcal{M}_2}$ for the pair of mapping sets at hand (Lines 9 - 10). The main loop is performed over the set $\mathcal{O}$ of overlapping volumes for this pair (Lines 11 - 20), where it is decided to which mapping set to assign each overlap volume. To this end, two candidate transformed query plans are derived from the current one (Lines 13 - 15) and compared for their estimated cost and whether they fulfill the constraint of minimum completeness (Line 17 - 19). Finally, OverlapGreedy returns the transformed query plan P' resulting from replacing resulting queries of mapping sets below the rewriting subgoals by conjunctive selections that represent a decomposition that exclude some overlapping volumes.

We highlight that to valuate the data contribution of a mapping set, we estimate the completeness of the complete local query plan resulting from excluding a certain overlap volume at that mapping set. This is the same approach as in our other optimization and query planning algorithms. It is due to the fact that the principle of optimality does not hold for the completeness of query plans in our context. To trade-off between overlap and completeness on a more fine-grained level, overlap and completeness histograms could be compared in detail to better choose the overlapping areas. But this refinement is beyond the scope of this thesis.

```
Input  : Fully-expanded local query plan $P$ for a query $Q$, threshold for the
         overlap density $t_O$, threshold for the completeness bound $t_C$
Output: Transformed query plan $P'$
1  $P' \leftarrow P$
2  foreach rewriting $Q'_i \in P$ do
3      foreach rewriting subgoal $C_{ij} \in Q'_i$ do
4          $\mathcal{A} \leftarrow \{\mathcal{M} \mid \mathcal{M} \in C_{ij}\}$          $\{\mathcal{M}$: Mapping set below $C_{ij}\}$
5          foreach $\mathcal{M} \in \mathcal{A}$ do
6              $\mathcal{D}_{\mathcal{M}} \leftarrow Sel(\mathcal{M} \cap Q)$          $\{D_{\mathcal{M}}$: (decomposed) volume for $\mathcal{M}\}$
7          end
8          foreach $(\mathcal{M}_1, \mathcal{M}_2) \in \mathcal{A}$, $\mathcal{M}_1 \neq \mathcal{M}_2$ do
9              $\mathcal{D}_O \leftarrow \mathcal{D}_{\mathcal{M}_1} \cap \mathcal{D}_{\mathcal{M}_2}$
10             $\mathcal{O} \leftarrow findOverlapVolumes(H_{\mathcal{M}_1,\mathcal{M}_2}, \mathcal{D}_O, t_O)$     $\{O$: overlapping vol.$\}$
11             foreach $O \in \mathcal{O}$ do
12                 foreach $k \in \{1, 2\}$ do          $\{$insert $O$ into $\mathcal{D}_{\mathcal{M}_k}\}$
13                     $\mathcal{D}'_{\mathcal{M}_k} \leftarrow \mathcal{D}_{\mathcal{M}_k} \setminus \{O\}$
14                     $P'_k \leftarrow$ substitute $\langle R(\mathcal{M}_k), Sel(\mathcal{M}_k)\rangle \in C_{ij}$ in $P'$
15                           by $\langle R(\mathcal{M}_k), Sel(\mathcal{D}_{\mathcal{M}_k})\rangle$    $\{Sel(\mathcal{D}_{\mathcal{M}_k})$ : selections of $\mathcal{D}_{\mathcal{M}_k}\}$
16                 end
17                 $l \leftarrow \begin{cases} 1 & \text{if } Cost(P'_1) < Cost(P'_2) \wedge C(P'_1)/C(P) \geq t_C \\ 2 & \text{if } Cost(P'_2) < Cost(P'_1) \wedge C(P'_2)/C(P) \geq t_C \end{cases}$    $\{$best $k\}$
18                 $P' = P'_l$
19                 $\mathcal{D}_{\mathcal{M}_l} = \mathcal{D}'_{\mathcal{M}_l}$
20             end
21         end
22     end
23 end
24 return $P'$
```

Algorithm 9.3.1: Assigning overlap volumes to the mapping sets (OverlapGreedy), simplified from [Tie09].

9.4. Experimental Evaluation

We review some of the experiments of the master's thesis [Tie09] conducted in the context of our research.

9.4.1. Experimental Setup

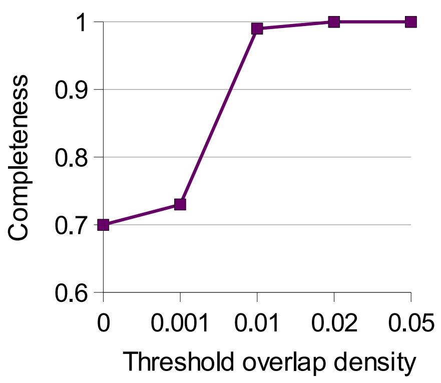
The PDMS instance has been randomly generated by the generator component of *Humboldt Peers*. The cardinality and overlap histograms are calibrated using a random training workload, which consists of 100 queries against randomly chosen peers in the PDMS. The measurement comprises 20 test queries over which the results are averaged. After the training workload the histograms are frozen since besides OVERLAPGREEDY some other strategies for distributing the overlapping volumes to the mapping sets were examined with the same state of the histograms [Tie09]. The experiments make use of the very simple cost model that each attribute value has the same cost, which is a priori estimated and fixed for all experiments with a certain PDMS instance.

	#Peers	Rank
$\mathcal{P}_{10}$	20	4
$\mathcal{P}_{11}$	20	3

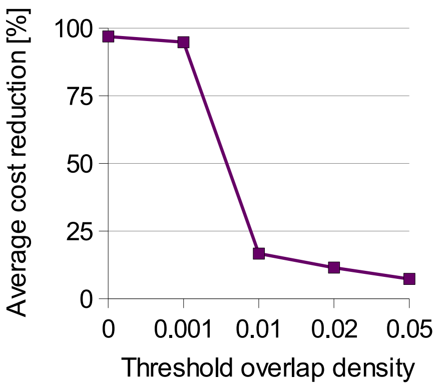
Table 9.2.: PDMS instance and its main characteristics (continued from Table 8.2 from Page 148).

9.4.2. Measurements

Experiment 8.1 (Identification of overlap.) The identification of data overlap in the multi-dimensional data space can be controlled by the threshold t_O for overlap density.
Methodology. This threshold is varied in repeated experiments with the same query workload. We are interested in how completeness and cost reduction depend on t_O. In this experiment, the completeness constraint is turned off by setting the completeness threshold $t_C = 0$ in OVERLAPGREEDY.
Discussion. The results of this experiment are depicted in Fig. 9.3. Starting from $t_O = 0$ the average completeness of the query results clearly increases towards 1 as the threshold is varied to $t_O = 0.01$. Within the same interval the actual cost savings drop below 25%. So it turns out that in this setting a threshold between 0.001 and 0.01 is the optimal choice. As the results of query optimization are heavily dependent on the choice of t_O, its values must be carefully determined.
Experiment 8.2 (Influence of completeness constraint.) This experiment examines whether our implementation actually follows the budget constraint and how different thresholds influence the efficiency of overlap-driven query optimization.
Methodology. The threshold for the completeness constraint is varied from 0 to 1 and for each value the training and test workload are applied. In each experiment, cost reduction and completeness score are measured and the resulting efficiency gain is calculated.



(a) Average completeness for OverlapGreedy.



(b) Average cost savings.

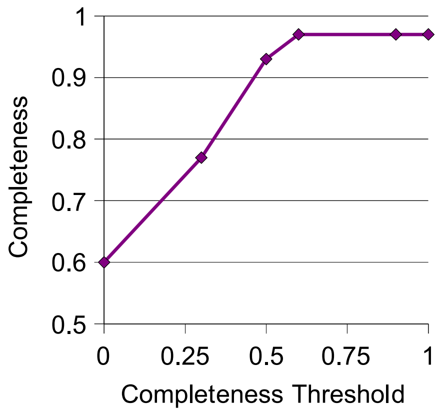
Figure 9.3.: Experiments varying the threshold for overlap identification for Pdms $\mathcal{P}_{11}$.

Discussion. The main results for the Pdms $\mathcal{P}_{10}$ documented in Table 9.2 are depicted in Fig. 9.4. The first interesting insight is that OverlapGreedy clearly follows the completeness constraint, Fig 9.4(a). For every value of the completeness threshold, the actual average completeness is much higher than the threshold. This is supposed to be due to an overestimation of the overlap by our completeness model as presented in Sec. 4.2. When estimating the cardinalities of alternative sets of mappings, this model assumes statistical independence of the sub-results returned by them although their overlap already has been reduced by partial pruning. Including overlap estimations into this cardinality estimation can potentially improve cardinality estimations and thus the effectivity of partial pruning approaches [Tie09].

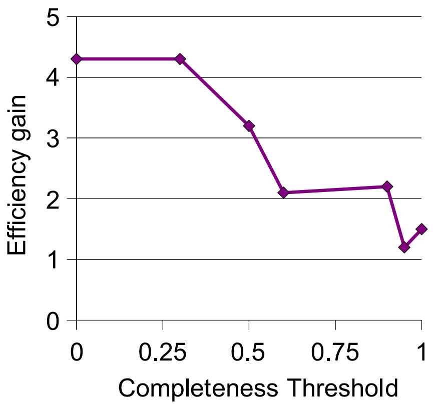
The second observation can be drawn from Fig. 9.4(b). The efficiency gain clearly drops as the completeness threshold increases. But even for a threshold of 0.95 the average efficiency is still more than a factor of 2, which is still a considerable increase. Additionally, this result can be combined with the information from Fig 9.4(a) that already a completeness threshold of 0.5, for instance, yields an average actual completeness of more than 0.9.

The Pdms instance $\mathcal{P}_{10}$ with 20 peers and a rank of 4 is comparably small. If we assume that—similar to our experience from Chapter 8—our optimization approaches yield higher efficiency gains for bigger Pdms we could extrapolate from the above result and expect even higher efficiency gains for Pdms with considerably more peers and longer mapping paths.

The experiments in [Tie09] cover several other objectives. For instance, it is examined how the results depend on the threshold of the overlap density. It also compares OverlapGreedy to other strategies, such as uniform distribution of the overlap volumes to the alternative mapping sets or an exhaustive enumeration of all possible transformed local query plans. Moreover, Tietz conducted experiments to find the break-even point, where data transport is as expensive as query planning and how it depends on the threshold for overlap density.



(a) Average completeness of the test workload by the completeness threshold for OverlapGreedy.



(b) Average efficiency of the test workload by the completeness threshold for OverlapGreedy.

Figure 9.4.: Measurements on the completeness constraint for $\mathcal{P}_{10}$.

9.5. Related Work

In the same spirit as in our work, overlap between sources accessed by mediators is considered in [VP98]. Similarly to our approach, the authors believe that in many cases users are satisfied with incomplete query answers, so-called partial answers. They propose an approach for query optimization that assumes that information about source overlap is available and does not vary over time. For unions over source relations, it employs a table of all possible source overlaps and proposes heuristics for choosing partial answer query plans. The authors recognize that these statistics have to be kept in addition at least for each class of similar queries involving selections.

In contrast, our work proposes a technique to gather statistics on overlap between neighboring peers in a *dynamic* environment. Moreover, in a PDMS maintaining these statistics is distributed among the peers. As every peer usually only knows a few neighbors, it is feasible to compare contributions of neighboring peers in a pairwise manner rather than for every possible query plan as proposed in [VP98].

The Piazza PDMS proposed to apply query containment checks during query planning to identify redundant parts of the overall query plan [TH04]. They prune a subplan if it is redundant with respect to the peers and the relations to be queried. Our approach on the one hand is different in that it compares alternative mapping paths on the data instance level rather than on the schema level. Another difference is that our approach works fully decentral whereas the Piazza technique requires information about the *global* query plan for a certain user query.

The problem of optimizing navigational queries in life sciences is covered with overlap awareness in [BKN+06]. The authors map this problem to the budgeted maximal coverage problem. We discuss the original work on that problem in Sec. 8.6.1. From the same domain, the work in [HTL07] used data overlap to rank query results.

The work in [BMT+05] applies statistics on overlap between sources for improving selection of them for peer-to-peer search engines. Interestingly, their main idea for keyword search is similar to our approach. However, their proposed solution cannot be applied to the problem of answering range queries over structured data.

9.6. Summary

The contribution of this chapter is a novel approach for PDMS query optimization that exploits information about data overlap to reduce this overlap while retrieving non-overlapping results at the same time. We formulated the problem of local cost minimization and presented a cost model that depends on the result size as well as on the size of the queries sent to neighboring peers. We analyzed how the overlap distribution depends on the properties of the PDMS. As the core of the approach, we discussed an algorithm to exclude overlapping sub-volumes from the mapping sets in the local query plan. The experimental evaluation we reported on reveals that even for PDMS instances with tens of peers the efficiency of query answering can be increased by several factors. Overlap-driven query optimization can be applied independently from the optimization approaches presented in previous chapters of this thesis.

10. Conclusion

This thesis shows that large-scale information sharing in PDMS with autonomous peers is indeed feasible if the completeness of the query result is compromised as with our best-effort optimization techniques. To finalize, it follows a summary of the contributions of this work before we propose future research directions in the context of large-scale PDMS.

10.1. Summary

The main contribution of this thesis is an exploration of a trade-off between completeness and cost of PDMS query answering achieved in a fully decentral manner. It proposes solutions to the problem of efficiently answering queries with satisfying completeness of the query results in large-scale, volatile peer data management systems consisting of autonomous peers. We return to the subproblems presented in the introduction and describe their solutions discussed throughout this thesis.

Description of semantic relationships between peers. The language for specifying semantic relationships between peer schemas is required to match to real-world peers with simple query capabilities. Therefore, we only allow selects and joins over relational schemas. The mappings between the peers are modeled as inclusion dependencies between queries of different arity. So we extend global-local-as-view mappings by projections for maximal flexibility. Since peers are not required to accept projections in queries, information loss can occur.

Constructing a query plan. For mappings having projections we introduced weak query containment mappings as a relaxed form of well-known containment mappings. Using them, we extended the MiniCon algorithm to be used for answering queries using views with projections. We provide an algorithm that transforms an intermediate rule-goal tree that reflects the usage of global-as-view and local-as-view mappings into a local query plan that contains queries to be sent to neighboring peers. Based on two different types of semantics of PDMS query answering and other architectural characteristics, we compared different PDMS approaches from the literature. We implemented our full-fledged PDMS *Humboldt Peers* as a testbed for the techniques developed in this thesis.

Estimation of query answer size. To be able to trade-off completeness and cost of query answers, an accurate, query-dependent estimate of the result size is needed. For the volatile and decentral setting of PDMS we identified self-tuning histograms as an appropriate approach for this task. This solution was compared with others from the literature. We showed how to find the dimensions to be considered in multi-dimensional histograms for mappings with projections. We also examined the influence of inaccurate cardinality statistics on the effectivity of our techniques.

Maintaining metadata for size of query answers. As the data distribution over a PDMS is highly sensitive against changes in the system, we put special emphasis on the maintenance of statistics about query answers of neighboring peers. We described how to to detect changes in the PDMS beyond the direct neighborhood of a peer solely from query feedback. Moreover, we presented a solution to the conflict between usage of query feedback for statistics maintenance and pruning of mapping paths in query planning. Here, we offered techniques with varying sensitivity that operate on different levels of granularity in the multi-dimensional data space. An extensive experimental study was conducted to prove the feasibility of result statistics maintenance.

Assessment of overlap between peers. We proposed to apply self-tuning histograms also for estimating data overlap between alternative mapping paths. This is a means to enable reduction of data overlap stemming from massive redundancy in the network of peers and the data locally stored at the peers. Similar to cardinality histograms, we described how to build overlap histograms for a pair of mapping sets containing projections.

Effective query planning. To preserve peer autonomy as much as possible, we focus on local query planning at a peer. Building on the statistics on result cardinality, we proposed a rather simple yet effective approach to prune mapping paths from a local query plan. It is based on a threshold for the potential data contribution that can be estimated using the cardinality histograms. Our experimental study revealed that the cost reduction of query answering can be reduced by two thirds compared to the full query answer. Since the completeness is still satisfying for a best-effort scenario, the efficiency of query answering is increased by about an order of magnitude or more depending on the network bandwidth and pruning strength.

Optimized retrieval of answers. Since threshold-based query planning can lead to arbitrarily poor results, we introduced approaches for query optimization operating locally at a peer but accepting a cost limit for *overall* query answering. To this end, we proposed to pass a budget along with each query that the called peers has to follow. We distinguished several strategies to spend budget. These strategies were augmented by techniques to use budget that had been refunded from neighboring peers.

An experimental parameter study examined the dependency of the effectivity of our novel approach on the size of the given budget and revealed that for PDMS with 100 peers the efficiency gain can be between one and two orders of magnitude or even more depending on the size of the budget and the network bandwidth.

All of our budget spending strategies proved to be quite robust in their efficiency against skewed data distributions and varied interconnectedness of the Pdms.

The overlap-driven optimization approach we introduced is orthogonally applicable to the budget-oriented optimization. The core of this novel solution is to partially prune mapping paths by excluding overlapping volumes of the multi-dimensional data space from queries sent to neighboring peers. We described how this technique works when peers only accept conjunctive selection queries. To preserve completeness, it follows a constraint for the minimum completeness compared to the full query plan. Experiments examined the trade-off between savings for data transport cost and cost to handle additional selections in queries. We also proved experimentally that by the overlap-driven optimization approaches the efficiency can at least be doubled even for a quite small Pdms instance with tens of peers.

Taken together, these solutions show that large-scale data sharing with Pdms is feasible. This thesis provides all necessary ingredients starting with query planning for complex mappings, over query-dependent and adaptable statistics on result cardinality and data overlap to query optimization algorithms that limit resource consumption or guarantee a certain level of completeness. Our solution addresses the problem of large-scale distributed query answering over heterogeneous and autonomous peers. There are several works in the literature that cover certain aspects of our problem. But this thesis is the first research approach that preserves peer autonomy with respect to query language, heterogeneity, and availability while returning query answers of satisfying size in acceptable response time.

10.2. Further Research Directions

Peer data management systems are especially suitable for large-scale scenarios where system structures are subject to sudden changes and users accept concessions on the completeness of query answers. We briefly discuss three main directions for future research in the context of Pdms.

Query model and semantics. The expressive power of the specification of semantic relationships between the information offerings of the peers directly influences the effort for creating and maintaining Pdms, the user skills necessary to write queries, the computing effort for query answering, and the quality of the query results. The more expressive the mapping language, the more difficult the system is to build and use, but the higher is the quality and usefulness of query answers.

In practice, it can often be observed that users are interested in easy-to-use means for information authoring, such as for unstructured information at the beginning of the information life-cycle. As a body of information grows, it becomes more desirable to apply richer query facilities to retrieve answers of higher relevancy. So the query model and its semantics is required to evolve over time to a more expressive, yet also more expensive level. Such transitions are addressed by approaches called dataspaces and pay-as-you-go information integration [HFM06, SDK+07, MJC+07, SDH08]. One can think about having different peers in the PDMS offer query services of different semantic expressivity.

Query operators and approximation. Semantic relationships between heterogeneous and autonomous peers are inherently inaccurate. This and the dynamic nature of a PDMS raise the question of what can be considered as a semantically complete query result. The best-effort query answering approach of this thesis can be extended by augmenting the query model by approximative operators as proposed in [HLS06].

Another direction is to explicitly model uncertainty of mappings and include corresponding estimations into query processing [AC03, DHY07]. This approach seems especially promising for PDMS since inaccuracies of mappings tend to accumulate along mapping paths. Furthermore, when performing local reasoning the origin of any certain data item is unclear. This is a more difficult problem than uncertainty assessment for data integration systems where the data sources are known by the integration site.

As many data sources contribute to PDMS query answering, resolution of conflicting data values is an important problem in practice. Therefore, it seems promising to examine conflict tolerant query answering [CGL+05] and information fusion [BBB+05].

Building and evolving. This thesis has focused on local reasoning. However, for many applications it is reasonable to consider the peer graph from a more global perspective to find interesting peers [PP04] or to optimize the mapping network [HHNR05]. This effort can draw from techniques for schema mapping composition [MH03, FKPT04]. Emergent semantics [ACMH03, ACMO+04] is also a field that can inspire new solutions for PDMS building and maintenance.

10.3. PDMS: Decentral and Flexible Information Systems

Peer data management systems are the most general model to study decentrally organized information sharing. Their flexible nature makes this architecture suitable for many application scenarios in today's rapidly changing world. Due to their flexibility, PDMS are a framework to provide appropriate information management functionality where it is needed.

This thesis has shown that such a decentral architecture may scale well without sacrificing flexibility and autonomy of the peers. Decentralization also promises to be an important principle to balance power between a variety of stakeholders of large-scale information systems.

Bibliography

[AC99] Aboulnaga, A.; Chaudhuri, S.: Self-tuning histograms: Building histograms without looking at data. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 1999.

[AC03] Altareva, E.; Conrad, S.: Statistical analysis as methodological framework for data(base) integration. In: *Proc. of the Int. Conf. on Conceptual Modeling (ER)*. 2003.

[ACMH03] Aberer, Karl; Cudré-Mauroux, Philippe; Hauswirth, Manfred: The Chatty Web: Emergent semantics through gossiping. In: *Proc. of the Int. World Wide Web Conf. (WWW)*. 2003.

[ACMO$^+$04] Aberer, K.; Cudre-Mauroux, P.; Ouksel, A. M.; Catarci, T.; Hacid, M.-S.; Illarramendi, A.; Kashyap, V.; Mecella, M.; Mena, E.; Neuhold, E. J.; Troyer, O. De; Risse, T.; Scannapieco, M.; Saltor, F.; de Santis, L.; Spaccapietra, S.; Staab, S.; Studer, R.: Emergent semantics Principles and issues. In: *Proc. of International Conference on Database Systems for Advanced Applications (DASFAA)*. 2004.

[AD98] Abiteboul, S.; Duschka, O.: Complexity of answering queries using materialized views. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 1998.

[AHV95] Abiteboul, Serge; Hull, Richard; Vianu, Victor: *Foundations of Databases*. Addison Wesley, 1995.

[ANR07] Albrecht, Alexander; Naumann, Felix; Roth, Armin: Networked PIM using PDMS. In: *Proc. of the Workshop on Networking Meets Databases (NetDB)*. 2007.

[ASS03] Akella, Aditya; Seshan, Srinivasan; Shaikh, Anees: An empirical evaluation of wide-area internet bottlenecks. In: *Proc. of the Internet Measurement Conference*. 2003.

[BBB$^+$05] Bilke, Alexander; Bleiholder, Jens; Böhm, Christoph; Draba, Karsten; Naumann, Felix; Weis, Melanie: Automatic data fusion with HumMer. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 2005.

[BCG01] Bruno, N.; Chaudhuri, S.; Gravano, L.: STHoles: a multidimensional workload-aware histogram. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 2001.

[BGK$^+$02] Bernstein, P. A.; Giunchiglia, F.; Kementsietsidis, A.; Mylopoulos, J.; Serafini, L.; Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. In: *Proc. of the ACM SIGMOD Workshop on The Web and Databases (WebDB)*. 2002.

[BH03] Brown, Paul G.; Haas, Peter J.: BHUNT: Automatic discovery of fuzzy algebraic constraints in relational data. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 2003.

[BKN$^+$06] Bleiholder, Jens; Khuller, Samir; Naumann, Felix; Raschid, Louiqa; Wu, Yao: Query planning in the presence of overlapping sources. In: *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*. 2006.

[BMT$^+$05] Bender, Matthias; Michel, Sebastian; Triantafillou, Peter; Weikum, Gerhard; Zimmer, Christian: Improving collection selection with overlap awareness in P2P search engines. In: *Proc. of the ACM Int. Conf. on Research and Development in Information Retrieval (SIGIR)*. 2005.

[CCGL04] Cali, A.; Calvanese, D.; Giacomo, G. De; Lenzerini, M.: Data integration under integrity constraints. In: *Information Systems*, volume 29(2):pp. 147–163, 2004.

[CGL$^+$05] Calvanese, Diego; Giacomo, Giuseppe De; Lembo, Domenico; Lenzerini, Maurizio; Rosati, Riccardo: Inconsistency tolerance in P2P data integration: an epistemic logic approach. In: *Proc. of the Int. Workshop on Database Programming Languages*. 2005.

[CGLR04] Calvanese, Diego; Giacomo, Giuseppe De; Lenzerini, Maurizio; Rosati, Riccardo: Logical foundations of peer-to-peer data integration. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 2004.

[CM77] Chandra, A. K.; Merlin, P. M.: Optimal implementation of conjunctive queries in relational databases. In: *Proc. of the ACM Symposium on Theory of Computing (STOC)*. 1977.

[Cod79] Codd, E.F.: Extending the relational database model to capture more meaning. In: *ACM Transactions on Database Systems (TODS)*, volume 4(4):pp. 397–434, 1979.

[CR94] Chen, C. M.; Roussopoulos, N.: Adaptive selectivity estimation using query feedback. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 1994.

[DGR01] Deshpande, Amol; Garofalakis, Minos; Rastogi, Rajeev: Independence is good: Dependency-based histogram synopses for high-dimensional data. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 2001.

[DH02] Doan, Anhai; Halevy, Alon: Efficiently ordering query plans for data integration. In: *Proc. of the Int. Conf. on Data Engineering (ICDE)*. 2002.

[DHM05] Dong, Xin; Halevy, Alon; Madhavan, Jayant: Reference reconciliation in complex information spaces. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 2005.

[DHY07] Dong, Xin; Halevy, Alon; Yu, Cong: Data integration with uncertainty. In: *VLDB*. 2007.

[Dus97] Duschka, O.: Query optimization using local completeness. In: *Proc. of the National Conf. on Artificial Intelligence (AAAI)*. 1997.

[EKMR06] Ewen, Stephan; Kache, Holger; Markl, Volker; Raman, Vijayshankar: Progressive query optimization for federated queries. In: *EDBT*. 2006.

[EN00] Elmasri, R.; Navathe, S. B.: *Fundamentals of Database Systems*. Addision-Wesley, 3rd edition, 2000.

[FHP$^+$02] Fagin, R.; Hernandez, M.; Popa, L.; Velegrakis, Y.; Miller, R. J.: Translating web data. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 2002.

[FK97] Faloutsos, C.; Kamel, I.: Relaxing the uniformity and independence assumptions using the concept of fractal dimension. In: *Journal of Computer and Systems Sciences*, volume 55(2):pp. 229–240, 1997.

[FKL97] Florescu, Daniela; Koller, Daphne; Levy, Alon Y.: Using probabilistic information in data integration. In: *Proc. of the 23rd VLDB Conference, Athens, Greece*. 1997.

[FKLS03] Franconi, Enrico; Kuper, Gabriel; Lopatenko, Andrei; Serafini, Luciano: A robust logical and computational characterisation of peer-to-peer database systems. In: *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*. 2003.

[FKPT04] Fagin, R.; Kolaitis, P.; Popa, L.; Tan, W.-C.: Composing schema mappings: Second-order dependencies to the rescue. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 2004.

[GHI$^+$01] Gribble, S.; Halevy, A.; Ives, Z.; Rodrig, M.; ; Suciu, D.: What can databases do for peer-to-peer? In: *Proc. of the ACM SIGMOD Workshop on The Web and Databases (WebDB)*. 2001.

[GK03] Grahne, G.; Kiricenko, V.: Partial answers in information integration systems. In: *Proc. of the Int. Workshop on Web Information and Data Management (WIDM)*. 2003.

[GLLR07] Giacomo, Giuseppe De; Lembo, Domenico; Lenzerini, Maurizio; Rosati, Riccardo: On reconciling data exchange, data integration, and peer data management. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 2007.

[Gra02] Grahne, G.: Information integration and incomplete information. In: *IEEE Data Engineering Bulletin*, volume 25(3), 2002.

[GRZZ00] Gruser, J.R.; Raschid, L.; Zadorozhny, V.; Zhan, T.: Learning response time for websources using query feedback and application in query optimization. In: *VLDB Journal*, volume 9(1), 2000.

[GTK01] Getoor, Lise; Taskar, Ben; Koller, Daphne: Selectivity estimation using probabilistic models. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 2001.

[Ḧ06] Hübner, Tobias: *Entwicklung einer Testumgebung für ein Peer Data Management System (Development of a testbed for a Peer Data Management System)*. Master's thesis, Humboldt-Universität zu Berlin, 2006. In German.

[Hal01] Halevy, A. Y.: Answering queries using views: A survey. In: *VLDB Journal*, volume 10(4), 2001.

[HFM06] Halevy, Alon; Franklin, Michael; Maier, David: Principles of dataspace systems. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 2006.

[HHNR05] Heese, Ralf; Herschel, Sven; Naumann, Felix; Roth, Armin: Self-extending peer data management. In: *Proc. of the Conf. Datenbanksysteme in Business, Technologie und Web (BTW)*. Karlsruhe, Germany, 2005.

[HIMT03] Halevy, Alon Y.; Ives, Zachary; Mork, Peter; Tatarinov, Igor: Piazza: Data management infrastructure for semantic web applications. In: *Proc. of the Int. World Wide Web Conf. (WWW)*. 2003.

[HIST03] Halevy, Alon Y.; Ives, Zachary; Suciu, Dan; Tatarinov, Igor: Schema mediation in peer data management systems. In: *Proc. of the Int. Conf. on Data Engineering (ICDE)*. 2003.

[HKWY97] Haas, L. M.; Kossmann, D.; Wimmers, E. L.; Yang, J.: Optimizing queries across diverse data sources. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 1997.

[HLS06] Hose, Katja; Lemke, Christian; Sattler, Kai-Uwe: Processing relaxed skylines in PDMS using distributed data summaries. In: *Proc. of the Conf. on Information and Data Management (CIKM)*. 2006.

[Hos09] Hose, Katja: *Processing Rank-Aware Queries in Schema-Based P2P Systems.* Ph.D. thesis, Technische Universität Ilmenau, 2009.

[HRZ$^+$08] Hose, Katja; Roth, Armin; Zeitz, Andre; Sattler, Kai-Uwe; Naumann, Felix: A research agenda for query processing in large-scale peer data management systems. In: *Information Systems*, volume 33(7-8):pp. 597–610, 2008.

[HTL07] Hussels, Philipp; Trißl, Silke; Leser, Ulf: What's new? What's certain? – Scoring search results in the presence of overlapping data sources. In: *Proc. of the Int. Workshop on Data Integration for the Life Sciences (DILS).* 2007.

[IMH$^+$04] Ilyas, Ihab F.; Markl, Volker; Haas, Peter; Brown, Paul; Aboulnaga, Ashraf: CORDS: Automatic discovery of correlations and soft functional dependencies. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 2004.

[Ioa03] Ioannidis, Y.: The history of histograms (abridged). In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 2003.

[IP95] Ioannidis, Y.; Poosala, V.: Balancing histogram optimality and practicality for query result size estimation. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 1995.

[Kha08] Kharlamov, Evgeny: Incompleteness in information integration. In: *Proc. of the VLDB 2008 PhD Workshop.* 2008.

[Kir03] Kiricenko, Victoria: *Partial Answers in Information Integration Systems: Their Meaning and Computation.* Master's thesis, Concordia University Montreal, Canada, 2003.

[KMN99] Khuller, S.; Moss, A.; Naor, J.S.: The budgeted maximum coverage problem. In: *Information Processing Letters*, volume 70(1):pp. 39–45, 1999.

[Kos00] Kossman, D.: The state of the art in distributed query processing. In: *ACM Computing Surveys*, volume 32(4):pp. 422–469, 2000.

[KPPT05] Koloniari, G.; Petrakis, Y.; Pitoura, E.; Tsotsos, T.: Query workload-aware overlay construction using histograms. In: *Proc. of the Conf. on Information and Data Management (CIKM).* 2005.

[Lem07] Lemke, Christian: *Verwaltung von Datenzusammenfassungen in PDMS (Maintenance of data summaries in PDMS).* Master's thesis, Technische Universität Ilmenau, 2007. In German.

[Len02] Lenzerini, Maurizio: Data integration: A theoretical perspective. In: *Proc. of the Symposium on Principles of Database Systems (PODS).* 2002.

[Les00] Leser, Ulf: *Query Planning in Mediator Based Information Systems*. Ph.D. thesis, Technische Universität Berlin, 2000.

[Lib07] Libkin, Leonid: Data exchange and incomplete information. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 2007.

[LMSS95] Levy, A.Y.; Mendelzon, A. O.; Sagiv, Y.; Srivastava, D.: Answering queries using views. In: *Proc. of the Symposium on Principles of Database Systems (PODS)*. 1995.

[LN90] Lipton, R. J.; Naughton, J. F.: Practical selectivity estimation using adaptive sampling. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. 1990.

[LNWS03] Löser, Alexander; Nejdl, Wolfgang; Wolpers, Martin; Siberski, Wolf: Information integration in schema-based peer-to-peer networks. In: *Proc. of the Conf. on Advanced Information Systems Engineering (CAiSE)*. 2003.

[LRO96] Levy, Alon Y.; Rajaraman, Anand; Ordille, Joann J.: Querying heterogeneous information sources using source descriptions. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 1996.

[MBDH02] Madhavan, J.; Bernstein, P. A.; Domingos, P.; Halevy, A. Y.: Representing and reasoning about mappings between domain models. In: *Proc. of the National Conf. on Artificial Intelligence (AAAI)*. 2002.

[MH03] Madhavan, J.; Halevy, A. Y.: Composing mappings among data sources. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 2003.

[MHK$^+$07] Markl, V.; Haas, P. J.; Kutsch, M.; Megiddo, N.; Srivastava, U.; Tran, T. M.: Consistent selectivity estimation via maximum entropy. In: *VLDB Journal*, volume 16(1):pp. 55–76, 2007.

[MJC$^+$07] Madhavan, J.; Jeffery, S. R.; Cohen, S.; Dong, X.; Ko, D.; Yu, C.; Halevy, A. Y.: Web-scale data integration: You can only afford to pay as you go. In: *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*. 2007.

[MMK$^+$05] Markl, V.; Megiddo, N.; Kutsch, M.; Tran, T. Minh; Haas, P.; Srivastava, U.: Consistently estimating the selectivity of conjuncts of predicates. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB)*. 2005.

[MPS98] Muthukrishnan, S.; Poosala, V.; Suel, T.: Partitioning two dimensional arrays: algorithms, complexity and applications. In: *Proc. of the Int. Conf. on Database Theory (ICDT)*. 1998.

[Nau02] Naumann, F.: *Quality-driven query answering for integrated information systems*. Number 2261 in Lecture Notes in Computer Science. Springer, 2002.

[New88] Newcombe, H.B.: *Handbook of Recond Linkage.* Oxford University Press, Oxford, 1988.

[NFL04] Naumann, Felix; Freytag, Johann-Christoph; Leser, Ulf: Completeness of integrated information sources. In: *Information Systems*, volume 29(7):pp. 583–615, 2004.

[NGT98] Naacke, Hubert; Gardarin, Georges; Tomasic, Anthony: Leveraging mediator cost models with heterogeneous data sources. In: *Proc. of the Int. Conf. on Data Engineering (ICDE).* 1998.

[NK01] Nie, Z.; Kambhampati, S.: Joint optimization of cost and coverage of query plans in data integration. In: *Proc. of the Conf. on Information and Data Management (CIKM).* 2001.

[NK04] Nie, Zaiqing; Kambhampati, Subbarao: A frequency-based approach for mining coverage statistics in data integration. In: *Proc. of the Int. Conf. on Data Engineering (ICDE).* 2004.

[NKH03] Nie, Zaiqing; Kambhampati, Subbarao; Hernandez, Thomas: Bibfinder/Statminer: Effectively mining and using coverage and overlap statistics in data integration. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 2003. Demonstration.

[NKN05] Nie, Zaiqing; Kambhampati, Subbarao; Nambiar, Ullas: Effectively mining and using coverage and overlap statistics for data integration. In: *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, volume 17(5), 2005.

[NL06] Naumann, Felix; Leser, Ulf: *Informationsintegration.* dpunkt.verlag, 2006. ISBN 10 3-89864-400-6. In German.

[NLF99] Naumann, F.; Leser, U.; Freytag, J. C.: Quality-driven integration of heterogenous information systems. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 1999.

[NR06] Naumann, Felix; Raschid, Louiqa: Information integration and disaster data management (DisDM). `http://db.cis.upenn.edu/iiworkshop/postworkshop/positionPapers/123.pdf`, 2006. Position paper.

[NR07] Naumann, Felix; Roth, Armin: Peer-Daten-Management-Systeme – PDMS. In: *Datenbank-Spektrum*, volume 7(23):pp. 57–59, 2007. In German.

[OB04] Ouzzani, Mourad; Bouguettaya, Athman: Query processing and optimization on the Web. In: *Distributed and Parallel Databases*, volume 15(3):pp. 187–218, 2004.

[OV11] Özsu, M. T.; Valduriez, P.: *Principles of distributed database systems.* Prentice Hall, 3rd edition, 2011.

[PAGM96] Papakonstantinou, Y.; Abiteboul, S.; Garcia-Molina, H.: Object fusion in mediator systems. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 1996.

[PFPG02] Peim, M.; Franconi, E.; Paton, N. W.; Gobble, C. A.: Querying objects with description logics. In: *Proc. of the Int. Workshop on Knowledge Represenation Meets Databases (KRDB).* 2002.

[PI97] Poosala, V.; Ioannidis, Y.: Selectivity estimation without value independence. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 1997.

[PKP04] Petrakis, Yannis; Koloniari, Georgia; Pitoura, Evaggelia: On using histograms as routing indexes in peer-to-peer systems. In: *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P).* 2004.

[PL00] Pottinger, R.; Levy, A. Y.: A scalable algorithm for answering queries using views. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 2000.

[PP04] Petrakis, Y.; Pitoura, E.: On constructing small worlds in unstructured peer-to-peer systems. In: *EDBT Workshops '04*, pp. 415–424. 2004.

[RN05] Roth, Armin; Naumann, Felix: Benefit and cost of query answering in PDMS. In: *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, number LNCS in 4125. 2005.

[RNHS06] Roth, Armin; Naumann, Felix; Hübner, Tobias; Schweigert, Martin: System P: Query answering in PDMS under limited resources. In: *Proc. of the Workshop on Information Integration on the Web (IIWeb).* 2006.

[ROH99] Roth, Mary Tork; Özcan, Fatma; Haas, Laura M.: Cost models DO matter: Providing cost information for diverse data sources in a federated system. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 1999.

[Rot07] Roth, Armin: Completeness-driven query answering in peer data management systems. In: *Proc. of the VLDB 2007 PhD Workshop.* 2007.

[SAP$^+$96] Stonebraker, Michael; Aoki, Paul M.; Pfeffer, Avi; Sah, Adam; Sidell, Jeff; Staelin, Carl; Yu, Andrew: Mariposa: A wide-area distributed database system. In: *VLDB Journal*, volume 5, pp. 48–63. 1996.

[Sch06] Schweigert, Martin: *Entwurf eines Peer Data Management Systems mit Steuerungs- und Simulationskomponente (Design of a Peer Data Management System with control and simulaton component).* Master's thesis, Humboldt-Universität zu Berlin, 2006. In German.

[SDH08] Sarma, Anish Das; Dong, Luna; Halevy, Alon: Bootstrapping pay-as-you-go data integration systems. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 2008.

[SDK+07] Salles, Marcos Antonio Vaz; Dittrich, Jens-Peter; Karakashian, Shant Kirakos; Girard, Olivier Renz; Blunschi, Lukas: iTrails: Pay-as-you-go information integration in dataspaces. In: *Proc. of the Int. Conf. on Very Large Databases (VLDB).* 2007.

[SHM+06] Srivastava, U.; Haas, P.; Markl, V.; Megiddo, N.; Kutsch, M.; Tran, T.: ISOMER: Consistent histogram construction using query feedback. In: *Proc. of the Int. Conf. on Data Engineering (ICDE).* 2006.

[SS06] Staab, Steffen; Stuckenschmidt, Heiner, editors: *Semantic Web and Peer-to-Peer – Decentralized Management and Exchange of Knowledge and Information.* Springer, 2006.

[SSR07] Schütt, Thorsten; Schintke, Florian; Reinefeld, Alexander: A structured overlay for multi-dimensional range queries. In: *Proc. of Euro-Par 2007 Parallel Processing*, volume 4641 of *LNCS*, pp. 503–513. Springer, 2007.

[SW05] Steinmetz, Ralf; Wehrle, Klaus, editors: *Peer-to-Peer Systems and Applications.* Number 3485 in LNCS. Springer, 2005.

[Tat04] Tatarinov, Igor: *Semantic Data Sharing with a Peer Data Management System.* Ph.D. thesis, University of Washington, 2004.

[TH04] Tatarinov, Igor; Halevy, Alon Y.: Efficient query reformulation in peer data management systems. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 2004.

[Tie06] Tietz, Véronique: Parallelisierung der Anfragebearbeitung in einem Peer Data Management System (Parallelization of query processing in a peer data management system). Student research project, Humboldt-Universität zu Berlin, 2006. In German.

[Tie09] Tietz, Véronique: *Nutzung von Statistiken über Daten-Overlap zur Anfrageoptimierung in Peer Data Management Systemen (Usage of statistics on data overlap for query optimization in PDMS).* Master's thesis, Humboldt-Universität zu Berlin, 2009. In German.

[TIGK02] Thaper, N.; Indyk, P.; Guha, S.; Koudas, N.: Dynamic multidimensional histograms. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 2002.

[TIM+03] Tatarinov, I.; Ives, Z. G.; Madhavan, J.; Halevy, A. Y.; Suciu, D.; Dalvi, N. N.; Dong, X.; Y. Kadiyska, G. Miklau; Mork, P.: The Piazza peer data management project. In: *SIGMOD Record*, volume 32(3):pp. 47–52, 2003.

[Ull88] Ullman, J. D.: *Principles of Database and Knowledge-Base Systems, Vol I: Classical Database Systems.* Computer Science Press, New York, NY, 1988.

[Ull97] Ullman, Jeffrey D.: Information integration using logical views. In: *Proc. of the Int. Conf. on Database Theory (ICDT).* 1997.

[VP98] Vassalos, Vasilis; Papakonstantinou, Yannis: Using knowledge of redundancy for query optimization in mediators. In: *AAAI Workshop on AI and Information Integration.* 1998.

[Wie92] Wiederhold, G.: Mediators in the architecture of future information systems. In: *IEEE Computer*, volume 25(3):pp. 38–49, 1992.

[WN05] Weis, M.; Naumann, F.: DogmatiX tracks down duplicates in XML. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 2005.

[Zip49] Zipf, G. K.: *Human behavior and the principle of least effort.* Addison-Wesley, 1949.

[ZL96] Zhu, Q.; Larson, P.A.: Developing regression cost models for multidatabase systems. In: *Proc. of the IEEE Int. Conf. on Parallel and Distributed Information Systems(PDIS).* 1996.

[ZRV$^+$02] Zadorozhny, Vladimir; Raschid, Louiqa; Vidal, Maria-Esther; Urhan, Tolga; Bright, Laura: Efficient evaluation of queries in a mediator for websources. In: *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD).* 2002.

[ZRZB01] Zadorozhny, Vladimir; Raschid, Louiqa; Zhan, Tao; Bright, Laura: Validating an access cost model for wide area applications. In: *Proc. of the Int. Conf. on Cooperative Information Systems (CoopIS).* 2001.

Selbständigkeitserklärung

Ich erkläre hiermit, daß

– ich die vorliegende Dissertationsschrift “Efficient Query Answering in Peer Data Management Systems” selbständig, ohne unerlaubte Hilfe und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe,

– ich mich nicht anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze, und

– mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist, gemäß Amtl. Mitteilungsblatt Nr. 34/2006.

Ulm, den 22.11.2010 Armin Roth